A Modern View of Classical Cryptography

Thomas Kaeding

© 2024 Thomas Kaeding

The ciphertexts, and corresponding plaintexts, for some exercises are from the British National Cipher Challenge and are copyright by the University of Southampton. They are used with permission.

Scrabble™ is a trademark of Hasbro in the United States and Canada, and of Mattel elsewhere.

Many of the example texts are from stories and novels that are no longer under copyright.

Photo credits

Still frame from the TV show *Futurama*
Inside cover from Ozzy Osbourne's album *Speak of the Devil*
Wadsworth cipher disk: U.S. National Security Agency
Wheatstone Cryptograph: eBay
Urkryptografen: Museum of Cypher Equipment, Fife, Scotland
Bazeries cylinder: Étienne Bazeries
M-94: robbo@ev1.net
M-138: U.S. Department of Defense
Enigma: Greg Goebel
Enigma rotors: Ted Coles
SIGABA: United States Air Force
Fialka and rotor: Tom Perera (EnigmaMuseum.com)
Siemens and Halske T52d: United States National Security Agency
Lorenz SZ40: United States government
Purple: United States Air Force

Acknowledgments

The British National Cipher Challenge: https://www.cipherchallenge.org

American Cryptogram Association: https://www.cryptogram.org

Mystery Twister C3: https://mysterytwister.org

Project Gutenberg: https://gutenberg.org

Preface

This book is an introduction to classical cryptography, with an emphasis on cryptanalysis. By *classical*, we mean cryptography that can be done with pen and paper. Historically, such ciphers were used for serious secret-keeping up to and into the Second World War, around which time mechanical ciphers came into use. Nevertheless, classical ciphers continue to be invented, even though the classical period has ended. You may have often heard that classical ciphers are insecure because modern computers can break them easily. Well, now you will learn how.

This is an expansion and slight reorganization of the book *A Book on Classical Cryptography* "by madness," which was written in 2020 to summarize what this author had learned in the years preceeding it. Since then, the book has been occasionally modified, but not in any large way. The time has come, however, to add some topics, such as group theory for substitution ciphers, some analysis of the Hutton ciphers, and a description of a few of the cipher machines in use before computers. We have kept to the organizational practice of presenting ideas in small chunks, rather than in a traditional textbook format in which chapters are long and tiring.

The overall layout of this book is to begin by looking at the characteristics of language that will help us later to break ciphers (Part I). Then we look at substitution ciphers (Parts II and III), transposition ciphers (Part IV), ciphers involving grids (Part V), the Hill cipher (based on matrices) (Part VI), stream ciphers (Part VII), ciphers based on changing the representation of letters (codes) (Part VIII), miscellaneous ciphers (Part IX), and ciphers based on devices (Parts X and XI). Warning: even though a cipher may appear in the miscellaneous section, it may still be important. A serious student can ignore most of that part of the book, but not symbolic substitution and homophonic substitution, which are historically important, and the one-time pad, which is relevant in modern and quantum cryptography.

Everything in this book will be done in English (except for a rare word or two every now and then). Nevertheless, most if not all of what we learn here can be used for any language that uses the Latin alphabet. The only modifications necessary are in the linguistic data. For another language, Part I on linguistic data can be reworked in the new language to compile a new set of data for use by the methods of the latter parts of the book.

Although classical cryptography can be done with pen and paper, this does not mean that we should use pen and paper. An emphasis in this book is on the use of a computer to cryptanalyze a ciphertext. To that end, we recommend the Python language for its ease of manipulating text. (In modern cryptography, Python is also useful because it handles large integers seamlessly.) Since version 2 of the language is no longer maintained, any tips and examples in this book will use version 3 of Python. There are only three differences between these versions that are important for us: In version 3, strings of characters and strings of bytes are different types of data. In version 3, the `print` statement has been replaced by a `print()` function, so that now parentheses are necessary. In version 2, the `/` operator behaved differently if it was dividing integers or floating-point numbers; in version 3 we will use two operators (`/` and `//`) for division. For this book, you should be able to write rudimentary

Python scripts. We will provide tips as we go along, and you should expect to learn more about the language as we go. You will not need to be able to write object-oriented scripts, but defining a cipher as a class of objects is something we may ask you to try occasionally.

It is best to work through each unit before continuing onward, since units build on one another, at least until the end of Part IV. If you are not interested in the abstract mathematics, you can skip the units involving group theory in Parts II and III, and the units involving directed graphs in Part X. Parts V through VIII are each roughly self-contained. Part IX contains optional pieces that can be done in any order if wished, with few exceptions.

Along the way, you will be building your own library of functions and programs in Python for cracking ciphertexts. These can help you if you enjoy on-line challenges, such as the British National Cipher Challenge (BNCC) (www.cipherchallenge.org) or MysteryTwister C3 (mysterytwister.org).

If you don't want to learn anything fancy, a fine course of study would be Units 1, 3-6, 9-13, 15, 19, 20-22, 24, 26, 27, 30-39, 47-52, 54-56, 83-91, 93, 94, 97, 106-109, 120, 121, 123, 124, 126, 127, 129-132, 134-138, 141-146, 153, and 154. That would leave you in good stead to tackle something like the BNCC.

Now get to work!

A note on pronouns

Your pronouns are you/your/yourself. If you grew up in the 1500s, and this bothers you, then just replace "you" with "thou" in thy head as thou readest this book.

Contents

Part 0: Preliminaries

Unit 1: Cryptographic systems
Unit 2: Eras of cryptography

Part I: Linguistic data

Unit 3: Textual corpus
Unit 4: Word lists
Unit 5: Monogram frequency tables
Unit 6: Tetragram frequency tables
Unit 7: The χ^2 statistic
Unit 8: Monogram fitness with the χ^2 statistic
Unit 9: Angle between vectors
Unit 10: Monogram fitness with the angle between vectors
Unit 11: Tetragram fitness
Unit 12: Index of coincidence
Unit 13: *n*-gram index of coincidence
Unit 14: Entropy

Part II: Monoalphabetic substitution

Unit 15: Monoalphabetic substitution ciphers
Unit 16: Atbash cipher
Unit 17: Mathematical groups
Unit 18: Alphabetic permutations
Unit 19: Modular arithmetic: addition and subtraction
Unit 20: Caesar shift cipher
Unit 21: Brute-force attack on the Caesar cipher
Unit 22: Attacking the Caesar cipher with cribs
Unit 23: Attacking the Caesar cipher with monogram frequencies (χ^2)
Unit 24: Attacking the Caesar cipher with monogram frequencies ($\cos \theta$)
Unit 25: Caesar salad cipher
Unit 26: Greatest common divisor
Unit 27: Modular arithmetic: multiplication and division
Unit 28: Multiplication cipher
Unit 29: Group isomorphism
Unit 30: Affine cipher
Unit 31: Brute-force attack on the affine cipher
Unit 32: Attacking the affine cipher with cribs
Unit 33: Attacking the affine cipher with monogram frequencies

Unit 34: Keyword substitution cipher
Unit 35: Dictionary attack on the keyword substitution cipher
Unit 36: Hill-climbing attack on monoalphabetic substitution ciphers

Part III: Polyalphabetic substitution

Unit 37: Periodic polyalphabetic substitution ciphers
Unit 38: Finding the period: Kasiski method
Unit 39: Finding the period: Sinkov method
Unit 40: Finding the period: twist method
Unit 41: Fuer GOD cipher
Unit 42: Brute-force attack on the Fuer GOD cipher
Unit 43: Attacking the Fuer GOD cipher with cribs
Unit 44: Dictionary attack on the Fuer GOD cipher
Unit 45: Hill-climbing attack on the Fuer GOD cipher
Unit 46: Attacking the Fuer GOD cipher with monogram frequencies
Unit 47: Vigenère cipher
Unit 48: Brute-force attack on the Vigenère cipher
Unit 49: Attacking the Vigenère cipher with cribs
Unit 50: Dictionary attack on the Vigenère cipher
Unit 51: Hill-climbing attack on the Vigenère cipher
Unit 52: Attacking the Vigenère cipher as a periodic Caesar cipher
Unit 53: Gronsfeld cipher
Unit 54: Beaufort cipher
Unit 55: Variant Beaufort cipher
Unit 56: Porta cipher
Unit 57: Bellaso 1552 cipher
Unit 58: Periodic affine cipher
Unit 59: Attacking the periodic affine cipher with monogram frequencies
Unit 60: Quagmire ciphers
Unit 61: Attacking quagmire ciphers with cribs: traditional method
Unit 62: Quagmire 1 cipher
Unit 63: Two-stage attack on the quagmire 1 cipher
Unit 64: Attacking quagmire 1 with cribs: symmetry
Unit 65: Attacking quagmire 1 with cribs: group theory
Unit 66: Quagmire 1: recovering keywords
Unit 67: Quagmire 2 cipher
Unit 68: Better dictionary attack on quagmire 2
Unit 69: Attacking quagmire 2 with cribs: symmetry
Unit 70: Attacking quagmire 2 with cribs: group theory
Unit 71: Quagmire 2: recovering keywords
Unit 72: Quagmire 3 cipher
Unit 73: Better dictionary attack on quagmire 3
Unit 74: Attacking quagmire 3 with cribs: symmetry
Unit 75: Attacking quagmire 3 with cribs: group theory
Unit 76: Quagmire 3: recovering keywords
Unit 77: Quagmire 4 cipher
Unit 78: Better dictionary attack on quagmire 4
Unit 79: Attacking quagmire 4 with cribs: symmetry

Unit 80: Attacking quagmire 4 with cribs: group theory
Unit 81: Quagmire 4: recovering keywords
Unit 82: Some other ciphers are also quagmires
Unit 83: Hill-climbing attack on periodic polyalphabetic substitution ciphers

Part IV: Transposition ciphers

Unit 84: Transposition ciphers
Unit 85: Permutations
Unit 86: Permutation cipher
Unit 87: Heap's algorithm
Unit 88: Factoradic numbers and permutations
Unit 89: Brute-force attack on the permutation cipher
Unit 90: Hill-climbing attack on the permutation cipher
Unit 91: Matrix transposition
Unit 92: Twisted scytale
Unit 93: Columnar transposition cipher
Unit 94: Double columnar transposition cipher
Unit 95: Improved hill-climbing attack on the double columnar transposition cipher
Unit 96: Nihilist transposition cipher
Unit 97: Railfence cipher
Unit 98: Redefence cipher
Unit 99: AMSCO cipher
Unit 100: Myszkowsky cipher
Unit 101: Cadenus cipher
Unit 102: Hill-climbing attack on the Cadenus cipher

Part V: Grid-based ciphers

Unit 103: Polybius square
Unit 104: Polybius cipher
Unit 105: Playfair cipher
Unit 106: Hill-climbing attack on the Playfair cipher
Unit 107: Vertical two-square cipher
Unit 108: Horizontal two-square cipher
Unit 109: Hill-climbing attack on two-square ciphers
Unit 110: Four-square cipher
Unit 111: Phillips cipher
Unit 112: Hill-climbing attack on the Phillips cipher
Unit 113: Phillips-RC cipher
Unit 114: Double Playfair cipher
Unit 115: Nihilist substitution cipher
Unit 116: Attack on the Nihilist substitution cipher
Unit 117: Bifid cipher
Unit 118: Attack on the bifid cipher
Unit 119: CM bifid cipher
Unit 120: Trifid cipher
Unit 121: ADFGX cipher
Unit 122: ADFGVX cipher

Part VI: Matrices

Unit 123: Matrices and vectors
Unit 124: Inverting a matrix: row operations
Unit 125: Inverting a matrix: adjugate matrix
Unit 126: Matrices over the set of residues
Unit 127: Hill cipher
Unit 127½: Brute-forcing Hill ciphers with less brute force
Unit 128: Attacking the Hill cipher with cribs: linear equations
Unit 129: Attacking the Hill cipher with cribs: matrices
Unit 130: Affine Hill cipher
Unit 130½: Brute-forcing affine Hill ciphers with less brute force

Part VII: Stream ciphers

Unit 131: Stream ciphers
Unit 132: Trithemius cipher
Unit 133: Autokey cipher
Unit 134: Hill-climbing attack on the autokey cipher
Unit 135: Attacking the autokey cipher with monogram frequencies
Unit 136: Running-key cipher
Unit 137: Progressive Vigenère cipher
Unit 138: Solitaire cipher
Unit 139: Hill-climbing attack on the solitaire cipher with partially known key

Part VIII: Codes and ciphers

Unit 140: Codes
Unit 141: Baconian code
Unit 142: Triliteral code
Unit 143: Morse code
Unit 144: Fractionated Morse cipher
Unit 145: Morbit cipher
Unit 146: Monome-dinome cipher
Unit 147: Straddling checkerboard cipher
Unit 148: Prefix-free codes
Unit 149: Huffman code

Part IX: Miscellaneous ciphers

Unit 150: Symbolic substitution
Unit 151: One-time pad
Unit 152: Slidefair cipher
Unit 153: Nicodemus cipher
Unit 154: Hutton 1 cipher
Unit 155: Attacking Hutton 1 with cribs
Unit 156: Hutton 2 cipher
Unit 157: Attacking Hutton 2 with cribs

Unit 158: Scrabble cipher
Unit 159: Homophonic substitution
Unit 160: Polyphonic substitution
Unit 161: Polyhomophonic substitution
Unit 162: Pollux cipher
Unit 163: Doubled-over substitution
Unit 164: Duplicitous ciphers
Unit 165: Combination-lock cipher
Unit 166: Chase cipher

Part X: Proto-mechanical ciphers

Unit 167: Disk ciphers (cipher clocks)
Unit 168: Attacking cipher clocks with cribs
Unit 169: Digram-counting attack on cipher clocks
Unit 170: Graphs
Unit 171: Hamiltonian paths
Unit 172: Attacking cipher clocks with directed graphs
Unit 173: Hill-climbing attack on cipher clocks
Unit 174: Cylinder ciphers
Unit 175: Hill-climbing attack on cylinders
Unit 176: Strip ciphers

Part XI: Cipher machines

Unit 177: Enigma
Unit 178: Attacking Enigma with the index of coincidence
Unit 179: Attacking Enigma with cribs (Bombe)
Unit 180: SIGABA
Unit 181: Fialka
Unit 182: Boolean variables and logical operations
Unit 183: Exclusive or
Unit 184: Baudot encoding
Unit 185: Siemens and Halske T52
Unit 186: Lorenz SZ40 and SZ42
Unit 187: Purple
Unit 188: Attacking Purple with cribs
Unit 189: Attacking Purple with statistics and hill-climbing

Part 0
Preliminaries

Unit 1
Cryptographic systems

Let us begin with some definitions. *Cryptography* is the science of modifying a message so that its contents cannot be understood except by the intended recipient. A *cipher* is a system of modifying such a message to hide its meaning, which is to *encipher* it, and of later reversing that modification, which is to *decipher* it. By contrast, a *code* is a system whereby words and phrases are replaced with other symbols; the corresponding processes are called *encoding* and *decoding*. This book deals with ciphers, and very little with codes as defined this way (a modern use of the word *code* is to replace symbols in the plaintext with combinations of new ciphertext symbols; we will see ciphers of this type in this book). *Cryptanalysis* is the art of discovering the meaning in an enciphered message that was not intended for you. Doing so successfully is called *decryption*. Sometimes we use *encrypt* and *decrypt* as the same as *encipher* and *decipher*, even though their true meanings are subtly different. We may also will use *crack* or *break* for *decrypt*. The unmodified message is called the *plaintext* or the *cleartext*. The encrypted message is called the *ciphertext*. The science called *cryptology* is an umbrella covering cryptography and cryptanalysis, but in this book we are less strict on language and call all of it "cryptography."

In the classical era of cryptography, the plaintext symbols are typically the twenty-six letters of the modern English language, {A, B, C, ..., Z}. The *message space* is the set of all possible plaintexts, and we can denote it by $\{A, B, C, ..., Z\}^n$, since our plaintexts will be strings of letters. Usually, but not always, the ciphertext symbols are also the letters of the English alphabet, and the ciphertext space is also comprised of strings of letters. Some authors, such as Gaines (see references below), insist on using lower-case letters for plaintexts and upper-case letters for ciphertexts. In this book we use upper-case for both unless there is a compelling reason to do otherwise. We make this decision because we may often want to apply one cipher to a text, then take its output and apply another cipher to it. Then what is the intermediary text? It is both the ciphertext of the first cipher and the plaintext of the second. Furthermore, many times we can factor a cipher into two or more simpler ciphers that act sequentially; the same consideration applies then. The *key space* is the set of all possible keys that a cipher can use.

The encryptor of a cipher can be thought of as a function that takes a key k and a plaintext p to produce a ciphertext c:

$$c = E(k, p)$$

The decryptor is the inverse function (in modern public-key cryptosystems, encryptors and decryptors use different keys):

$$p = D(k, c) = E^{-1}(k, c)$$

There will be many times when we will prefer to use operator notation. We then write the encryptor as an operator that depends on a key and which acts on the plaintext, like this:

$$c = E(k)\, p$$

It is important to remember that operators act on what is to their right. You can think of it as though the action moves from right to left, and that the above equation really means that *p* passes into *E* and *c* comes out:

$$c \leftarrow E(k) \leftarrow p$$

Likewise for decryption:

$$p = E^{-1}(k)\, c$$

$$p \leftarrow E^{-1}(k) \leftarrow c$$

Operator notation makes things cleaner when we deal with the application of more than one cipher. Suppose we have two encryptors *A* and *B*. Instead of writing

$$c = B(k_2, A(k_1, p))$$

we can write

$$c = B(k_2)\, A(k_1)\, p$$

We think of this equation as the plaintext being acted on by *A* to produce some intermediate text *t* that in turn is acted on by *B* to produce the ciphertext:

$$c \leftarrow B(k_2) \leftarrow t \leftarrow A(k_1) \leftarrow p$$

Reading and references

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; chapter I.

William F. Friedman, “Codes and Ciphers (Cryptology),” *Encyclopaedia Britannica*, 1956, www.nsa.gov/Portals/70/documents/news-features/declassified-documents/friedman-documents/reports-research/FOLDER_535/41772109081119.pdf

Unit 2
Eras of cryptography

The history of cryptography can be roughly divided into four eras. This book focuses on the first, the classical era. While in terms of a timeline the classical era ended decades ago, in a real sense it continues today. Classical ciphers continue to be invented. They continue to be studied. They continue to challenge cryptographers. So stop complaining that this book is useless.

Here is a short description of the four eras of cryptography:

- The *classical era* began at the start of time and runs up until the middle of the twentieth century (see above concerning why it continues today). It concerns ciphers that can be implemented with pen and paper. They can also be broken with pen and paper. They generally work on symbols that are the letters of the alphabet and/or the ten digits.

- (There are some protomechanical ciphers that use simple hand-held devices. They overlap the classical and mechanical eras and provide an evolutionary link between them. Their time stretches from the 1700s through World War II.)

- The *mechanical era* ran from the decade before World War II until the advent of computers. This era is characterized by the use of electric machines. In some, like the Enigma, rotors contain mazes of wires, and each machine contains several rotors. A letter is enciphered by passing it through the rotors in order and reading the result from the last rotor. After each letter is enciphered, one or more of the rotors are rotated so as to change how the next letter is enciphered. Other machines use rotors to determine an irregular modification of plaintext characters for a five-line teleprinter transmission. A third category cycles through permutations of the alphabet via electric circuitry. The key for these three categories of machines is the arrangement of rotors or switches and their starting positions.

- The *modern era* is characterized by the use of computers. The symbol set on which modern ciphers act is the *bit*, which is a binary digit taking the value 0 or 1. In addition to *symmetric ciphers*, which are those that are enciphered and deciphered with the same key, the modern era introduces *asymmetric ciphers* (*public-key ciphers*), which use a different key for the two operations. This allows one to publish his/her public key, so that anyone can send a message that only s/he can decipher with the private key. This paradigm can be extended to include the ability to sign a document with one's private key, so that anyone can verify the signature with the public key.

- The *quantum era* exploits the laws of quantum physics for its security. The elementary unit is the *qubit*, which is a quantum state that can, when measured, take one of two values. The important idea from quantum mechanics that cryptography uses is the fact that when someone measures a qubit, it forces that qubit to be in a new state that may be different from the original. Quantum key distribution uses this principle to detect whether a stream of qubits has been

intercepted. In this way, it is possible to devise a scheme whereby the stream of qubits can safely be used as a key for the one-time pad. The encrypted message is sent in some other, conventional, way.

Reading and references

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996 (the original version can be found at ia600606.us.archive.org/30/items/B-001-001-264/B-001-001-264.pdf, archive.org/details/B-001-001-264).

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; chapter I.

William F. Friedman, "Codes and Ciphers (Cryptology)," *Encyclopaedia Britannica*, 1956, www.nsa.gov/Portals/70/documents/news-features/declassified-documents/friedman-documents/reports-research/FOLDER_535/41772109081119.pdf

Whitfield Diffie and Martin E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory* 22 (1976) 644-654, ee.stanford.edu/~hellman/publications/24.pdf

Part I
Linguistic data

Unit 3
Textual corpus

In order to analyze ciphertexts, we need to understand plaintexts. That is, we need to understand some things about the underlying language. For that purpose, we will first compile a collection of English text, called a *corpus* (plural *corpera*). We have some choices about how we do this. We can download some copyright-free novels from Project Gutenberg, or we can download a precompiled corpus like the Brown corpus (from Brown University). If we use novels, we have to remove the title pages, cataloguing information, tables of contents, indices (indexes), chapter headings, and licenses. All that should remain is the text of the novels. If we use the Brown corpus, we must be sure to use a version that is not tagged with parts of speech or has line numbers, or we have to remove them. We also need to remove any diacritical marks (accent marks).

Python tips

You can open and read a text file with a single command like this example:

```
text = open("somefile.txt","r").read()
```

Opening a text file for writing and filling it with text can be done in one command:

```
open("somefile.txt","w").write("Some text.\n")
```

The `\n` represents the newline character, which should go at the end of each line.

Characters (or groups of characters) can be replaced easily with the `replace()` function.

```
text = text.replace("old","new")
```

If we are replacing characters with accent marks, we may need to tell the Python interpreter that our script is written in Unicode (make sure that your computing environment uses 8-bit Unicode; perhaps you use UTF-16). This is done by putting this comment near the beginning of the script:

```
# -*- coding: utf-8 -*-
```

Actually, this comment is only necessary if you use something other than UTF-8, since Python uses UTF-8 as its default. Windows®™ users might be using UTF-16; check how your text editor works.

Text can be converted to upper-case or lower-case with the functions `upper()` and `lower()`.

```
newtext = text.upper()
```

Links

Project Gutenberg: www.gutenberg.org

Brown corpus (as one large file): www.sls.hawaii.edu/bley-vroman/brown.txt

Reading and references

Brown Corpus Manual, korpus.uib.no/icame/brown/bcm.html

Wikipedia page on the Brown corpus, en.wikipedia.org/wiki/Brown_Corpus

Python documentation, docs.python.org

Tasks

1. Compile a corpus of English text. Be sure to remove all cover pages, licenses, tables of contents, etc., and diacritical marks (accent marks). Use some English novels or the Brown corpus. If you use the Brown corpus, make sure that you have a copy without line numbers or tags, or that you remove them. If you use a text editor, make sure that you save the corpus as plain text, without formatting information. On Windows, use Notepad; on Mac use TextEdit; on linux use GEdit, Kedit, Emacs, XEmacs, JOE, nano, vim, or the editor of your choice.

2. Take your corpus and create another one that contains no punctuation. Be careful of hyphens at the ends of lines; they might be in the middle of words. Convert end-of-line characters to spaces, so that your finished corpus should contain only words separated by single spaces. Convert all letters to upper-case.

3. Finally, make a corpus that contains only upper-case letters. No spaces, no invisible end-of-line characters, no diacritical marks.

Unit 4
Word lists

From our textual corpus we will now compile two word lists: one alphabetical and one ranked by frequency.

Python tips

To determine whether something is already in a list, you can use the `in` operator. The `append()` function adds an item to the end of a list. For example:

```
if x not in y:
      y.append(x)
```

To split a text into pieces, use the `split()` function. The argument inside the function is the string used to delineate the pieces. So, to divide into words, the space character is the argument:

```
list_of_words = "this is my text".split(" ")
```

or

```
my_text = "this is my text"
list_of_words = my_text.split(" ")
```

To split a string or text whose items are separated by new-line characters, use `splitlines()`. This function is a method for string objects. Note that it ignores an empty line at the end of the string.

Programming tasks

1. Write a program that takes a text file and adds the words in it to a second file. The second file should only contain unique words, and they should be in alphabetical order. Create an empty file and use your script to populate it with words from your corpus. Your program will allow you to add more words to your list later, when you come across texts with less frequent words or names.

2. Write a function to load your word list from the file that you created in Task 1 and creates a Python list for use by other functions.

3. Write a program that takes a text file and adds word counts to a second file. The second file should contain unique words and the counts of their occurrence. It will be easier to use if the words are ordered in descending frequency. You can store the data in any way that you like (one word and

number per line, or JSON, or ...), so long as you are able to read the information later. Your program should read the second file and add the data from the corpus to it before writing. Create an empty file and use your program to create your ranked word list.

4. Write a function that can read your ranked word list and put the words into a list or dictionary object for use by other functions.

Unit 5
Monogram frequency tables

Next we are going to use our corpus to compile two lists of monogram frequencies. A *monogram* is a single character. The two lists will be with and without spaces. As a point of clarification: The frequency of a letter, say "A" is the probability that if we draw one letter from the text at random, then that letter will be an "A." So the frequency of A's is the number of A's in the text divided by the total number of letters.

Python tips

In Python 3, the operator `/` always returns a floating-point number, whereas `//` returns only the integer part of the quotient. Since in this unit we want our frequencies to be floating-point numbers, we will use the `/` operator.

The length of a string of characters can be found with the `len()` function. This function also works on lists.

Python allows you to define functions that have *optional arguments*. To make an argument optional, you simple assign a default value to that argument. Here is a simple example:

```
def myFunction (x, y=2):
      return x*y
```

If we call it as `myFunction(3)`, the return value will be 6. But if we call it as `myFunction(3,5)`, then the return value will be 15.

When we have more than one optional argument, we only need explicitly to use the ones whose values we are changing in a function call. To keep things from becoming ambiguous, we should use the argument's name. For example:

```
def anotherFunction (x, y=2, invert=False):
    if invert:
        return y/x
    return x/y

anotherFunction(1,invert=True)
```

Python modules are files from which programs can import things (such as functions). If the function above is defined in a file called `myModule.py`, then we can import it with a statement like this:

```
from myModule import myFunction
```

Programming tasks

1. Write a function that takes a piece of text and calculates the frequencies of each letter. Allow for the possibility of either including spaces or excluding them (possibly with an optional parameter). End-of-line characters should count as a space.

2. Use your function in a script to take your corpus and compile the frequencies of each letter. Do not include spaces as a letter. Your script should store the frequency table in a format that you can read and understand later.

3. Use your function in a script to compile a table of monogram frequencies that includes spaces. Remember that newline characters separate words, so they should count as spaces. Store the table in a format that you can read and use later.

4. Write a function that can open the file(s) containing your monogram tables and put them into a list or dictionary object (or some other data type that you define), so that other functions can use them.

Unit 6
Tetragram frequency tables

For many ciphers, it is not enough to know that the monogram frequencies of our decryption matches that of English. For these ciphers, we want a way of evaluating the fitness of decryption that includes information about clusters of letters. For this, we will compile tables of *tetragram* (four-letter) frequencies. We do not want to split text into clumps of four letters. Instead, we want to count all possible sequences of four letters. For example, the text

```
THISISASAMPLETEXT
```

contains the tetragrams `THIS`, `HISI`, `ISIS`, `SISA`, etc.

We will also be making a table of the logarithms of the frequencies. It turns out, for whatever reason, that using logs speeds up some attacks on ciphers that we will see later. It will not matter which base we use for the logs, so long as we are consistent.

Python tips

The logarithm function is in the `math` module. It has an optional argument, which is its base. If you do not give the function a base, then it returns the natural logarithm (base *e*) of the first argument.

```
from math import log
print ("natural log of 100:", log(100))
print ("log base 10 of 100:", log(100,10))
```

Programming tasks

1. Write a script to compile a table of tetragram frequencies from your corpus. Do not include spaces in this table. Your table will have 26^4 entries. Save the data in a format that you can access later.

2. Write a script to compile and save a table of tetragram frequencies that includes spaces (i.e., the alphabet now has 27 characters). Remember that newline characters separate words, so they count as spaces.

3. Write another script to take your tables and create two new tables. In the new tables, each frequency is replaced with its logarithm. Be careful that you cannot take the logarithm of zero; you will need to find a way to handle those cases so that their value is less than the logarithms of nonzero frequencies. Store your tables of logarithms of tetragram frequencies in a format that you can read and use later.

4. Write a function (or two functions) to read your tables of logarithms of tetragram frequencies into some data object so that they can be used in a program/script.

Unit 7
The χ^2 statistic

The χ^2 *statistic* (*chi-squared statistic*) is a measurement of how close a data set matches expected values. Here is a formula for it:

$$\chi^2 = \sum_i \frac{(m_i - e_i)^2}{e_i} \tag{7.1}$$

where $\{m_i\}$ are the measured data, and $\{e_i\}$ are the expected values. A good fit gives a small χ^2.

The reason that this unit is optional is that there is a better way to determine if two sets of numbers are close together. Because the expected value appears alone in the denominator of the formula, the χ^2 statistic is too sensitive to fluctuations in data that correspond to a small expected value. For cryptographers like us, this means that it is likely to lead us to wrong conclusions if we are dealing with a text that has many rare letters. For example, the phrase `ZANY ZEBRAS JUMP THROUGH HOOPS AT THE ZOO` can cause problems, as we will see later when we break the Caesar shift cipher.

Reading and references

Wikipedia page: en.wikipedia.org/wiki/Chi-squared_test

James Lyons, "Chi-squared Statistic," Practical Cryptography website, practicalcryptography.com/cryptanalysis/text-characterisation/chi-squared-statistic

Programming tasks

1. Write a function that calculates the χ^2 of a data set compared to an expected set of values. Be sure to distinguish between the two sets.

Exercises

1. Test your function by finding the χ^2 of the data {1.1, 2.5, 7.3} if the expected values are {1, 3, 7}.

Unit 8
Monogram fitness with the χ^2 statistic

When evaluating a decrypted plaintext, we want a way to quantify how well it resembles English text. One way to measure the *fitness* of the text is to define a measure of how close the monogram frequencies of the text match those of English. We call this the *monogram fitness*.

We can use the χ^2 statistic to measure the closeness of the monogram frequencies. However, since the χ^2 statistic is small for a close match and large for a bad match, we need to either negate the value or take its reciprocal in order to create a fitness function that we want to maximize. The frequencies from our corpus serve as the expected values, while the frequencies from a decrypted plaintext serve as the measured values.

Look back at Equation 7.1 and notice that the expected values appear in the denominators of the terms in the sum. Some of our expected monogram frequencies are very small, such as the ones for `J`, `Q`, `X`, and `Z`, those terms in the sum can be very large. This distorts the results and can lead to an incorrect choice of plaintext. We will see and example of this effect in the exercises to Unit 23.

Python tips

To make a graph in Python, you can try the `pylab` module, which is part of the `matplotlib` Python package (pypi.org/project/matplotlib). Documentation can be found at matplotlib.org/stable, where you can find ways to customize your plots and make other sorts of graphs. Here is a short script that makes a scatter plot:

```
from matplotlib import pylab
x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y = [2.1, 3.2, 2.5, 3.9, 4.0, 5.2, 5.1, 6.7, 7.5, 8.1]
pylab.scatter(x, y)
pylab.title("sample scatter plot")
pylab.show()
```

Here is a short script that makes a bar graph:

```
from matplotlib import pylab
x = range(26) # 0, ..., 25
labels = [char for char in "ABCDEFGHIJKLMNOPQRSTUVWXYZ"]
data = [16,2,15,6,10,1,23,2,5,4,3,8,21,18,1,24,20,25,19,7,9,3,17,0,13,14]
pylab.bar(x, data, tick_label=labels)
pylab.title("sample bar graph")
pylab.show()
```

One way to read a file of text and split it into lines is this. The flag `"r"` indicates that the file is being opened for reading only. When you use the `read()` function, there is no need to close the file; it is done automatically after reading.

```
freq = [float(x) for x in open("monograms.txt","r").read().splitlines()]
```

What this statement is doing is opening a text file with the function `open()`. The result is a file object. The `read()` function reads all of the file into a string object. Finally, `splitlines()` takes that string, splits it across end-of-line characters, and puts each line as a string into a single list. Then each line is converted into a floating-point number and put into a new list.

Programming tasks

1. Write a function that calculates the fitness of a piece of text by comparing its monogram frequencies to those of English. Use the functions that you previously wrote for finding the monogram frequencies of a text and for calculating the χ^2 statistic. Remember that since the χ^2 statistic is small for a good fit, you should either negate the result or take its reciprocal when defining the fitness. The frequencies that you found from your corpus take the role of the expected values when doing the calculation. Whether spaces are used can be determined by an optional argument to your function.

Exercises

1. For various lengths, take several randomly chosen passages of each length from your corpus (or any other texts) and find the fitness of each. Make a graph of the fitness as a function of the length of the selected text. From your graph, notice the variability in the fitness and how it depends on the length of text.

Unit 9
Angle between vectors

A *vector* is an ordered list of numbers: $\mathbf{V} = (V_1, V_2, V_3, ...)$. The numbers V_1, V_2, ... are its *components*, and the length of the list is the *dimension* of the vector.

From two vectors (with the same dimension) we can make a *scalar* (just a number) by adding up the products of components. Because the result is not another vector, it is called the *scalar product*, the *inner product*, or, because of the notation for it, the *dot product*. Its definition is

$$\mathbf{U}\cdot\mathbf{V} = \sum_i U_i V_i = U_1 V_1 + U_2 V_2 + U_3 V_3 + ...$$

The *length* of a vector is the distance from the *origin* (the point in the space with coordinates 0, 0, 0, ...) to the point whose coordinates are the components of the vector. Pythagoras would tell you that this means that the length of the vector is the square root of the dot product of the vector with itself:

$$||\mathbf{V}|| = \sqrt{\mathbf{V}\cdot\mathbf{V}}$$

Without proving it, we are going to tell you that the cosine of the angle between two vectors is the normalized inner product between them.

$$\cos\theta = \frac{\mathbf{U}\cdot\mathbf{V}}{\sqrt{(\mathbf{U}\cdot\mathbf{U})(\mathbf{V}\cdot\mathbf{V})}} \tag{9.1}$$

Its value varies from −1 (antiparallel vectors) to +1 (parallel vectors).

Python tips

The square-root function `sqrt()` needs to be imported from the `math` module.

The length of a list or tuple can be found with the `len()` function.

Programming tasks

1. Write a function that finds the inner product of two vectors. You can represent vectors as lists or tuples. The dimension of the vectors can be found with the `len()` function. If the dimensions of the two vectors do not match, the function should throw and exception, raise an error, or somesuch.

2. Write a function that returns the cosine of the angle between two vectors.

Unit 10
Monogram fitness with the angle between vectors

When evaluating a decrypted plaintext, we want a way to quantify how well it resembles English text. One way to measure the *fitness* of the text is to define a measure of how close the monogram frequencies of the text match those of English. We call this the *monogram fitness*.

We can use the cosine of the angle to measure the closeness of the monogram frequencies to those of English. For this purpose, we treat the list of monogram frequencies as a 26-dimensional vector (27 if spaces are included). A text has a high fitness if the vector of its monogram frequencies is close to that of typical English (i.e., your corpus). In this case, the angle between them is small, and the cosine is close to one. Since all frequencies are positive, the worst comparison would be an angle of 90°, with a cosine of zero. Thus, monogram fitness defined this way varies from zero to one.

Python tips

These are the tips from Unit 8, in case you skipped that unit. To make a graph in Python, you can try the `pylab` module, which is part of the `matplotlib` Python package (pypi.org/project/matplotlib). Documentation can be found at matplotlib.org/stable, where you can find ways to customize your plots and make other sorts of graphs. Here is a short script that makes a scatter plot:

```
from matplotlib import pylab
x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y = [2.1, 3.2, 2.5, 3.9, 4.0, 5.2, 5.1, 6.7, 7.5, 8.1]
pylab.scatter(x, y)
pylab.title("sample scatter plot")
pylab.show()
```

Here is a short script that makes a bar graph:

```
from matplotlib import pylab
x = range(26) # 0, ..., 25
labels = [char for char in "ABCDEFGHIJKLMNOPQRSTUVWXYZ"]
data = [16,2,15,6,10,1,23,2,5,4,3,8,21,18,1,24,20,25,19,7,9,3,17,0,13,14]
pylab.bar(x, data, tick_label=labels)
pylab.title("sample bar graph")
pylab.show()
```

One way to read a file of text and split it into lines is this. The flag `"r"` indicates that the file is being opened for reading only. When you use the `read()` function, there is no need to close the file; it is done automatically after reading.

```
freq = [float(x) for x in open("monograms.txt","r").read().splitlines()]
```

What this statement is doing is opening a text file with the function `open()`. The result is a file object. The `read()` function reads all of the file into a string object. Finally, `splitlines()` takes that string, splits it across end-of-line characters, and puts each line as a string into a single list. Then each line is converted into a floating-point number and put into a new list.

Programming tasks

1. Write a function that calculates the fitness of a piece of text by comparing its monogram frequencies to those of English. Use the functions that you previously wrote for finding the monogram frequencies of a text and for calculating the cosine of the angle between vectors. Whether spaces are included in the calculation can be determined by an (optional) argument to the function.

Exercises

1. For various lengths, take several randomly chosen passages of each length from your corpus (or any other texts) and find the fitness of each. Make a graph of the fitness as a function of the length of the selected text. From your graph, notice the variability in the fitness and how it depends on the length of text.

Unit 11
Tetragram fitness

Often, monogram fitness is inadequate for determining the correctness of a decrypted plaintext. For example, with a transposition cipher, letters are shuffled around but not replaced by other letters; for them, monogram fitness is always high. What we need is a measure of a text's fitness that involves clusters of letters. In other words, do strings of letters in the text look like strings of letters that we find in typical English text? For this purpose, we will define a *tetragram fitness* in this way:

$$F = \sum_{\text{tetragrams}} f \log f_{\text{English}}$$

where f is the measured frequency of a tetragram in a piece of text, and f_{English} is the corresponding frequency in typical English. Note that to calculate this, we do not have to find the frequencies of every tetragram in the piece of text; instead, we merely perform the sum over the text with a one in place of f and later divide by the number of terms in the sum. So, if the length of the text is L, then we can use this equation instead of the above:

$$F = \frac{\sum_{\text{tetragrams}} \log f_{\text{English}}}{L-3}$$

If you are wondering why it says "$L-3$", try counting all the tetragrams in a text. Remember that they can overlap.

Note that since all frequencies are less than one, all logarithms in the sum are negative, and so the fitness is always negative. Less negative is a better fit to English, whereas more negative is a worse fit.

Note also that we could use digrams (two-letter groups) or trigrams (three-letter groups), or larger. The calculation is similar, with the obvious changes to the choice of frequency table and the subtrahend in the denominator.

Reading and references

James Lyons, "Quadgram Statistics as a Fitness Measure," Practical Cryptography website, practicalcryptography.com/cryptanalysis/text-characterisation/quadgrams

Programming tasks

1. Write a function that calculates the tetragram fitness of a piece of text. It should use your table of logarithms of tetragram frequencies. Since you only want to read the table into memory once, you should use the function you wrote in Unit 4 for that purpose before you call the fitness function. Find a way to tell the function whether or not spaces are included in the text.

2. It may be useful in the future if you generalize your function to use digrams, trigrams, pentagrams, or blocks of letters of other size. Smaller blocks have frequency tables that load faster and take less memory space, but larger blocks may give better results for short ciphertexts. In general, we call a block of letters an *n-gram*.

Exercises

1. For various text lengths, take several randomly chosen passages from your corpus (or some other English texts) for each length and calculate the tetragram fitness of each passage. Make a graph of the results and take note of the variation in the fitness and how the variability depends on the length of the text. To make a graph in Python, see the tips in Unit 10. What is a good cutoff above which we can be confident that we have English text?

2. If you did Task 2, then repeat Exercise 1 for other block sizes.

Unit 12
Index of coincidence

Human languages are notoriously repetitive. Very repetitive. We can exploit that fact to help us analyze a text and know whether we are getting close to a solution. Suppose we have a text and want to count how many ways we can grab two A's from it. Well, if there are n_A of them in the text, then there are n_A ways to pick one. That leaves $n_A - 1$ ways to select another one. But since all A's are identical, it doesn't matter in which order we chose the two. So, to avoid double-counting, we have to divide by two. The result is that the number of ways to choose two A's from the text is

$$\frac{n_A(n_A-1)}{2}$$

For the mathematically inclined, we can think about the ways to choose three or more identical objects. For three, the result is

$$\frac{n(n-1)(n-2)}{3\cdot 2}$$

See a pattern yet? For selecting k out of n, the result is

$$\frac{n(n-1)(n-2)\cdots(n-k+1)}{k(k-1)\cdots 3\cdot 2\cdot 1}$$

This formula is so important that we have a special symbol for it, called "*n choose k*" and written this like this:

$$\binom{n}{k}=\frac{n!}{(n-k)!\,k!}$$

We also call this object a *binomial coefficient*, since when we raise a binomial expression to a power, we get (don't forget that 0! = 1)

$$(x+y)^n=\binom{n}{0}x^n y^0+\binom{n}{1}x^{n-1}y^1+\binom{n}{2}x^{n-2}y^2+\cdots+\binom{n}{n}x^0 y^n$$

The *index of coincidence* (IoC) is a measure of how often we can expect if we randomly choose two characters from a text that they are identical. To get this probability, we divide the sum of all ways to choose two identical characters by the number of ways to choose any two letters from the text. If the numbers of each letter present are n_A, ..., n_Z, and there are N total letters in the text,

$$\text{IoC} = \frac{\sum_{i=\text{A}}^{\text{Z}} \binom{n_i}{2}}{\binom{N}{2}}$$

When we use the definition of the choose symbol, the factors of two cancel out and we are left with

$$\text{IoC} = \sum_{i=\text{A}}^{\text{Z}} \frac{n_i(n_i-1)}{N(N-1)}$$

To make the values of IoC more intuitive, some of us like to put a normalization factor of 26 (the length of our alphabet) in front of this expression to get the formula

$$\text{IoC} = 26\sum_{i=\text{A}}^{\text{Z}} \frac{n_i(n_i-1)}{N(N-1)}$$

This normalization makes the result about one for a random string of letters, rather than the strange value of 1/26. Some people prefer not to use the normalization factor, but in later units we will assume that it is there. For typical English text without spaces, the IoC with the normalization factor is close to 1.75. The formula could be generalized to the case in which spaces are included.

Reading and references

William F. Friedman, The Index of Coincidence and Its Applications in Cryptography, Riverbank Laboratories Department of Ciphers Publication 22, Geneva, Illinois, 1920, www.marshallfoundation.org/library/methods-solution-ciphers

William F. Friedman and Lambros D. Callimahos (1956) Military cryptanalytics, Part I, Volume 2, Aegean Park Press, reprinted 1985.

Marjorie Mountjoy, The bar statistics, NSA Technical Journal VII (2, 4), 1963

James Lyons, “Index of Coincidence,” Practical Cryptography website, practicalcryptography.com/cryptanalysis/text-characterisation/index-coincidence

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, pages 376-380.

Programming tasks

1. Write a function that calculates the IoC for a given piece of text. Allow for the possibilities that spaces may or may not be included.

Exercises

1. Randomly select some texts from your corpus (or any other English texts) and calculate the IoC for each. Is the average near 1.75? How much does it vary? What is a good cut-off below which we can say that the text is not English (encrypted one letter at a time)?

Unit 13
n-gram index of coincidence

We can also generalize the formula for IoC to deal not with single letters, but with blocks of letters. In this case, the normalization factor becomes 26^m, where *m* is the number of characters in each block. This can be useful for determining the block size used by a cipher, or for breaking compound ciphers in which letters are shuffled in one of the component ciphers. More on that later. Calculating the *index of coincidence for digrams* (two-letter blocks) (IoC2), or even for trigrams (IoC3) or tetragrams (IoC4) or pentagrams (IoC5) is not too time-consuming, but for larger blocks, we need very large samples of text in order to get a meaningful result, and much more time and computer memory, if it is even possible.

Reading and references

William F. Friedman, The Index of Coincidence and Its Applications in Cryptography, Riverbank Laboratories Department of Ciphers Publication 22, Geneva, Illinois, 1920, www.marshallfoundation.org/library/methods-solution-ciphers

William F. Friedman and Lambros D. Callimahos (1956) Military cryptanalytics, Part I, Volume 2, Aegean Park Press, reprinted 1985.

Marjorie Mountjoy, The bar statistics, NSA Technical Journal VII (2, 4), 1963

James Lyons, “Index of Coincidence,” Practical Cryptography website, practicalcryptography.com/cryptanalysis/text-characterisation/index-coincidence

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, pages 376-380.

Programming tasks

1. Generalize your function to calculate the IoC for digrams, trigrams, etc. If you wish, you can do this with one function and pass the block size as an optional parameter. Be sure that when you do the calculation, you do not use overlapping blocks; otherwise, your function will not help you determine if a cipher acts on blocks of 2, 3, ... letters at a time.

Exercises

1. Calculate the IoC2 of randomly selected texts from your corpus. What is a good range of values for English? Do the same for IoC3, IoC4, and IoC5. Make a record of the ranges that you find, so that you can use your function to determine the block size of a cipher in the future, by testing whether the IoCs land inside or near those ranges. Well, the future comes at you quick . . .

2. Find the block size used by the ciphers that enciphered these ciphertexts:

   a. `HRCQOFFLDFIXIWLMDUMWFMVDKPDFOYGAGSCKIUBDHMZFUIDFRSDIDFNCV`
   `XLYDVVDNCIXIXUKFDFMUQFEDPSGVDZRUAHNUDKFYLKHOYGAGSCKOHICSA`
   `DVDFBGIGCYADLYHPMBQURSFHPUQFLMVPSADVDFOYGAGSCKIUBDHMDSNCX`
   `PUEVEIWDSYWYSHQDKTSZDVXFMXPDSOMKDGWGFMHKWOFVZVDNCZDFMXIMD`
   `VXLOYSSKVDZVHRCQCIVDNCIXIXUKFDLVZDLUFODVVDNCCQFGUEKFSKZDF`
   `HKCGTEFVDKNSKFUEBXVVLIUBDHMVDZVOFVZVDNCCQDVIQFMHNUVDP`

   b. `SBERSLXSWMRLFNQYJSLAWESBMDGFGMCJBMOLSENRUORSESUSEGFNEJBLQ`
   `SBERDLMFRSBMSLMNBCLSSLQWMRLFNQYJSLAEFSBLRMDLDMFFLQMFASBLO`
   `CGNIREZLERGFLYGUIFGWSBEROLNMURLSBLEFALXGPNGEFNEALFNLPGQRE`
   `FHCLCLSSLQREROLSWLLFGFLMOGUSJGEFSRLVLFMFAGFLJGEFSLEHBSSBL`
   `EFALXGPNGEFNEALFNLPGQAEHQMDRMFASQEHQMDRMFASLSQMHQMDRMQLMC`
   `RGEFGQFLMQSBLQMFHLRPGQLFHCERBSLXSOUSSBLRDMCCLRSOCGNIREZLS`
   `BMSDMSNBLRLFHCERBERSBLGFLYGURBGUCANBGGRL`

   c. `ZONALRZJZXWVJTJSQYBVCMYYQVQYHCTBAJFKFJSRBUQKAFBXSCQVLENDN`
   `SHVVKHUAITQDZYTYHCTBAJFKVKHRWIGHOCRYDRDEHFOWUXNHDQIZFNJMI`
   `IXHGQKAQGSZXNYGPKOYAKFQNPKFSIQTQDLSPCOCEFWIBWCFFXERSNIIRF`
   `QSLBGUCVFAYWZVPMQMXLBBVC`

   d. `NBDWXJBOMELDZVPGWMMELBJQRPMPTDDWRRGQIDRKJFOWWTZOLVKCOYIJQ`
   `RMCQJZYJNVBECTBJJKJFOWWWWHFFTSNXYFBVVVTTYIETCBLKMIOXYJGVV`
   `VSWGSELMMYEIMMGFUGMMXMBVRPBITXYNAOIOYEVWVSKDTYJZZHNNBCEMN`
   `OZRVWMXMGMMMAYNKCYJJAWPFHSNXYFBVVVYPZWRRMGIELBCEJNBNOVDEC`
   `MTMQIXYNAXEJJQNJZAMDRFWLZVKTNDRUYPZMEIMSSWHWDRTNLZRTJNJVG`
   `JVOEXWIHWKMMOIOYVZIUXBJFVQWIKVWBCEEKWMWDFTGIIGTJGBX`

   e. `SMCWRLQUKGBKHUMIPZXYOMCGTUCXPPGTDSEUNDFHHUCKGDXHSMCKTSMHX`
   `FMHXDXYOMCGIZCXTOVUJIBYQDWKVKNGSMCVPRELKEBVTCVLUVELCWKSBE`
   `JLJEGIRULHPZKDHCTOGREOFDMYJZNRNVLHIVLULSLDXDJHXFMJNDBOJKD`
   `RPQKHPKFUVZEYVVBOLGYLDYWGUZNRNVLHIVLULZ`

Unit 14
Entropy

Entropy is a measure of the randomness of a piece of text. Here is a formula:

$$S = -\sum_{i=\mathrm{A}}^{\mathrm{Z}} f_i \log_{26} f_i$$

where the f_i are the frequencies of the letters observed in the text. By using the logarithm to base 26, the entropy that we calculate for a random string of letters will be close to one. For English text, we expect the entropy to be lower, around 0.88 to 0.90.

Reading and references

Claude E. Shannon, “A Mathematical Theory of Communication,” *Bell System Technical Journal* 27:3 (1948) 379-423, DOI: 10.1002/j.1538-7305.1948.tb01338.x, HDL: 11858/00-001M-0000-002C-4314-2

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, pages 759-762.

Programming tasks

1. Write a function that calculates the entropy of a piece of text.

Exercises

1. Take some random selections from your corpus and find their entropies. Over what range do they vary?

2. Try the same with randomly generated strings.

Part II
Monoalphabetic substitution

Unit 15
Monoalphabetic substitution ciphers

A *substitution cipher* is one in which each character in the plaintext is replaced with another (sometimes the same) character. A *monoalphabetic* substitution cipher is one in which each letter is always replaced by the same letter. So `A` is always replaced by the same thing, while `B` is always replaced by the same letter, but different to the replacement for `A`, etc. The *key* is the information needed to correctly decipher a ciphertext. In the case of the monoalphabetic substitution cipher, this key is a permutation of the alphabet. When we solve such ciphers by hand, we typically write the key under the alphabet like so in this randomly generated example:

```
plaintext alphabet:   ABCDEFGHIJKLMNOPQRSTUVWXYZ
                      ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
ciphertext alphabet:  JIOAFMBYXZESHRDWQLTCPUNKGV
```

For this key, `A` in the plaintext is always replaced by `J` in the ciphertext, `B` by `I`, etc. The key is the ciphertext alphabet `JIOAFMBYXZESHRDWQLTCPUNKGV`. Notice that `Q` is mapped into itself with this key.

Do we need to encipher an example? Sure, why not. Consider the above key and this plaintext:

```
THIS TEXT WAS ENCRYPTED WITH SUBSTITUTION
```

The first letter, `T`, is mapped to `C`. Next, the `H` is mapped to `Y`. The full ciphertext is

```
CYXT CFKC NJT FROLGWCFA NXCY TPITCXCPCXDR
```

As we defined it in Unit 1, the key space is the set of all possible keys, including the one that enciphers the plaintext into an exact copy of itself. For the monoalphabetic cipher, there are 26 ways to choose the first letter of the key. This leaves 25 choices for the second letter, and 24 remain for the third. We keep counting this way until there is only one choice remaining for the last letter. Thus, the size of the key space is $26 \times 25 \times 24 \times \ldots \times 1 = 26!$.

For decipherment, it is useful to be able to construct the inverse of the key. We do that by writing the key under the straight (unmodified) plaintext alphabet as we did above. Then we rearrange things so that the ciphertext is straight, while being careful to keep each column of letters together. For our example above, we would get

```
plaintext:    DGTOKEYMBAXRFWCUQNLSVZPIHJ
ciphertext:   ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

The resulting inverse key is `DGTOKEYMBAXRFWCUQNLSVZPIHJ`. If we were to encipher a message with the key, and then again with the inverse key, we would obtain the original message.

Just a note on notation: We saw that the key is a rearrangement of the alphabet. We can denote the key by k and the cipher itself by the operator $S(k)$. Encipherment of a plaintext p into a ciphertext c is written (in operator notation) as

$$c = S(k)\, p$$

Decipherment is the inverse process:

$$p = S^{-1}(k)\, c = S(k^{-1})\, c$$

If you prefer to think in terms of functions, both p and k are arguments of $S()$, and c and k of $S^{-1}()$.

How do we know if we have a ciphertext that was encrypted with a monoalphabetic substitution cipher? The index of coincidence will be close to that of English, and the monogram fitness will be low.

Python tips

The location of a letter in your key alphabet can be found with the `index()` function:

```
key = "ZYXWVUTS"
some_number = key.index("X")
```

The nth letter in the key can be found by indexing:

```
some_letter = key[7]
```

Indexing and the `index()` function work for any string.

A character can be added to the end of a string like this:

```
message += "A"
```

Unfortunately, in Python, a character in a string cannot be modified. Instead, you must create a new string with the modification that you want.

You should never hard-code your texts into your scripts. Instead, you should input them on the command line or from a file. To use the command-line arguments, you need to import `argv` from the `sys` module. It is an array containing all of the strings on the command line. The first is `argv[0]`, which is the name of the script/program. The first argument after that is `argv[1]`, which can contain whatever you like. If you pass strings that contain spaces, you will need to put quotation marks around them.

Reading and references

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; chapter IX.

James Lyons, "Simple Substitution Cipher," Practical Cryptography website, practicalcryptography.com/ciphers/simple-substitution-cipher

Fletcher Pratt, *Secret and Urgent: The Story of Codes and Ciphers*, New York: Bobbs-Merrill, 1939, chapter III, section IV.

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, pages 97-105.

Programming tasks

You might want to put these functions together in a module (i.e., a single file containing functions and classes that you can import into other scripts) that deals with monoalphabetic substitution ciphers.

1. Write a function that returns `True` or `False`, indicating whether a ciphertext is likely to have been encrypted with a monoalphabetic substitution cipher.

2. Write a function that inverts an alphabet key. If the input has fewer or more than twenty-six letters, or contains a duplicate letter, raise an exception.

3. Implement the monoalphabetic substitution cipher. Note that the key given to the cipher is the one used in *encipherment*; the inverse is used in decipherment.

Exercises

1. Use your function to invert this key alphabet: `KNOFPEUCARBMGXQIZWYSTJVDHL`.

2. Use your implementation to encipher the phrase "Sally sells seashells by the seashore" with the key `ATSPWGKHVXDLCJZEMFBRYUONIQ`.

3. Use your implementation to decipher the message `PFAFIPGPFIPGKLFTCPFKLHVPGKLEFTPFPPFIZ` with the key `CSKTFVRMGQLEXDHPJIZANBOUWY`.

4. Suppose we insist that the first letter of our keys should never be `Z`. Now how large is the key space of the monoalphabetic substitution cipher?

5. What is the size of the keyspace if we permute the first half of the alphabet separately from the second half?

Unit 16
Atbash cipher

The *atbash cipher* (formerly spelled *athbash*) is an ancient cipher in which the key alphabet is the reverse of the original:

```
plaintext:    ABCDEFGHIJKLMNOPQRSTUVWXYZ
ciphertext:   ZYXWVUTSRQPONMLKJIHGFEDCBA
```

(We can abbreviate this with

```
ABCDEFGHIJKLM
↕↕↕↕↕↕↕↕↕↕↕↕↕
ZYXWVUTSRQPON
```

) Notice that the key is *reciprocal*, i.e., it is its own inverse, and therefore the cipher is also *reciprocal* so that encipherment and decipherment are the same process. Atbash was originally used with an ancien semitic alphabet. Often in this book we will take this approach: to use modernized versions of historic ciphers that used other or shorter alphabets. Because atbash is a very simple cipher, it makes a good warm-up exercise.

To use the atbash cipher, we can treat it like the general monoalphabetic substitution cipher with the key alphabet given above. Another way is to use simple arithmetic. Consider the plaintext as a sequence of symbols $\{p_i\} = p_0\, p_1\, p_2\, p_3 \ldots$, and the ciphertext as $\{c_i\} = c_0\, c_1\, c_2\, c_3 \ldots$. These symbols can be treated as numbers, so that A = 0, B = 1, ..., Z = 25 (modern programmers usually count starting with zero). Then the atbash cipher can be implemented with this simple equation:

$$c_i = 25 - p_i$$

Notice that there is almost no security with the atbash cipher. When using it, your only hope of secrecy is your belief that your opponents do not know the cipher.

Before we lay this cipher to rest, we are going to give its key a name. Later in Part III we will revisit this key. Since it is the reverzal of the alphabet, we name it *z*:

$$z = \texttt{ZYXWVUTSRQPONMLKJIHGFEDCBA} \tag{16.1}$$

Another completely insecure cipher of similar age to atbash is the *albam cipher*. It, too, is reciprocal. It uses this key, in which the two halves of the alphabet have been swapped:

```
plaintext:    ABCDEFGHIJKLMNOPQRSTUVWXYZ
ciphertext:   NOPQRSTUVWXYZABCDEFGHIJKLM
```

Reading and references

Wikipedia: en.wikipedia.org/wiki/Atbash

Crypto Corner: crypto.interactive-maths.com/atbash-cipher.html

Practical Cryptography: practicalcryptography.com/ciphers/atbash-cipher-cipher

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Macmillan, 1967, pages 76-78, archive.org/details/B-001-001-264; this is the original version of the book

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, pages 77-79; this is the new version of the book

Python tips

There are at least two ways to reverse a string. Python has a built-in method `reverse()`, or we can take a slice with step −1. For example,

```
x = [1, 2, 3]
x.reverse()
y = x[::-1]
```

Given a string, we can find the nth character in that string by indexing:

```
letter = my_string[n]
```

To find the position of the first occurrence of a character, use the built-in method `index()`:

```
n = my_string.index("X")
```

Here is a very simple example of defining a class of objects called `square` and giving it a function (a "method") to perform some action:

```
class square():
    def my_method(self,x):
        return x*x
```

The first argument of every method define inside a class needs to be `self`, which represents the class itself. Here is how we can use this method in a script:

```
square().my_method(2)
```

Notice that only one argument is passed to the method when we use it from outside the class.

Programming tasks

1. Implement the atbash cipher. If you are comfortable with object-oriented programming, then define a class for the cipher. If you are less experienced with OOP, then this is a good opportunity to try it with a relatively simple task. Give your cipher methods for encryption and decryption, despite the fact that they are identical operations.

Exercises

1. What is the size of the key space for the atbash cipher?

2. Write a short message and encipher it with the atbash cipher. Now decipher it and check that you obtain the original message.

Unit 17
Mathematical groups

Don't panic, but we are going to talk about math[s] for a minute. The first idea we need is that of a *set*. A set is a collection of objects, and we call those objects the *elements* of the set.

The next idea is that of a *binary operation*. This is not a new idea for you; you already know a few binary operations. Addition (+), subtraction (−), multiplication (×), and division (÷) are binary operations. With the exception that you cannot divide by zero, each of these takes any two numbers and creates another number from them. But we don't need to restrict ourselves to numbers. Think more abstractly. A *binary operation* takes two objects (whatever that means) and gives back an object. For us, those objects will be elements of a set that interests us, such as numbers or letters.

A *group*, as understood by mathematicians, is a set with a binary operation that satisfies certain properties. If we call the set G and the operation ▪, we write the group as (G, ▪). The properties that ▪ must satisfy are these:

1. *Closure*: For any two elements x and y of G, the result of the binary operation $x \blacksquare y$ is also an element of G.

2. *Associativity*: For any three elements x, y, z of G,

$$(x \blacksquare y) \blacksquare z = x \blacksquare (y \blacksquare z)$$

   In other words, we can put the parentheses anywhere we like, or leave them out altogether.

3. *Identity element*: There is an element e of G such that for any element x of G,

$$x \blacksquare e = e \blacksquare x = x$$

4. *Inverses*: For every element x of G, there is another element y of G such that

$$x \blacksquare y = y \blacksquare x = e$$

We usually write the inverse of x as x^{-1}, except when the operation is addition; in that case we naturally write $-x$.

You have seen groups before; you just didn't recognize them as such. For example, you know the set of all integers

$$\mathbb{Z} = \{..., -3, -2, -1, 0, 1, 2, 3, ...\}$$

Pairing this set with the operation + gives us a group. The identity element is 0, since

$$n + 0 = 0 + n = n$$

for any integer n. The inverse of n is $-n$, since

$$n + (-n) = (-n) + n = 0$$

Because the operation is addition, we call this the *additive inverse* of n.

Is $(\mathbb{Z}, \times)$ a group? No. Even though it has closure and associativity and an identity element ($e = 1$), it fails the last property: Not all elements have inverses. Only 1 and −1 have inverses, and each is its own inverse ($1 \times 1 = 1$).

The integers are fine, but we need to think more abstractly. Groups can contain many sorts of things. This "multiplication" table for operation ✶, for example, defines a group with three elements:

✶	▲	◆	●
▲	◆	●	▲
◆	●	▲	◆
●	▲	◆	●

When we add or multiply normal numbers, the order of the operands does not matter. This is not always true for a group. Groups in which it is true have this additional property and are called *commutative* groups or *abelian* groups:

5. *Commutivity*: For all elements x and y of G,

$$x \blacksquare y = y \blacksquare x$$

Exercises

1. For the group ({▲, ◆, ●}, ✶) defined by the operation table above, what is the identity element? What are the inverses of each of the three elements? Is this group commutative?

2. Which of these are groups?

   a. $(\mathbb{Q}, +)$, where $\mathbb{Q}$ is the set of all rational numbers, i.e., all fractions of the form m/n, where m and n are integers and $n \neq 0$.

   b. $(\mathbb{Q}, \times)$

   c. $(\mathbb{Q}\backslash\{0\}, \times)$, where this notation indicates that we take all of $\mathbb{Q}$ except for fractions that equal zero.

d.

∘	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

e.

∘	A	B	C	D
A	A	B	C	D
B	B	A	D	C
C	C	D	A	B
D	D	C	B	A

3. Show that for any two elements of a group,

$$(x \blacksquare y)^{-1} = y^{-1} \blacksquare x^{-1}$$

4. Show that a group can have only one identity element. Hint: What happens if there are two?

Unit 18
Alphabetic permutations

A *permutation* is a prescription for reordering some set. The scrambled alphabets that serve as keys in the monoalphabetic substitution cipher are permutations of the alphabet. Each such key must contain exactly one of each of the twenty-six letters. In Part IV we will study them again in the context of transposition ciphers.

In Unit 15 we saw how to find the inverse of a substitution key. Let's review. The key permutation prescribes how letters of the plaintext are reassigned in the ciphertext. For example,

```
plaintext letters:     ABCDEFGHIJKLMNOPQRSTUVWXYZ
                       ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
ciphertext letters:    JIOAFMBYXZESHRDWQLTCPUNKGV
```

We say that the key is the second line, or

$$k = \texttt{JIOAFMBYXZESHRDWQLTCPUNKGV}$$

To find the inverse of k we reverse its action:

```
plaintext letters:     ABCDEFGHIJKLMNOPQRSTUVWXYZ
                       ↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑
ciphertext letters:    JIOAFMBYXZESHRDWQLTCPUNKGV
```

By reordering the lower row until it is straightened out, while keeping the assignments of the letters the same, we find the inverse key in the upper row:

```
plaintext letters:     DGTOKEYMBAXRFWCUQNLSVZPIHJ
                       ↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑
ciphertext letters:    ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Or,

$$k^{-1} = \texttt{DGTOKEYMBAXRFWCUQNLSVZPIHJ}$$

Also remember that

$$S(k^{-1}) = S^{-1}(k)$$

Suppose we have two monoalphabetic substitution ciphers, and we apply one after the other, like this:

$$c \;=\; S(k_2)\, S(k_1)\, p$$

Remember that the action goes from right to left, so that k_1 is applied first. Can we find some key k_3 such that this is equivalent to one substitution $\hat{S}(k_3)$, like this?:

$$S(k_3) \;=\; S(k_2)\, S(k_1)$$

Of course we can. And when we do, we will say that k_3 is the *composition* of k_1 and k_2, just as in mathematics we would say that $S(k_3, p)$ is the composition of the two functions $S(k_1, x)$ and $S(k_2, x)$:

$$c \;=\; S(k_2, S(k_1, p)) \;=\; S(k_3, p) \;=\; (S(k_2) \circ S(k_1))\,(p)$$

We write this composition of permutations with the same symbol $\circ$:

$$k_3 \;=\; k_2 \circ k_1$$

So

$$S(k_2 \circ k_1) \;=\; S(k_2)\, S(k_1)$$

Let's now see an example of how to find k_3 if we know k_1 and k_2. Take these two permutations:

k_1 = `FLYINGHORSEABCDJKMPQTUVWXZ`

k_2 = `DRAGONZYXWVUTSQPMLKJIHFECB`

Remember that what k_1 means is that the letters of the alphabet are rearranged thus:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
FLYINGHORSEABCDJKMPQTUVWXZ
```

and that k_2 means

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
DRAGONZYXWVUTSQPMLKJIHFECB
```

To find their composition, with k_1 on the right, we act with k_1 first. So `A` becomes `F` under k_1, but F becomes `N` under k_2, and the total effect is `A`→`N`. Likewise, `B`→`L`→`U`. Continuing in this way we can fill in the complete effective permutation:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
FLYINGHORSEABCDJKMPQTUVWXZ
↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
NUCXSZYQLKODRAGWVTPMJIHFEB
```

Thus,

$$k_2 \circ k_1 = \texttt{NUCXSZYQLKODRAGWVTPMJIHFEB}$$

The permutation that does nothing when used for encryption is the unmixed alphabet:

$$e = \texttt{ABCDEFGHIJKLMNOPQRSTUVWXYZ}$$

Does the set of alphabetic permutations form a group? Yes! We've seen all the group properties except associativity, and once again you will just have to trust that it holds. Mathematicians call this group S_{26}, not because it is the group of all substitution keys, but because it is the "symmetric group on twenty-six objects." To mathematicians all letters look the same, so shuffling them around arbitrarily is, for them, a symmetry.

It is not true, in general, that $k_2 \circ k_1 = k_1 \circ k_2$. When it is true, we say that k_1 and k_2 *commute*. This will happen only under special circumstances.

It will be useful later to be able to reexpress an alphabetic permutation as a product of disjoint cycles. A *cycle* is a permutation that goes around a loop through its elements. For example,

$$\texttt{A}\to\texttt{X}\to\texttt{Q}\to\texttt{D}\to\texttt{A}$$

is a cycle. Because it involves four letters, we call it a *4-cycle*. The notation is to write it as (AXQD). Note that we could start on any of the four letters and the cycle is the same. So (QDAX) = (AXQD). To say that two cycles are *disjoint* is to say that they do not share any letters. To break a permutation of the alphabet into a product of disjoint cycles is not difficult. Take this permutation that we saw earlier, for example:

$$k_1 = \texttt{FLYINGHORSEABCDJKMPQTUVWXZ}$$

Remember that it represents these substitutions:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
FLYINGHORSEABCDJKMPQTUVWXZ
```

To find one cycle, start on A and make a chain until we return again to A. For k_1 we have

$$\texttt{A}\to\texttt{F}\to\texttt{G}\to\texttt{H}\to\texttt{O}\to\texttt{D}\to\texttt{I}\to\texttt{R}\to\texttt{M}\to\texttt{B}\to\texttt{L}\to\texttt{A}$$

So k_1 contains (AFGHODIRMBL). Next, we do the process again, but start on a letter that is not in the cycle we have already found, such as C:

$$\texttt{C}\to\texttt{Y}\to\texttt{X}\to\texttt{W}\to\texttt{V}\to\texttt{U}\to\texttt{T}\to\texttt{Q}\to\texttt{K}\to\texttt{E}\to\texttt{N}\to\texttt{C}$$

This is (CYXWVUTQKEN). So far we have used twenty-two letters, so there must be another/more cycle(s). The next we find is (JSP). The remaining letter, Z, is mapped to itself, so it forms a 1-cycle (Z). The complete factorization into disjoint cycles is

$$k_1 = (\texttt{AFGHODIRMBL})\,(\texttt{CYXWVUTQKEN})\,(\texttt{JSP})\,(\texttt{Z})$$

Reading and references

Wikipedia, "Cyclic permutation," https://en.wikipedia.org/wiki/Cyclic_permutation

https://docs.python.org/3/library/operator.html

Python tips

Here is an example of a class that overrides the addition operator. Notice that the return value of the operation is another object of the class.

```
class myClass():
    def __init__(self,x):
        self.value = x
        return
    def __str__(self):
        return "[" + str(self.value) + "]"
    def __eq__(self,other):
        return (self.value == other.value)
    def __add__(self,other):
        return myClass(self.value + other.value + 1)
```

Here are the names of methods for overriding various operators:

operator	method
==	__eq__
!=	__ne__
~	__invert__
- (negation)	__neg__
+	__add__
- (subtraction)	__sub__
*	__mul__
/	__truediv__
//	__floordiv__

For more, see the Python documentation (link above in references).

Programming tasks

1. Create a Python class for permutations of the alphabet. It needs is an initializer that stores a 26-letter string that is fed to it. If the string does not contain exactly one of each letter, then raise an exception. Override == and `!=` (otherwise, two objects with the same value will be unequal because they are not the same object). Override the conversion to string (`__str__`) so that you can output a permutation in readable form. Override the unary operator ~ to return the inverse of a permutation. Override multiplication to find the composition of two permutations; be very careful that you get the correct results, keeping in mind that permutations are applied in order from right to left. Check to be sure that your routines correctly reproduce the examples that we saw in this section.

2. Create a function that returns a boolean value indicating whether two permutations of the alphabet commute with one another. This can be a stand-alone function or a method inside the class of permutations.

3. Create a function that returns a boolean value indicating whether a permutation is its own inverse. This definitely should be a method inside the class of permutations.

4. Create a function that returns the order of an alphabetic permutation (yes, put this in a method inside the class of permutations). The *order* is the smallest number of times a permutation is composed with itself so that the result is the identity permutation (the unmixed alphabet). Naïvely you might think that this number could be very large, but it is never more than 1260, so it is safe to use a simple loop, rather than some nice trick you may know for finding large exponents. By definition, the order of the identity permutation is 1; for a permutation that is its own inverse the order is 2. If the order of k is n, we write $k^n = e$ as a short-hand version of $k \circ k \circ \cdots \circ k = e$.

5. Create a function to express an alphabetic permutation as a product of disjoint cycles. Put it inside your class as a method. If you are interested to know, the order of a permutation is the least common multiple of the lengths of the cycles in its factorization, provided, of course, that the cycles are disjoint.

Exercises

1. What is the alphabetic permutation for the atbash cipher? How large would you say is the key space of this cipher?

2. How many elements are in S_{26}? Give the full number.

3. Suppose we insist that the first letter of our keys should never be `Z`. Now how large is the key space of the monoalphabetic substitution cipher?

4. What is the size of the keyspace if we permute the first half of the alphabet separately from the second half?

5. Verify with your routines that we found the correct inverse in our example by taking the composition of the key with the putative inverse that we presented. Check both ways of ordering the composition.

6. Find the inverses of each of these alphabetic permutations:

   a. k = `ALIENSBCDFGHJKMOPQRTUVWXYZ`

   b. k = `TALJZBGXUQRHYEICFSDKNPVMOW`

   c. k = `HIFZKYXWEVGJUBTSRDQCAPONML`

   d. k = `WGROTSZLNMPUXKBACJFDIEQYHV`

   e. k = `BFKYDGHJCLEMNPUQSIVRAWXZOT`

7. Find the composition $k_2 \circ k_1$ of each of these pairs of alphabetic permutations. Be careful of the ordering.

   a. k_1 = `SPACELYBDFGHIJKMNOQRTUVWXZ`
      k_2 = `SPROCKETABDFGHIJLMNQUVWXYZ`

   b. k_1 = `SUNLIGHTABCDEFJKMOPQRVWXYZ`
      k_2 = `NEBULAZYXWVTSRQPOMKJIHGFDC`

   c. k_1 = `ONFZRPXJWYQHBLAVEKDSCMGIUT`
      k_2 = `HFQNLIVBWYMGSRTEAKPZXJODUC`

   d. k_1 = `BLUXMPSHATFQGEYDWVOKCJIRZN`
      k_2 = `IYOHVQTLBEWXRKMNPSAFJUDCZG`

8. Determine whether each of these pairs of alphabetic permutations commute:

   a. k_1 = `MNOPQRSTUVWXYZABCDEFGHIJKL`
      k_2 = `VWXYZABCDEFGHIJKLMNOPQRSTU`

b. k_1 = `ZYXWVUTSRQPONMLKJIHGFEDCBA`
   k_2 = `GFEDCBAZYXWVUTSRQPONMLKJIH`

c. k_1 = `LGFJECBMODNAHKIRZPYXWVUTSQ`
   k_2 = `NXWRVUTSQPLKYAZJIDHGFECBMO`

d. k_1 = `OWQXTRKZHFLBYJINGVMSPAUCDE`
   k_2 = `GEARUXIBLDHZJYKMOCNPSQTVFW`

9. Find a pair of alphabetic permutations that commute, other than those from Exercise 8.

10. Find the order of each of these permutations:

    a. k = `XSQVPLKJIHGFTWUECYBMODNARZ`

    b. k = `SRUBCZNDYHJFKIMOPEGQATVWLX`

    c. k = `EYOBUSCVHKPZWNJQFRAXDMTILG`

11. Show that

$$(k_2 \circ k_1)^{-1} = k_1^{-1} \circ k_2^{-1} \tag{18.1}$$

for any two permutations k_1 and k_2. Some call this the "shoes-and-socks theorem," because you put your socks on before your shoes, but then take your shoes off before your socks.

12. Express the permutations in Exercise 6 as products of disjoint cycles.

Unit 19
Modular arithmetic: addition and subtraction

Consider the integers, $\mathbb{Z} = \{ ..., -3, -2, -1, 0, 1, 2, 3, ... \}$. Now pick one, say 12, and call it the *modulus*. Next, rename each integer by the remainder when you divide by the modulus. So $0 \longrightarrow 0$, $1 \longrightarrow 1$, $2 \longrightarrow 2$, $3 \longrightarrow 3$, ..., $11 \longrightarrow 11$. But $12 \longrightarrow 0$, since $12 = 1 \times 12 + 0$, and the remainder is 0. The sequence 0, ..., 11 starts all over again, as $12 \longrightarrow 0$, $13 \longrightarrow 1$, ... The result is a new set $\mathbb{Z}_{12} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$, which contains only 12 members. This is called the set of *residues*. We say things like "13 modulo 12 is 1" or "13 = 1 modulo 12" or "13 = 1 mod 12."

Things get interesting when we start to do arithmetic with the set of residues. For addition, we simply perform the operation as if the numbers are regular integers, but then take the remainder of the result. Continuing with our example of $\mathbb{Z}_{12}$, we can add 7 and 9 to get 16. But 16 is 4 modulo 12. So in $\mathbb{Z}_{12}$, $7 + 9 = 4$.

You might think that subtraction works the same way as addition in modular arithmetic, and you would be correct, but it is worthwhile to look at subtraction in a more rigorous way. After all, there are no negative numbers in modular arithmetic. This approach will also help us when we try to understand division. First, let us generalize and take some modulus m without specifying its value. Notice that we have that

$$x + 0 = x$$

for all x in $\mathbb{Z}_m$. Because 0 does not change the value of a number under addition, we call 0 the *additive identity element*. Now, in regular arithmetic, $x - x = x + (-x) = 0$, so we call $-x$ the *additive inverse* of x. In the set of residues, however, we do not have negative numbers, so there is a *positive* number that takes the role of x's inverse. We might write that number as "$-x$," but that is just notation for additive inverse. What we call "subtraction" is actually addition using the additive inverse.

How do we find the additive inverse of a number x? Well, we want the number y such that

$$x + y = 0 \pmod{m}$$

Even though the modulus m itself does not appear in the set of residues, we can calculate y as

$$y = 0 - x = m - x \pmod{m}$$

In our example of $\mathbb{Z}_{12}$, the inverse of 7 is $12 - 7 = 5$, or $-7 = 5$, so that $7 + 5 = 12 = 0 \pmod{12}$.

If you have been paying attention, you realize that $(\mathbb{Z}_{12}, +_{12})$ is a group (see Unit 17). Sometimes we put the subscript on the + sign to remind us that we are using modular addition (sometimes we can

remember without it). For our cryptographic concerns, we use the set of twenty-six letters, so it is useful to notice that $(\mathbb{Z}_{26}, +_{26})$ is also a group. We will talk more about that in Part III.

Python tips

When dividing integers, the operator `//` gives the quotient, and the operator `%` gives the remainder. Both operators return an integer. For example, `13 // 5` is evaluated as 2, and `13 % 5` is evaluated as 3, since 13 = 2 × 5 + 3.

In programming, modular arithmetic is not as complicated as in the text above. Feel free to use the operators + and − as you normally would, but then modulate the result with the `%` operator.

```
answer = (7 + 9) % 12
answer = (7 - 9) % 12
```

When defining a function, we can have *optional parameters* that need not be given when the function is called. To define them, they must come after the required parameters and must be given a default value. For example:

```
def myFunction (x, y=0):
      return x + y
print (myFunction (10))
print (myFunction (10, 20))
print (myFunction (x=10, y=20))
```

Python is an object-oriented scripting language, and it would be a shame to waste that potential. Furthermore, we can override (redefine) operators for classes of objects that we create. Here is a simple example in which we create a class and override the + and - operators (and the unary - operator responsible for negation), and define how objects should appear when printed. What it does with numbers is just nonsense, but it gives you the basic idea of how to do it. Notice that the result of an operation has to be in the class.

```
class myClass ():
    def __init__ (self, value):
        self.storedvalue = value
    def __neg__ (self):
        return myClass (5)
    def __add__ (self,other):
        return myClass (self.storedvalue + other.storedvalue + 17)
    def __str__ (self):
        return str (self.storedvalue)
```

See docs.python.org/3/library/operator.html for more information about overriding operators.

Programming tasks

1. Go outside and take a walk. If it's raining, open a window so you can hear it while you play an intellectually stimulating game with someone else in your home.

2. Define a Python class for integers modulo 26. Use an optional parameter to allow a different modulus. Override the + and – operators for modular addition and subtraction, and the unary – operator for negation.

Exercises

1. Check carefully your work on programming task #1 and re-do it if necessary.

2. Evaluate:

   a. 528147 + 790378 modulo 62

   b. 72177 − 162737 mod 81

3. a. Find an element of $\mathbb{Z}_{18}$ that is its own (additive) inverse.

   b. Find an element of $\mathbb{Z}_{19}$ that is its own inverse.

   c. What condition must we put on m such that $\mathbb{Z}_m$ has such an element?

Unit 20
Caesar shift cipher

The *Caesar shift cipher*, also known simply as the *Caesar cipher* or the *additive cipher*, can be viewed in two ways. Given its key k, which is a number from 0 to 25, we can think of the Caesar cipher as a monoalphabetic substitution in which the ciphertext alphabet is shifted left by k letters, with wrap-around. For example, if the key is 5, we have

plaintext:	`ABCDEFGHIJKLMNOPQRSTUVWXYZ`
ciphertext:	`FGHIJKLMNOPQRSTUVWXYZABCDE`

A table of the alphabetic permutations that the cipher can use is on the next page. We have given each a label "R_n", where n denotes the number of steps the alphabet has been shifted leftward. No shift is the same as the unaltered alphabet (the identity element in the group of alphabetic permutations). We call a table of alphabetic keys like this one a *tableau* (plural *tableaux*).

Another way to view the Caesar cipher is with modular arithmetic. The plaintext and ciphertext are equivalent to sequences of numbers $\{p_i\}$ and $\{c_i\}$. Encipherment is done with addition:

$$c_i = p_i + k \mod 26$$

Decipherment is done with subtraction:

$$p_i = c_i - k \mod 26$$

When the shift is 13, the cipher is often called *ROT13*. This shift has a special name because 13 is half of 26. Furthermore, since 13 is its own inverse modulo 26, this cipher is its own inverse and called *self-reciprocal*; i.e., encipherment is the same process as decipherment. Also notice that ROT13 is the same as the albam cipher (see Unit 16).

key	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ	label for permutation
0	ABCDEFGHIJKLMNOPQRSTUVWXYZ	$R_0 = e$
1	BCDEFGHIJKLMNOPQRSTUVWXYZA	R_1
2	CDEFGHIJKLMNOPQRSTUVWXYZAB	R_2
3	DEFGHIJKLMNOPQRSTUVWXYZABC	R_3
4	EFGHIJKLMNOPQRSTUVWXYZABCD	R_4
5	FGHIJKLMNOPQRSTUVWXYZABCDE	R_5
6	GHIJKLMNOPQRSTUVWXYZABCDEF	R_6
7	HIJKLMNOPQRSTUVWXYZABCDEFG	R_7
8	IJKLMNOPQRSTUVWXYZABCDEFGH	R_8
9	JKLMNOPQRSTUVWXYZABCDEFGHI	R_9
10	KLMNOPQRSTUVWXYZABCDEFGHIJ	R_{10}
11	LMNOPQRSTUVWXYZABCDEFGHIJK	R_{11}
12	MNOPQRSTUVWXYZABCDEFGHIJKL	R_{12}
13	NOPQRSTUVWXYZABCDEFGHIJKLM	R_{13}
14	OPQRSTUVWXYZABCDEFGHIJKLMN	R_{14}
15	PQRSTUVWXYZABCDEFGHIJKLMNO	R_{15}
16	QRSTUVWXYZABCDEFGHIJKLMNOP	R_{16}
17	RSTUVWXYZABCDEFGHIJKLMNOPQ	R_{17}
18	STUVWXYZABCDEFGHIJKLMNOPQR	R_{18}
19	TUVWXYZABCDEFGHIJKLMNOPQRS	R_{19}
20	UVWXYZABCDEFGHIJKLMNOPQRST	R_{20}
21	VWXYZABCDEFGHIJKLMNOPQRSTU	R_{21}
22	WXYZABCDEFGHIJKLMNOPQRSTUV	R_{22}
23	XYZABCDEFGHIJKLMNOPQRSTUVW	R_{23}
24	YZABCDEFGHIJKLMNOPQRSTUVWX	R_{24}
25	ZABCDEFGHIJKLMNOPQRSTUVWXY	R_{25}

Table 20.1: Tableau for the Caesar cipher. Plaintext letters appear across the top, keys on the left, and ciphertext letters in the bulk of the table. Each row of the bulk is a permutation of the alphabet; along the right side are labels that we have chosen for these permutations; "e" represents the identity permutation, and "R" refers to "rotation."

Reading and references

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, pages 83-84.

Abraham Sinkov, *Elementary Cryptanalysis: A Mathematical Approach*, 2nd edition, revised by Todd Feil, published by Mathematical Association of America, 2009, www.jstor.org/stable/10.4169/j.ctt19b9krf, sections 1.1-1.3.

Wikipedia: en.wikipedia.org/wiki/Caesar_cipher

Crypto Corner: crypto.interactive-maths.com/caesar-shift-cipher.html

Practical Cryptography: practicalcryptography.com/ciphers/caesar-cipher

Programming tasks

1. Write a function to generate the key alphabet for a Caesar cipher from the key number. You might not use this function if you work with modular arithmetic, but it is good practice to write the function.

2. Implement the Caesar cipher and a given key. Use whatever method you like to perform its operations.

Exercises

1. What is the size of the key space for the Caesar cipher?

2. Encipher this text with the key 11:

   A horse is a horse, of course of course, and no one can talk to a horse, of course, unless, of course, that horse of course is the famous Mister Ed.

3. Decipher this text with the key 15:

   ```
   WTLPHDCRTPAXIIATVGTTCQPAADURAPNVJBQNQJINDJHWDJASHTTLWPIWTRP
   CSDIDSPNVJBQNWTRPCLPAZXCIDBPCNQDDZHLXIWWXHEDCTNEPAEDZTNIDDP
   CSXUNDJWPKTPWTPGIIWTCVJBQNHPEPGIDUNDJVJBQN
   ```

4. Find the order of each of the alphabetic permutations (the rotations of the alphabet) that are used with the Caesar cipher. Do you notice a pattern? Make a table of the R_n and their orders and save it for later.

5. Rewrite each of the alphabetic permutations (the rotations of the alphabet) that are used with the Caesar cipher as products of disjoint cycles. Also keep a table of these.

Unit 21
Brute-force attack on the Caesar cipher

A *brute-force attack* involves trying every possible key until an acceptable decryption is found. Since the Caesar cipher has only 26 possible keys (even when we include the identity operation [that leaves the plaintext unaltered]), a brute-force attack takes very little time. Nevertheless, this is a good opportunity to learn the concept of a brute-force attack and how it is implemented.

Programming tasks

1. Implement a brute-force attack on a ciphertext that was encrypted with the Caesar cipher. Use your tetragram-fitness function to find the best possible decryption. Your function should return the key or the plaintext or both.

Exercises

1. Perform a brute-force attack on this ciphertext:

   `WLALYWPWLYWPJRLKHWLJRVMWPJRSLKWLWWLYZ`

   Remember this text; we will see it again.

2. Perform a brute-force attack on this ciphertext:

   `KLYJKPMCLDUFXAESCZFRSSZZADLEESPKZZ`

   Remember this text, too; we will see it again.

Unit 22
Attacking the Caesar cipher with cribs

A *crib* is a word or phrase that you have good reason to believe is contained in the plaintext. To use a crib to break a ciphertext that was encrypted with the Caesar cipher, we try to fit the crib in all possible positions in the ciphertext. For each position, we find the shift needed to decrypt that portion of the ciphertext to match each letter of the crib. If each letter requires the same shift, then we have a good candidate for the key. It is not guaranteed to be the true key, but if we find several candidates, then we can decipher the text with each and use tetragram fitness to pick out the best plaintext.

Let us look at a short example. Here is a piece of encrypted text that we promised you would see again (and you will see it again at least once more):

```
WLALYWPWLYWPJRLKHWLJRVMWPJRSLKWLWWLYZ
```

Let's try to break it with the crib `PIPER`. We line up the crib with the ciphertext at each position until we find one at which all the shifts are equal:

```
ciphertext:   W L A L Y W P W L Y W P J R ...

crib:         P I P E R _ _ _ _ _ _ _ _ _ ...
shifts:       7 3 11 7  7

crib:         _ P I P E R _ _ _ _ _ _ _ _ ...
shifts:         22 18 22 20 5

⋮

crib:         _ _ _ _ _ P I P E R _ _ _ _ ...
shifts:                 7 7 7 7 7
```

We now have a good candidate for the key: 7. If we decipher with that key, we find that the plaintext looks like English, and we are satisfied that we have found the correct key.

Python tips

To take a “slice” of a string (a substring), we use indexing. So, for example, to make a substring that takes the fifth through tenth character:

```
newString = oldString[4:10]
```

Remember that we count from zero, so the fifth character has index 4. The last character that we want to include has index 9, but in Python we have to add one to the index when we specify the last place in a string or array. To duplicate a string (or an array), we also use indexing, but leave the indices out:

```
newString = oldString[:]
```

Programming tasks

1. Implement the attack that takes a ciphertext and a crib and breaks the ciphertext.

Exercises

1. Break this ciphertext with the crib `CUSTOM`.

```
RCCXRLCZJUZMZUVUZEKFKYIVVGRIKJKYVWZIJKZJZEYRSZKVUSPSVCXZREJ
KYVJVTFEUSPKYVRHLZKREZREJNYFTRCCKYVDJVCMVJTVCKJREUKYVKYZIUS
PKYVXRLCJKYVJVUZWWVIWIFDFEVREFKYVIZECREXLRXVTLJKFDJREUCRNJK
YVIZMVIXRIFEEVJVGRIRKVJKYVXRLCJWIFDKYVTVCKJKYVIZMVIJDRIEVRE
UJZVEVJVGRIRKVKYVSVCXZREJWIFDKYVFKYVIJ
```

2. Break this ciphertext. It may or may not include some of these words: `VICTORY SPAIN DISCO EUROVISION`.

```
MFFTUEFUYQOMQEMDIMEQXQOFQPRADFTQRAGDFTFUYQMEPUOFMFADMZPIMEO
AYUZSFAEBMUZFARUZUETFTQIMDAZFTQIMKTQIMEYQFNKMYNMEEMPADERDAY
OADPANMITATMPPQEQDFQPSQZQDMXBAYBQKFTQKUZRADYQPTUYFTMFUFIAGX
PNQQMEUQEFFAFMWQFTQOUFKMFZUSTFNQOMGEQFTQQZQYKTMPNKFTQZZAWZA
IXQPSQARPUEOAADARFTQZUSTFXURQADARFTQNAASUQ
```

Unit 23
Attacking the Caesar cipher with monogram frequencies (χ^2)

Here is an attack on the Caesar cipher that uses only monogram frequencies. The technique is to make a table of the monogram frequencies from the ciphertext, then to shift that frequency table until it resembles that of typical English. In this unit, we will use monogram fitness based on the the χ^2 statistic to measure how well the frequencies match.

Let's work through an example. Consider this ciphertext:

```
LIKHKDGDQBWKLQJFRQILGHQWLDOWRVDBKHZURWHLWLQFLSKHUWKDWLVE
BVRFKDQJLQJWKHRUGHURIWKHOHWWHUVRIWKHDOSKDEHWWKDWQRWDZRUG
FRXOGEHPDGHRXWLIDQBRQHZLVKHVWRGHFLSKHUWKHVHDQGJHWDWWKHLU
PHDQLQJKHPXVWVXEVWLWXWHWKHIRXUWKOHWWHURIWKHDOSKDEHWQDPHO
BGIRUDDQGVRZLWKWKHRWKHUV
```

The frequencies of letters in this ciphertext are

A	0	J	0.020	S	0.016
B	0.020	K	0.097	T	0
C	0	L	0.065	U	0.048
D	0.081	M	0	V	0.048
E	0.020	N	0	W	0.141
F	0.020	O	0.028	X	0.024
G	0.040	P	0.016	Y	0
H	0.133	Q	0.060	Z	0.016
I	0.032	R	0.073		

Let's write the table as a list:

0.000, 0.020, 0.000, 0.081, 0.020, 0.020, 0.040, 0.133, 0.032, 0.020, 0.064, 0.065, 0.000, 0.000, 0.028, 0.016, 0.060, 0.073, 0.016, 0.000, 0.048, 0.048, 0.141, 0.024, 0.000, 0.016

Here is a graph of those frequencies (in **blue**), along side typical English frequencies (in yellow):

We can already see that a shift of the blue bars left by three will give a good match. But let's use the χ^2 statistic and see how that works. Our monogram table for English looks like this (yours may be slightly different) (rounded to three decimal places):

> 0.081, 0.015, 0.024, 0.045, 0.123, 0.021, 0.021, 0.064, 0.070, 0.001, 0.009, 0.041, 0.016, 0.068, 0.077, 0.017, 0.001, 0.057, 0.062, 0.091, 0.029, 0.009, 0.025, 0.002, 0.021, 0.001

Now, if we shift the frequencies from the ciphertext leftward by one, we get

> 0.020, 0.000, 0.081, 0.020, 0.020, 0.040, 0.133, 0.032, 0.020, 0.064, 0.065, 0.000, 0.000, 0.028, 0.016, 0.060, 0.073, 0.016, 0.000, 0.048, 0.048, 0.141, 0.024, 0.000, 0.016, 0.000

The χ^2 between this and the English table is 18.02, indicating a poor fit. We can continue in this manner, and for each shift we obtain these values for χ^2:

shift	**χ^2**	**shift**	**χ^2**	**shift**	**χ^2**
0	6.51	9	3.91	18	7.88
1	18.02	10	10.27	19	3.83
2	6.73	11	13.92	20	19.38
3	0.10	12	10.99	21	9.52
4	9.81	13	31.47	22	4.00
5	5.17	14	3.91	23	24.56
6	25.14	15	2.90	24	21.31
7	7.67	16	3.91	25	11.63
8	24.11	17	22.50		

The best fit (lowest χ^2) is for a shift of three. We conclude that the key is 3. To get a feel for what a good fit looks like, here is a graph of the shifted frequency table compared to typical English:

Programming tasks

1. Implement the attack. Make sure that you shift the table in the correct direction; generate some ciphertexts and test it to be certain.

Exercises

1. Finish decrypting the ciphertext in the example above.

2. Apply the attack to this ciphertext:

```
QEBZXBPXOZFMEBOFPKXJBAXCQBOGRIFRPZXBPXOTELXZZLOAFKDQLPRBQLK
FRPRPBAFQTFQEXPEFCQLCQEOBBQLMOLQBZQJBPPXDBPLCJFIFQXOVPFDKFC
FZXKZBTEFIBZXBPXOPTXPQEBCFOPQOBZLOABARPBLCQEFPPZEBJBLQEBOPR
YPQFQRQFLKZFMEBOPXOBHKLTKQLEXSBYBBKRPBABXOIFBOFQFPRKHKLTKEL
TBCCBZQFSBQEBZXBPXOZFMEBOTXPXQQEBQFJBYRQFQFPIFHBIVQLEXSBYBB
KOBXPLKXYIVPBZROBKLQIBXPQYBZXRPBJLPQLCZXBPXOPBKBJFBPTLRIAEX
SBYBBKFIIFQBOXQB
```

3. Apply the attack to this ciphertext, which we have seen before:

```
WLALYWPWLYWPJRLKHWLJRVMWPJRSLKWLWWLYZ
```

   You should find that the attack fails. Why did it fail?

4. Apply the attack to this ciphertext, which we have seen before:

```
KLYJKPMCLDUFXAESCZFRSSZZADLEESPKZZ
```

   You should find that the attack fails. Why did it fail? Now you should understand why we do not recommend using the χ^2 statistic for cracking ciphertexts, and we will not use it again in this book.

Unit 24
Attacking the Caesar cipher with monogram frequencies (cos θ)

Here is an attack on the Caesar cipher that uses only monogram frequencies. The technique is to make a table of the monogram frequencies from the ciphertext, then to shift that frequency table until it resembles that of typical English. In this unit, we will use monogram fitness based on the cosine of the angle between vectors to measure how well the frequencies match. In the future, this will be our preferred method for evaluating monogram fitness (we will no longer use χ^2).

Let's work through an example. Consider this ciphertext:

```
LIKHKDGDQBWKLQJFRQILGHQWLDOWRVDBKHZURWHLWLQFLSKHUWKDWLVE
BVRFKDQJLQJWKHRUGHURIWKHOHWWHUVRIWKHDOSKDEHWWKDWQRWDZRUG
FRXOGEHPDGHRXWLIDQBRQHZLVKHVWRGHFLSKHUWKHVHDQGJHWDWWKHLU
PHDQLQJKHPXVWVXEVWLWXWHWKHIRXUWKOHWWHURIWKHDOSKDEHWQDPHO
BGIRUDDQGVRZLWKWKHRWKHUV
```

The frequencies of letters in this ciphertext are

A	0	J	0.020	S	0.016
B	0.020	K	0.097	T	0
C	0	L	0.065	U	0.048
D	0.081	M	0	V	0.048
E	0.020	N	0	W	0.141
F	0.020	O	0.028	X	0.024
G	0.040	P	0.016	Y	0
H	0.133	Q	0.060	Z	0.016
I	0.032	R	0.073		

Let's write the table as a 26-dimensional vector:

(0.000, 0.020, 0.000, 0.081, 0.020, 0.020, 0.040, 0.133, 0.032, 0.020, 0.064, 0.065, 0.000, 0.000, 0.028, 0.016, 0.060, 0.073, 0.016, 0.000, 0.048, 0.048, 0.141, 0.024, 0.000, 0.016)

Here is a graph of those frequencies (in **blue**), along side typical English frequencies (in yellow):

We can already see that a shift of the blue bars left by three will give a good match. But let's use the monogram fitness and see how that works. Our monogram table for English looks like this (yours may be slightly different) (rounded to three decimal places):

(0.081, 0.015, 0.024, 0.045, 0.123, 0.021, 0.021, 0.064, 0.070, 0.001, 0.009, 0.041, 0.016, 0.068, 0.077, 0.017, 0.001, 0.057, 0.062, 0.091, 0.029, 0.009, 0.025, 0.002, 0.021, 0.001)

Now, if we shift the frequencies from the ciphertext leftward by one, we get

(0.020, 0.000, 0.081, 0.020, 0.020, 0.040, 0.133, 0.032, 0.020, 0.064, 0.065, 0.000, 0.000, 0.028, 0.016, 0.060, 0.073, 0.016, 0.000, 0.048, 0.048, 0.141, 0.024, 0.000, 0.016, 0.000)

The cos θ between this vector and the English table is 0.393, indicating a poor fit. We can continue in this manner, and for each shift we obtain these values for cos θ:

shift	**cos θ**	**shift**	**cos θ**	**shift**	**cos θ**
0	0.493	9	0.512	18	0.693
1	0.393	10	0.547	19	0.569
2	0.544	11	0.452	20	0.443
3	0.970	12	0.434	21	0.435
4	0.593	13	0.478	22	0.608
5	0.443	14	0.648	23	0.499
6	0.485	15	0.580	24	0.444
7	0.589	16	0.584	25	0.600
8	0.493	17	0.550		

The best fit (largest cos θ) is for a shift of three. We conclude that the key is 3. To get a feel for what a good fit looks like, here is a graph of the shifted frequency table compared to typical English:

We may ask what would be a good cut-off for deciding if we have found the correct plaintext. For that purpose, we have prepared this graph of the cosine value as a function of shift, where the frequencies used were the ones we found in Unit 5 from our textual corpus. To be clear, we are comparing the same set of frequencies with a shifted copy. It appears that 0.8 is a good threshold for discrimination. A comparison to the table of cosine values for our example suggests that this threshold is appropriate for real data as well.

Programming tasks

1. Implement the attack. Make sure that you shift the table in the correct direction; generate some ciphertexts and test it to be certain.

Exercises

1. Finish decrypting the ciphertext in the example above.

2. Apply the attack to this ciphertext:

```
QEBZXBPXOZFMEBOFPKXJBAXCQBOGRIFRPZXBPXOTELXZZLOAFKDQLPRBQL
KFRPRPBAFQTFQEXPEFCQLCQEOBBQLMOLQBZQJBPPXDBPLCJFIFQXOVPFDK
FCFZXKZBTEFIBZXBPXOPTXPQEBCFOPQOBZLOABARPBLCQEFPPZEBJBLQEB
OPRYPQFQRQFLKZFMEBOPXOBHKLTKQLEXSBYBBKRPBABXOIFBOFQFPRKHKL
TKELTBCCBZQFSBQEBZXBPXOZFMEBOTXPXQQEBQFJBYRQFQFPIFHBIVQLEX
SBYBBKOBXPLKXYIVPBZROBKLQIBXPQYBZXRPBJLPQLCZXBPXOPBKBJFBPT
LRIAEXSBYBBKFIIFQBOXQB
```

3. Apply the attack to this ciphertext, which we have seen before:

```
WLALYWPWLYWPJRLKHWLJRVMWPJRSLKWLWWLYZ
```

   You should find that the attack fails. Why did it fail? When we used the brute-force attack, we succeeded. From this we learn that using monogram fitness is useful for longer ciphertexts, but not very reliable for shorter texts. Nevertheless, this attack is necessary if you have a ciphertext that has been encrypted with both a Caesar cipher and a transposition cipher, which rearranges the letters of the text and thereby makes it impossible to use tetragram fitness.

4. Apply the attack to this ciphertext, which we have seen before:

```
KLYJKPMCLDUFXAESCZFRSSZZADLEESPKZZ
```

   You should find that the attack succeeds. When we used the monogram fitness based on the χ^2 statistic, we failed to decrypt this ciphertext.

Unit 25
Caesar salad cipher

Suppose now we build a Caesar shift cipher not with the usual alphabet, but with one that was scrambled (saladified) with a keyword or in some other way. This is not a standard cipher, but makes an excellent stepping-stone on the way to the quagmire and other polyalphabetic ciphers (in the next part of this book). To understand this cipher, take this scrambled alphabet for example:

```
FLYINGSAUCERBDHJKMOPQTVWXZ
```

If the shift we use is 5, then the substitution follows this scheme:

```
FLYINGSAUCERBDHJKMOPQTVWXZ
↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
GSAUCERBDHJKMOPQTVWXZFLYIN
```

If we straighten out the plaintext alphabet (the one on top), then we find the equivalent key for a monoalphabetic substitution cipher:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
BMHOJGEPUQTSVCWXZKRFDLYIAN
```

Note that there is a different alphabetic permutation for each shift; they are not merely shifted versions of each other. In Table 17.2 we see them all. Notice that the identity permutation appears when the key is zero. To complete the example, let’s encrypt a short message with the above key:

```
plaintext:    DRESSING ON THE SIDE
ciphertext:   OKJRRUCE WC FPJ RUOJ
```

If we know the scrambled alphabet or the keyword used to scramble it, then we can break the cipher as we did the Caesar shift cipher with monogram frequencies. The first step is to reorder the table of frequencies to match to order of the scrambled alphabet. Then, matching the frequency distribution of the ciphertext to it works in the same way as with the normal Caesar cipher.

key	plaintext alphabet FLYINGSAUCERBDHJKMOPQTVWXZ
0	FLYINGSAUCERBDHJKMOPQTVWXZ
1	LYINGSAUCERBDHJKMOPQTVWXZF
2	YINGSAUCERBDHJKMOPQTVWXZFL
3	INGSAUCERBDHJKMOPQTVWXZFLY
4	NGSAUCERBDHJKMOPQTVWXZFLYI
5	GSAUCERBDHJKMOPQTVWXZFLYIN
6	SAUCERBDHJKMOPQTVWXZFLYING
7	AUCERBDHJKMOPQTVWXZFLYINGS
8	UCERBDHJKMOPQTVWXZFLYINGSA
9	CERBDHJKMOPQTVWXZFLYINGSAU
10	ERBDHJKMOPQTVWXZFLYINGSAUC
11	RBDHJKMOPQTVWXZFLYINGSAUCE
12	BDHJKMOPQTVWXZFLYINGSAUCER
13	DHJKMOPQTVWXZFLYINGSAUCERB
14	HJKMOPQTVWXZFLYINGSAUCERBD
15	JKMOPQTVWXZFLYINGSAUCERBDH
16	KMOPQTVWXZFLYINGSAUCERBDHJ
17	MOPQTVWXZFLYINGSAUCERBDHJK
18	OPQTVWXZFLYINGSAUCERBDHJKM
19	PQTVWXZFLYINGSAUCERBDHJKMO
20	QTVWXZFLYINGSAUCERBDHJKMOP
21	TVWXZFLYINGSAUCERBDHJKMOPQ
22	VWXZFLYINGSAUCERBDHJKMOPQT
23	WXZFLYINGSAUCERBDHJKMOPQTV
24	XZFLYINGSAUCERBDHJKMOPQTVW
25	ZFLYINGSAUCERBDHJKMOPQTVWX

Table 25.1: First version of the tableau for our example of the Caesar salad cipher. Plaintext letters appear across the top, keys on the left, and ciphertext letters in the bulk of the table.

key	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	ABCDEFGHIJKLMNOPQRSTUVWXYZ
1	UDEHRLSJNKMYOGPQTBAVCWXZIF
2	CHRJBYAKGMOIPSQTVDUWEXZFNL
3	EJBKDIUMSOPNQATVWHCXRZFLGY
4	RKDMHNCOAPQGTUVWXJEZBFLYSI
5	BMHOJGEPUQTSVCWXZKRFDLYIAN
6	DOJPKSRQCTVAWEXZFMBLHYINUG
7	HPKQMABTEVWUXRZFLODYJINGCS
8	JQMTOUDVRWXCZBFLYPHIKNGSEA
9	KTOVPCHWBXZEFDLYIQJNMGSARU
10	MVPWQEJXDZFRLHYINTKGOSAUBC
11	OWQXTRKZHFLBYJINGVMSPAUCDE
12	PXTZVBMFJLYDIKNGSWOAQUCEHR
13	QZVFWDOLKYIHNMGSAXPUTCERJB
14	TFWLXHPYMINJGOSAUZQCVERBKD
15	VLXYZJQIONGKSPAUCFTEWRBDMH
16	WYZIFKTNPGSMAQUCELVRXBDHOJ
17	XIFNLMVGQSAOUTCERYWBZDHJPK
18	ZNLGYOWSTAUPCVERBIXDFHJKQM
19	FGYSIPXAVUCQEWRBDNZHLJKMTO
20	LSIANQZUWCETRXBDHGFJYKMOVP
21	YANUGTFCXERVBZDHJSLKIMOPWQ
22	IUGCSVLEZRBWDFHJKAYMNOPQXT
23	NCSEAWYRFBDXHLJKMUIOGPQTZV
24	GEARUXIBLDHZJYKMOCNPSQTVFW
25	SRUBCZNDYHJFKIMOPEGQATVWLX

Table 25.2: Second version of the tableau for our example of the Caesar salad cipher. In this version, the plaintext letters appearing across the top have been put into order. Keys appear on the left, and ciphertext letters are in the bulk of the table. Each row of the bulk is a permutation of the alphabet. Notice that the identity permutation appears for key = 0. Also notice that a shifted copy of the scrambled alphabet appears in each column.

Python tips

One way to shift a string (with wrap-around) by `n` steps is

```
newstring = oldstring[n:] + oldstring[:n]
```

The command takes the first `n` characters and concatenates them to the end of the other part of the string. The same trick works to shift a list.

Programming tasks

1. Implement the cipher.

2. Write a routine that takes your monogram frequency table and rearranges is according to a permutation of the alphabet.

3. Implement the attack mentioned in this unit. The scrambled alphabet is an input.

4. Suppose that a keyword is used to scramble the alphabet, and that the remaining letters of the alphabet appear in order after the keyword, as in our example. Implement an attack for the cipher when the keyword is unknown. Use nested loops, with the outer loop running over the words in your word list. The inner loop is the routine in Task 3 above.

Exercises

1. Encipher this text with keyword `FLYINGZEBRAS` and shift 17.

   Four attempts were made to cross the zebra with Shetland ponies: only one succeeded. The hybrid was a smaller edition of Romulus.

   (from *Pearls & Parasites* by A. E. Shipley)

2. Decipher this text with keyword `ZEBRASDONTFLY` and shift 9.

   ```
   ZDKJKMPQQBHQRUUKKFUFQBOAOWCUDKTOATUQXVMOZKUFWUZQBBZDKZOFKOE
   KQPZDQZDKUXWZWXBRGOAOWCUMCZSCOBZRWEUWFKVPKQVECBAPOFKZDKFKPK
   UCSSKUZOWXFQTKUFKEQBZKPOXHPOZOXSWEZDOUHOATKVMPCZKQBZDWCSDDK
   DQUKUAQLKVZDKDQBZKPDKHKQPUEWPBOEKQAWXGOAZUUCOZ
   ```

3. Break this ciphertext. The keyword is `SALADIFICATION` (drop repeated letters to get `SALDIFCTON`).

   ```
   ZWYILTQSKGQBVLYVLPZWWZZUUQVRKXZSPETLUSZSLQBLTZBLVTOZBYQSYYI
   QSQPQRQUBVDIYRKXZTOZESULUQSUYILBLVPSZYIVSDTZBLLJQPOLBQYVSDY
   IQSQSVSWLBVZBZSLYILPQRQUVPYILOBVSXLZWYILTLSEQSUQRYIZEDIQUVS
   ```

```
SLBALOLBWLXYVSLGLBKZYILBULYQVRLJXLOYYILPQRQUYILQWWQVBHVRRAL
GZYLUQWQVREBLVWYIQYALOZZB
```

4. Break this ciphertext.

```
MIJIJIIJQFHIJKLRIXIQHLYFGLATLRWUJRSHFYSFNHMIJIJIRIJQPNAATLS
HNUAUJAFRXUPLRZIHLIJQPNAFJLTIXSQFDLJFHIJKLRUJAFRYIXXZULPLRI
JQHLYFGLATLRLLQRZNAFHIJKLRIJQMIJIJIRIXALHJIALXCUJAFIKXIRRQU
RTEUATRNKIHRZHUJWXLQMLAELLJRLAATLYFJUPLSFHFJLTFNHIJQATLJRLH
GL
```

Unit 26
Greatest common divisor

The *greatest common divisor* (gcd) of two integers is the largest integer that divides them both evenly (without remainder).

One way to find the gcd of two integer is to write out the prime-number factorization of each and select the largest set of factors that is contained in both. For example,

$$84 = 2 \times 2 \times 3 \times 7$$
$$360 = 2 \times 2 \times 2 \times 3 \times 3 \times 5$$

The largest subset contained in both factorizations is {2, 2, 3}, so the gcd of 84 and 360 is $2 \times 2 \times 3 = 12$.

Another way to find the gcd is with Euclid's algorithm, which relies on the fact that if d divides m and n, then it must also divide $m-n$. It is easy to implement in a Python script. Here is the algorithm for two integers m and n:

1. while n is not 0
    a. set m equal to m modulo n
    b. swap m and n
2. once n is 0, output m

The *least common multiple* (lcm) of two integers is the smallest number that is a multiple of both of them. We can find the lcm from the prime-number factorizations as well, by selecting the smallest set of factors that is a superset of each factorizations. For our example of 84 and 360, we see that we need three 2's to accommodate the 2's in 360, two 3's, one 5 and one 7. So the lcm of 84 and 360 is $2 \times 2 \times 2 \times 3 \times 3 \times 5 \times 7 = 2520$. An interesting fact is that

$$\text{gcd}\,(m, n) \times \text{lcm}\,(m, n) = m \times n$$

We say that two numbers are *coprime* if their gcd is one. In this case, the two numbers have no factors in common, other than 1, and neither number evenly divides the other.

Python tips

Python has a built-in limit on the number of levels of recursion. If you are working with large numbers, and trying to find a gcd, then you could exceed this limit and get an error. (Recursion is when a function calls itself. When some condition becomes true, the function exits all the way back to the main control flow.) We recommend that you implement Euclid's algorithm without recursion.

Python allows variables to hold boolean values. These values are either `True` or `False`. Furthermore, functions can return boolean values. Notice that the following two short code blocks are equivalent; both return `True` if the value of `x` is one and `False` otherwise.

code block 1:

```
if x == 1:
      return True
else:
      return False
```

code block 2:

```
return (x == 1)
```

Reading and references

Wikipedia

en.wikipedia.org/wiki/Greatest_common_divisor
en.wikipedia.org/wiki/Euclidean_algorithm
en.wikipedia.org/wiki/Least_common_multiple

Programming tasks

1. Write a function that calculates the gcd of two integers. Use whatever method you prefer.

2. Write a function that finds the lcm of two integers. Feel free to use your function for the gcd.

3. Write a function that returns a boolean value that indicates whether two integers are coprime. This can be a very short function.

Exercises

1. Randomly generate a few pairs of positive integers. For each pair, find the gcd and lcm. Verify that the product of the gcd and lcm equals the product of the pair.

Unit 27
Modular arithmetic: multiplication and division

Let's return to our example of $\mathbb{Z}_{12} = \{0, 1, ..., 11\}$, and construct a multiplication table. Remember that what we need to do is find the product of two members of the set, and then find the remainder when we divide the product by the modulus 12.

$\times_{12}$	0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	10	11
2	0	2	4	6	8	10	0	2	4	6	8	10
3	0	3	6	9	0	3	6	9	0	3	6	9
4	0	4	8	0	4	8	0	4	8	0	4	8
5	0	5	10	3	8	1	6	11	4	9	2	7
6	0	6	0	6	0	6	0	6	0	6	0	6
7	0	7	2	9	4	11	6	1	8	3	10	5
8	0	8	4	0	8	4	0	8	4	0	8	4
9	0	9	6	3	0	9	6	3	0	9	6	3
10	0	10	8	6	4	2	0	10	8	6	4	2
11	0	11	10	9	8	7	6	5	4	3	2	1

(Sometimes we put a subscript on the multiplication sign to remind us that we are working with modular multiplication.)

That wasn't so hard. But what about division? Fractions do not exist in $\mathbb{Z}_{12}$, so what are we to do? In the same way that we defined the additive inverse, we must now do the same for multiplication. Remember that the multiplicative inverse of some number x is another number y such that $x \times y = 1$. The multiplicative identity element is 1. We write "x^{-1}" for the inverse of x, but we do not intend that it should be interpreted as "1 over x" or "1 divided by x." From this point of view, division is actually multiplication by an inverse.

If we look at the multiplication table for $\mathbb{Z}_{12}$ we can see, for example, that $5 \times 5 = 1$, so the inverse of 5 is 5 itself. But what about the inverse of 3? There is no number y such that $3 \times y = 1$. In this case, we are forced to say that 3 does not have an inverse. This means that we cannot divide by three in $\mathbb{Z}_{12}$. But what if we try to do it anyway? Notice that $3 \times 3 = 9$, and $3 \times 7 = 9$, and $3 \times 11 = 9$. Then if we want to find 9 / 3, we have to face the fact that there are three possible answers. Since the quotient is not well defined, it cannot exist.

In general, an element x of $\mathbb{Z}_m$ has an inverse if and only if gcd $(x, m) = 1$. This also means that if m is prime, all elements (except 0) of $\mathbb{Z}_m$ are invertible.

To find the inverse of an element in $\mathbb{Z}_m$ (or to determine whether it exists), Euclid comes to the rescue again. His *extended Euclidean algorithm* for finding the inverse of x in $\mathbb{Z}_m$ is presented here:

1. set $t = 0$, $t' = 1$, $r = m$, $r' = x$
2. while r' is nonzero
   a. set $q = r / r'$ (discard the remainder)
   b. set new values of t and t' to the current values of t' and $t - q \cdot t'$ (respectively)
   c. set new values of r and r' to the current values of r' and $r - q \cdot r'$ (respectively)
3. if r is nonzero, then x is not invertible; exit
4. if $t < 0$, add m to t
5. t is the inverse of x

If we remove from the multiplication table above all rows and columns for elements that do not have inverses, we get this table:

$\times_{12}$	1	5	7	11
1	1	5	7	11
5	5	1	11	7
7	7	11	1	5
11	11	7	5	1

If we check, we will find that this is a group in which every element is its own inverse. We call it ($\mathbb{Z}_{12}^*$, $\times_{12}$). The fact that each element is its own inverse is an accident. When we look at ($\mathbb{Z}_{26}^*$, $\times_{26}$) we will find that it, too, is a group, but it does not share that property.

Python tips

Python allows parallel assignment. For example, to set `x` to 1 and `y` to old value of `x`, this one statement suffices:

```
x, y = 1, x
```

This technique should make implementing steps 2b and 2c in Euclid's extended algorithm easier.

To override multiplication (`*`) and inversion (`~`) for a class of objects, redefine the `__mul__` and `__invert__` operators. For division, there are two operators, `/` and `//`, and you will need to redefine `__truediv__` and `__floordiv__` if you want to be able to use both. See docs.python.org/3/library/operator.html for more information about overriding operators.

Reading and references

en.wikipedia.org/wiki/Extended_Euclidean_algorithm

Programming tasks

1. Write a function that finds the multiplicative inverse (if it exists) of an integer with a given modulus. If the inverse does not exist, it should have a way of telling the program that called the function that it does not; one way to do this is to return the boolean value `False`.

2. Return to your class of integers modulo 26. Override the multiplication operator `*` to perform modular multiplication and the unary operator `~` to return the multiplicative inverse. Override the two division operators to perform multiplication by an inverse (check that you have the operands in the correct order). If an attempt is made to divide by a noninvertible number, raise an exception.

Exercises

1. Randomly generate some integer pairs such that the first of each pair is smaller than the second. For each pair find the multiplicative inverse of the first number modulo the second.

2. Generate the multiplication table for $\mathbb{Z}_{26}$. Reduce it to only include invertible elements, and then you have the table for $\mathbb{Z}_{26}^{*}$.

Unit 28
Group isomorphism

An *isomorphism* is a one-to-one mapping that preserves the algebraic structure. If we have two groups $(G, \blacksquare)$ and $(H, \triangledown)$, they are isomorphic if they have the same number of elements and there is a one-to-one mapping φ: $G \to H$ such that for every x and y in G,

$$\varphi(x \blacksquare y) = \varphi(x) \triangledown \varphi(y) \tag{28.1}$$

Notice that the operation on the left-hand side is the binary operation in the source group, and the operation on the right-hand side is from the target group. The symbol we have for isomorphism is $\cong$, and we write $(G, \blacksquare) \cong (H, \triangledown)$. If these two groups are isomorphic, it also means that the identity element in the source is mapped to the identity element in the target:

$$\varphi(e_G) = e_H$$

and that inverses are mapped to inverses:

$$\varphi(x^{-1}) = \varphi(x)^{-1}$$

(for all x in G). Inversion on the left-hand side is with respect to the operation of the source group, and on the right-hand side is with respect to the operation of the target group.

Isomorphism is a powerful tool that will allow us to use what we know about the structure of one set on another.

An isomorphism from a group to itself is called an *automorphism*. For example, $n \to 5 \times n \pmod{26}$ is an automorphism of $(\mathbb{Z}_{26}, +_{26})$:

$$\varphi(x + y) = 5(x + y) = 5x + 5y = \varphi(x) + \varphi(y)$$

The astute reader might interject “Hey! You used the distributive property without showing that it holds in modular arithmetic.” True, but it’s OK. Trust me.

Exercises

1. Show that the set of rotations of the alphabet that we saw in the Caesar cipher together with composition is isomorphic to $(\mathbb{Z}_{26}, +_{26})$:

$$(\{R_n\}, \circ) \cong (\mathbb{Z}_{26}, +_{26}) \tag{28.2}$$

   Choose a mapping and check all 676 possibilities in equation 28.1, perhaps with a Python script. The obvious mapping $R_n \to n$ works, but there are others. Can you find one? Equation 28.2 means that the Caesar ciphers form a subset of S_{26}. This is an important result, and we will see it again in Part III.

2. Show that

$$(\mathbb{Z}_{26}^{*}, \times_{26}) \cong (\mathbb{Z}_{12}, +_{12}) \tag{28.3}$$

   Find a mapping that works and check all 144 possibilities in equation 28.1, perhaps with a Python script.

3. We saw above that multiplication by 5 is an automorphism of $\mathbb{Z}_{26}$. Find more such automorphisms. Hint: There are twelve. Can you relate them to the results of Exercise 1?

Unit 29
Multiplication cipher

The *multiplication cipher* is a monoalphabetic substitution cipher that replaces each letter by one found by multiplying its value by a key integer modulo 26. It is the multiplicative analog of the additive Caesar shift cipher. Taking our plaintext and ciphertext to be sequences of numbers $\{p_i\}$ and $\{c_i\}$ equivalent to letters in the usual way (A=0, B=1, ..., Z=25), we encipher with modular multiplication:

$$c_i = p_i \times k \mod 26$$

Because the cipher must produce decipherable ciphertexts, the key must be invertible modulo 26. This limits us to twelve choices, which are the numbers coprime with 26: 1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, and 25. Decipherment is done by multiplying by the inverse of the key (remember that there is no such thing as division):

$$p_i = c_i \times k^{-1} \mod 26$$

Here is a tableau of the twelve possible alphabetic permutations for the multiplication cipher with twenty-six letters. The first one, for key equal to 1, is the identity. We have given them all labels, so that we can refer to them more easily in the future.

key	**plaintext alphabet** ABCDEFGHIJKLMNOPQRSTUVWXYZ	**label for permutation**
1	ABCDEFGHIJKLMNOPQRSTUVWXYZ	$M_1 = e$
3	ADGJMPSVYBEHKNQTWZCFILORUX	M_3
5	AFKPUZEJOTYDINSXCHMRWBGLQV	M_5
7	AHOVCJQXELSZGNUBIPWDKRYFMT	M_7
9	AJSBKTCLUDMVENWFOXGPYHQZIR	M_9
11	ALWHSDOZKVGRCNYJUFQBMXITEP	M_{11}
15	APETIXMBQFUJYNCRGVKZODSHWL	M_{15}
17	ARIZQHYPGXOFWNEVMDULCTKBSJ	M_{17}
19	ATMFYRKDWPIBUNGZSLEXQJCVOH	M_{19}
21	AVQLGBWRMHCXSNIDYTOJEZUPKF	M_{21}
23	AXUROLIFCZWTQNKHEBYVSPMJGD	M_{23}
25	AZYXWVUTSRQPONMLKJIHGFEDCB	M_{25}

Table 29.1: Alphabetic permutations for the multiplication cipher.

Reading and references

Abraham Sinkov, *Elementary Cryptanalysis: A Mathematical Approach*, 2nd edition, revised by Todd Feil, published by Mathematical Association of America, 2009, www.jstor.org/stable/10.4169/j.ctt19b9krf, section 1.6.

Programming tasks

1. Write a function that returns a boolean value to indicate whether a key is valid. Surely you can use something that you have written before and put a wrapper around it.

2. Implement the multiplication cipher. Be sure to check for valid keys and raise an exception if an invalid key is used.

3. Implement a brute-force attack on the multiplication cipher.

4. Make a modified copy of your attack on the Caesar cipher from Unit 24 to attack the multiplication cipher.

Exercises

1. What is the key space if we have twenty-seven letters? Twenty-nine?

2. Encipher with key 15:

   > It has been said that "the human mind has never invented a labor-saving machine equal to algebra." If this be true, it is but natural and proper that an age like our own, characterized by the multiplication of labor-saving machinery, should be distinguished by an unexampled development of this most refined and most beautiful of machines.
   >
   > (from "On Multiple Algebra," an address before The American Association for the Advancement of Science, August, 1886, by J. Willard Gibbs)

3. Decipher with key 7:

```
ZCAJOKDDENQWAPCXANVZCVENDXCWAGCGANNCPAWWUJDWDCGOKDDENQWWUJA
PAWDCGBCPADKPCANVGUEWDKPCAPCOUNOCPNCVDXCPCAPCOUGBAPADERCZMJ
CYWBCOECWENYXEOXDXCMJUPGDXCGUWDARAEZAHZCGCANWUJGKZDEBZEOADE
UNENWUGCOAWCWRAPECQADEUNYEZZNUDHCPCBPUVKOCVHMDXCPUUDCVZCAJ
```

4. Break this ciphertext:

```
UGXXMNLGGLSAKJRGDOAXSMOJOAKVXGOOGLMORGURIOGJTANOWTGOOMINMOB
ITWMZGNBITGZGTKVXGOOMNWBXIUOMNJIJRGOIEXAOJRGQINOGYEGNQGIBLM
ZMNGBITWMZGNGOOJRGUITLMNJRGRGVTGUTGNLGTGLVXGOOGLMOMNJRGDXET
AXNESVGTJIORIUJRAJJRGTGMONIJINGVXGOOMNWINXKVEJSEXJMDXMGLVXG
```

OOMNWOANLSEXJMDXMGLSGTQMGOAXXODTMNWMNWBTISJRMOINGOIETQGJRGB
ITWMZGNGOOIBOMN

Unit 30
Affine cipher

The *affine cipher* extends the Caesar and multiplication ciphers to include the use of both modular addition and multiplication. When we treat the characters of the plaintext $\{p_i\}$ and ciphertext $\{c_i\}$ as integers modulo 26, then encipherment under the affine cipher is done with this equation:

$$c_i = a\, p_i + b \mod 26$$

Decipherment is accomplished by solving the above equation for p_i:

$$p_i = a^{-1} (c_i - b) \mod 26$$

where the key consists of the multiplier a and the shift b. Note that a must invertible, so gcd (a, 26) must be equal to one.

Another way to handle the affine cipher is to construct a key alphabet and treat the cipher as a monoalphabetic substitution (which it *is*). It's not complicated. Start with A and take every a^{th} letter; then shift (with rollover) so that the letter whose number is b is at the beginning. For example, if $a = 3$ and $b = 5$, write down every third letter, starting with A:

```
ADGJMPSVYBEHKNQTWZCFILORUX
```

Then, shift so that the fifth letter (starting from zero) of the normal alphabet is at the beginning:

```
FILORUXADGJMPSVYBEHKNQTWZC
```

There are some special cases of the affine cipher:

- $a = 1$, $b = 0$: the identity operation
- $a = 1$: Caesar shift cipher with key = b
- $a = 25$, $b = 25$: atbash cipher
- $b = 0$: multiplicative cipher with key = a

Reading and references

Abraham Sinkov, *Elementary Cryptanalysis: A Mathematical Approach*, 2nd edition, revised by Todd Feil, published by Mathematical Association of America, 2009, www.jstor.org/stable/10.4169/j.ctt19b9krf, section1.8.

en.wikipedia.org/wiki/Affine_cipher

practicalcryptography.com/ciphers/affine-cipher

crypto.interactive-maths.com/affine-cipher.html

Programming tasks

1. Write a function that verifies a key for the affine cipher and returns a boolean value indicating whether the key is valid or invalid.

2. Write a function that takes a key for an affine cipher and returns an equivalent alphabet key for a monoalphabetic substitution. This function should first call the function from the previous task to verify the key before proceeding.

3. Implement the affine cipher. Use whatever method you prefer to perform the encryption and decryption. You should verify the key before proceeding. One way to implement the cipher is to apply a multiplication cipher followed by a Caesar cipher; if you do it this way, keep in mind that decryption must use the Caesar before the multiplication cipher.

Exercises

1. How large is the key space for the affine cipher? Include only valid keys for which the multiplier is invertible. Include the identity operation (the key that leaves the plaintext unchanged under encipherment).

2. Encipher this text with multiplier 11 and shift 9:

   > An affine transformation is an automorphism of an affine space which preserves both the dimension of any affine subspaces and the ratios of the lengths of parallel line segments. Consequently, sets of parallel affine subspaces remain parallel after an affine transformation. An affine transformation does not necessarily preserve angles between lines or distances between points, though it does preserve ratios of distances between points lying on a straight line.

3. Decipher this text with multiplier 15 and shift 3:

```
TSLCZFRCEFRPECCETNUMDTQCLKCVFRMWTQGFMGLNFBLBDCELBDCTHDMQFQN
LQNLCEDCQFFQLRQWLYNCDQWNTQNCLDWELYLTNDIFXLVEZWTWCELHETHXLQH
YFNNCELBFLSTRNNCYTUCFPLCCFCELNDBLNTWL
```

4. If we extend the alphabet with a space character so that it now contains 27 letters, how many valid keys are there?

5. Suppose that we encrypt a text by an affine cipher with key (m, n), and then encrypt the resulting text by another affine with key (p, q). Can the two affine ciphers be replaced with a single affine cipher with key (r, s)? If not, why not? If so, then how would we express r and s in terms of m, n, p, q?

Unit 31
Brute-force attack on the affine cipher

The brute-force attack on the affine cipher tries all possible keys and chooses the deciphered plaintext with the best textual fitness.

Programming tasks

1. Implement the attack. Use tetragram fitness. Be sure to try only valid keys.

Exercises

1. Break this ciphertext:

```
HAGDGKDJKQNDFICEIVZOSABBNWIJIDIBWIDIBWSTITIDWTABDJKPDNIDGDI
JDWZTJSYDNKGDJSPKCPSJDIFSIJZDNKGDKVOGNKPDNWYIDWUIGIYKQNDOGI
KBKVYIVDNWGEKPPWJFJIXWIVZGAJWTKXWPIGGWVQWJGGWDGIKBDNIDZIOTS
JIDNJWWNSAJDSAJIDNJWWNSAJDSAJDNWUWIDNWJGDIJDWZQWDDKVQJSAQND
NWDKVOGNKPUIGDSGGWZKTVSDTSJDNWCSAJIQWSTDNWTWIJBWGGCJWUDNWYK
VVSUUSABZFWBSGDDNWYKVVSUUSABZFWBSGDDNWGNKPGWDQJSAVZSVDNWGNS
JWSTDNKGAVCNIJDWZZWGWJDKGBWUKDNQKBBKQIVDNWGEKPPWJDSSDNWYKBB
KSVIKJWIVZNKGUKTWDNWYSXKWGDIJDNWPJSTWGGSJIVZYIJOIVVNWJWSVQK
BBKQIVGKGBW
```

Unit 32
Attacking the affine cipher with cribs

If we can match two letters from the ciphertext with two letters from the plaintext, then it is possible to solve for the multiplier and shift of the affine cipher. If all of the letters in a crib can be found from a section of the ciphertext with the same multiplier and shift, then we have found a candidate key for the cipher. This resembles the attack on the Caesar cipher, but whereas the Caesar cipher has only one parameter, the affine cipher has two.

Let's run through a short example to see how the algebra works. Suppose our crib is `CRIB`, and that we are matching it to the sequence `KFIT` in the ciphertext. First, we replace the letters with their numbers, where we begin the alphabet at 'A' = 0.

`C`	→	2	`K`	→	10
`R`	→	17	`F`	→	5
`I`	→	8	`I`	→	8
`B`	→	1	`T`	→	19

These give us four equations involving the parameters a and b of the affine cipher's key:

$$\begin{aligned} 2a + b &= 10 \mod 26 \\ 17a + b &= 5 \mod 26 \\ 8a + b &= 8 \mod 26 \\ a + b &= 19 \mod 26 \end{aligned}$$

Suppose we subtract the first equation from the second to get

$$15a = -5 = 21 \mod 26$$

The multiplicative inverse of 15 modulo 26 is 7, so we multiply both sides by 7:

$$\begin{aligned} 7 \cdot 15a &= 7 \cdot 21 \mod 26 \\ a &= 17 \end{aligned}$$

Then, from the last of our four original equations, $b = 2$. If we encipher `CRIB` with this key, we do indeed get `KFIT`, so we have a good candidate for the cipher's key. Finally, notice that if we had subtracted the first equation from the third, we would have found

$$6a = -2 = 24 \mod 26$$

but we cannot remove the coefficient of a because 6 is not invertible modulo 26.

Programming tasks

1. Write a function or script that attacks a ciphertext encrypted with an affine cipher by using a crib. Be careful that you only try to remove coefficients that are invertible.

Exercises

1. Try your program on this ciphertext. A good crib is `CRIB`.

```
OYFSTGLYYRSBXPTCLLIRSBSLZANYSGYNXXPFYXTONWRTVYRAYLDJRYLQOWL
BLSOCLSLTTFSQIYLVRTRNSNGFSLUICNTRELNYQSFSVLQRTINTFCOLVWSRVR
FSRGRGFRCLQWLZNJCQZFHLJIZNJCQSOBNYRBWOAFVHONTCLLIFSQNGOLSIY
NOLTOLQCNJQCP
```

Unit 33
Attacking the affine cipher with monogram frequencies

This attack is very fast, but requires a ciphertext that is lengthy enough to do the required statistics. It is also a necessary attack if we are confronted with a ciphertext that was encrypted with an affine cipher and a transposition cipher (which changes positions of characters so that tetragram fitness is useless), since it can break the affine cipher so that we can then break the transposition cipher.

The method of attack begins by tabulating the monogram frequencies in the ciphertext. They are then shuffled in the same way that the key alphabet is generated in the affine cipher for given multiplier and shift. The multiplier and shift that give the best match to English frequencies is taken as the cipher's key. Armed with the key, we can then decipher the text. The shuffling of the frequencies can be done by replacing the nth entry in the table with the $(an + b)^{th}$ entry (modulo 26), where a and b are the multiplier and shift of the affine cipher.

Below is a graph of the cosine values used to compare our set of monogram frequencies from Unit 5 with shuffled versions of them, for the 312 possible affine keys. Only one, the matching set, has a cosine value above 0.8. It appears that 0.8, or to be safe, 0.85, is a good threshold for determining that we have found the correct key.

Programming tasks

1. Implement the attack.

Exercises

1. Use your implementation to break this ciphertext:

```
ZTYWFFAJYKAXTYLASNYLIKQJSZLWAJYRWJRZTYLYFQLYYWSIZQDLYWOUAZT
SZWZASZAKSYNYJUAZTWZYBZZTASSTQLZ
```

2. The following ciphertext was encrypted with an affine cipher and a transposition cipher in which each block of three letters has been reversed. Note that the transposition makes it impossible for the brute-force or crib-based attacks to succeed. Use your implementation of the attack based on frequencies to break the affine cipher, then reverse each three-letter block to reveal the plaintext.

```
MNRVVCMPWHWUZMNJIWKZMPYUZRIPWCCDMRDPZMNYVMMMZKICLYRCMZMSOKM
ZMAYMHBWRUCEPVWDZMVRPMCXHRPWRBM
```

Unit 34
Keyword substitution cipher

The *keyword substitution cipher*, also called the *keyed substitution cipher*, or simply the *keyword cipher*, is a monoalphabetic substitution cipher in which the key alphabet is constructed from a keyword. The keyword is placed at the beginning of the key, its repeated letters are removed, and then the remainder of English letters are added to the key. The three most common ways of filling the key are these:

- Add the remaining letters in alphabetical order. For example, if the keyword is `AUTOMOBILE`, then the key alphabet is (remember to drop the repeated `O`)

  `AUTOMBILECDFGHJKNPQRSVWXYZ`

- Start adding letters from the alphabetically next after the last letter of the keyword. For the keyword AUTOMOBILE, we start with the next letter after `E`, which is `F`. When we reach `Z` we place the remaining missing letters `C` and `D`.

  `AUTOMBILEFGHJKNPQRSVWXYZCD`

- Start adding letters from the next letter after the alphabetically last letter of the keyword. The keyword `AUTOMOBILE` has `U` as its alphabetically last letter. So we fill in starting with `V`.

  `AUTOMBILEVWXYZCDFGHJKNPQRS`

Here are some other variations. They can be used alone or in combinations.

- Fill in the remaining letters after the keyword in reverse alphabetical order.
- Put the keyword at the end of the key.
- Use a keyword for the plaintext alphabet rather than the ciphertext alphabet.
- Use keywords for both the plaintext and ciphertext alphabets.

Reading and references

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, pages 103-104.

Programming tasks

1. Create a function that returns a permutation of the alphabet from a keyword. Use an optional parameter to determine whether the remaining letters are added in order starting from the beginning of the alphabet or from the last letter of the keyword. The return type of this function should be the data type that you defined in Task 1 of Unit 18.

2. Implement the cipher. Allow for the user to choose a key-filling method.

3. Write a function to generate an alphabet key if the keyword is used in the plaintext alphabet rather than the ciphertext alphabet.

4. Write a function to generate an alphabet key if keywords are used in both the plaintext and ciphertext alphabets. Note that you can combine tasks 1, 3, and 4 into one function if you like. Make a new version of your cipher that can handle keywords for both alphabets.

Exercises

1. Encipher this text with the keyword `KNIGHTS`. Use the first key-filling method that we discussed above.

   Before Cai was born, Cynyr made the prophesy that his son would have a frozen heart and be extremely stubborn. He added the prediction that no one would be able to endure fire or water as well as his son.

2. Decipher this text with the keyword `ROUNDTABLE`. Use the second key-filling method that we discussed above.

```
AVLIDWDPDAVLIDWDPDXBLSBDPQBRGGLAJHVQSLDINVPDQJVSSDPGZRGJIDS
JNPRAHZGLTDJVSBRVISDNDWDPHJPDOZKRGDRUUVQLIAQKDUSPDQJTNDRNFI
LABSQLURIIJSGDRWDSBDDKLSZHZNDQKRLPAXLIDWDPD
```

3. This ciphertext is an example from Gaines's book. It uses a keyword for both the plaintext and ciphertext alphabets. In the plaintext, `J` is used as a space between words. Break it by hand and reconstruct both alphabets to obtain the two keywords.

```
ROVLL ABTLD LBCQM PXLBA FBTCT ATCOR LTOLC RHPDT XLYOA
ELBXP HLXBT XXQLD RGLTK XRLGD BKLDP PLOHL YOAEL KOMXB
LHOEL VCRRC RJLTK DTLRC INXPL LLTKX LRCIN XPLVD BLVOR
LPORJ LDJOL FYLIO PORXP LMDEN XELKC TTLVK OLOHH XEXGL
TOLIO QMEOQ CBXLH OELTV OLIXR TBLBC RIXLK XLVDB LDFPX
LTOLB XRGLT KXLBO PATCO RLFYL EXTAE RLQDC PLLBT CPPLC
TLVOA PGLFX LVOET KLDRO TKXEL RCINX PLTOL HCRGL OATLT
KXLNX YLLTK CBLQA BTLFX LTKXL XWMPD RDTCO RLOHL TKXLE
XHXEX RIXLT OLDLI EOORX ELDRG LTKXL XQMKD BCBLO RLDLG
DTXLL MLBLT KXLTV OLIXR TBLKD BLROT LYXTL FXXRL MDCGL
```

Unit 35
Dictionary attack on the keyword substitution cipher

A *dictionary attack* is an attack in which one tries to decrypt a ciphertext by using a list of possible keywords. We typically terminate the search when we find a plaintext with a high enough fitness.

Python tips

Remember that to split a text and store the pieces in an array, we use the `split()` function. To read a list of words that are separated by newline characters (`\n`), you can do something like this:

```
words = open("dictionary.txt","r").read().split("\n")
```

If you are working on a Mac or Windows computer, then the ends of lines might be `\n\r` or `\r\n` instead of simply `\n`. You should experiment to determine what is the correct thing to put inside the `split()` function. In addition, to avoid an empty word at the end of the list, you might try keeping only up to the second to last entry, which in Python is denoted as the -1st entry:

```
words = open("dictionary.txt","r").read().split("\n")[:-1]
```

Programming tasks

1. Implement a dictionary attack on a substitution cipher whose key was generated with a keyword. Remember that there are several ways to fill the key. Use the word lists that you compiled earlier. Optionally, allow for the use of a custom word list. Use tetragram fitness of the plaintext to determine if and when you have found the correct keyword.

Exercises

1. Perform a dictionary attack and break this ciphertext. The keyword is a common English word.

```
UHENTWVVENLCMCAONTAIOWTNETTYBTLIGHUEVWPFSOMYIUHINBNVITBYINB
XIXIVYBCHOYUHEYHOLEWNIXESTEYBTMBVEWPOFPBSUIALETOFMBUESIBLYH
IAHNOMBUUESHOYVWLLBNVLIFELETTUHECMIGHUTEEMYESENEXESUHELETTF
ILLEVYIUHUHITINUENTEBNVXIUBLREBWUC
```

Unit 36
Hill-climbing attack on monoalphabetic substitution ciphers

This unit describes an attack on the monoalphabetic substitution cipher from Jakobsen's paper in 1995. It is called *stochastic* because it makes random choices as it goes along, and is *hill-climbing* because it works to maximize some function. The function that it maximizes is textual fitness, and for us that means the tetragram fitness that we defined earlier.

The algorithm begins by choosing a key alphabet as the "parent" key and a plaintext is found by deciphering the ciphertext with it. From the parent, a "child" key is obtained by swapping two randomly chosen letters in the parent. The ciphertext is deciphered with the child key, and if the fitness of the resulting plaintext is higher than the parent's plaintext, then the child becomes the new parent, and the process repeats. This continues until the fitness does not improve for a few thousand trials.

Here is the algorithm, so you can't complain that something was unclear:

1. choose a parent key, such as `ABCDEFGHIJKLMNOPQRSTUVWXYZ`
2. decipher the ciphertext with the parent key to obtain the parent's plaintext
3. calculate the fitness of the parent's plaintext
4. set counter to 0
5. while the counter is less than a limit of around 10,000
    a. set the child key as a copy of the parent key
    b. randomly choose two distinct numbers x and y in 0, ..., 25
    c. swap the xth and y^{th} letters in the child key
    d. decipher the ciphertext with the child key
    e. calculate the fitness of the new plaintext
    f. if the fitness of the new plaintext is larger than the fitness of the parent's plaintext
        i. set the parent key as a copy of the child key
        ii. set the parent's plaintext as a copy of the child's plaintext
        iii. set the parent's fitness equal to the child's fitness
        iv. set the counter back to 0
    g. add 1 to the counter
6. output the parent key and/or the parent's plaintext

Python tips

While you cannot modify characters in a string, you can modify items in a list.

Bad:

```
myString = "abcdef"
myString[3] = "z"
```

Good:

```
myArray = ["a", "b", "c", "d", "e", "f"]
myArray[3] = "z"
```

Reading and references

Thomas Jakobsen, “A fast method for cryptanalysis of substitution ciphers,” *Cryptologia* 19:3 (1995) 265-274. DOI: 10.1080/0161-11959188394

Programming tasks

1. Implement the attack.

2. The “cryptoquotes” in the newspaper are enciphered with monoalphabetic substitutions, and spaces between words are preserved. Make a modified copy of the attack that breaks cryptoquotes by using a table of tetragram frequencies that includes spaces and making the required changes to your fitness function and decryptor. Remember that spaces are *not* enciphered.

Exercises

1. Break this ciphertext:

```
IDSIYUDHJZXIXTOQOXUSVOROMNSRMOREXOESGOMMSVOMNSRSUSDHJSYSTNI
JOUQSTSMKSREMNUDJTNIISRJNIDOUKQLHMNGUXLUINIXINHRGNCDSTIDSUN
QCUHRUZSOIXTSVOMNSRSUSNRHRSSCNUHVSIDSUSGTSIQSUUOESIHMVYNSJS
TUIHJOIGDZXIXTOQOIDXTUVOKUOIISR
```

2. Try to break some cryptoquotes from a newspaper.

Part III
Polyalphabetic substitution

Unit 37
Periodic polyalphabetic substitution ciphers

A *periodic polyalphabetic substitution cipher* is cipher in which there is a set of key alphabets which are used in cyclic order as a text is enciphered or deciphered. The number of key alphabets is the *period*.

Let's run through an example. Suppose we want to encipher the message

```
SECRET MEETING TONIGHT PREPARE THE VEGAN PIZZAS
```

with these five randomly generated key alphabets, which are written under the plaintext alphabet for convenience in our *key table*:

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	GAEUPDOXKYTZJWIMBQVHRCSNLF
1	FXBNKVWQAJLECTMHPOSGIRUYZD
2	ZWCPVIHLFXOJEYNTGRDMKUQABS
3	LNVJIEPMADCQZOTYGXWKBSUHFR
4	SUNXTEKWQZLVRIDJACGFBPOYHM

To make things more clear, we can label the characters of the message by which key alphabet we will use to encipher each. The first letter S is enciphered with the first alphabet to V. The second letter E in enciphered with the second alphabet to K. The third letter becomes C, the fourth letter becomes X, the fifth letter becomes T, and the sixth letter is enciphered with the first alphabet to an H. We cycle through the five key alphabets until we reach the end of the message.

```
SECRET MEETING TONIGHT PREPARE THE VEGAN PIZZAS
012340 1234012 3401234 0123401 234 01234 012340
VKCXTH CVIFKTH KDWAHMF MOVYSQK MMT CKHLI MASRSV
```

Python tips

In Python, an array (list) can contain just about anything, including strings or other arrays. While list elements can be changed, letters in a string cannot.

Reading and references

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; chapters XII-XV and XVIII.

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, chapter 4 and pages 236-239.

Auguste Kerckhoffs, "La cryptographie militaire," *Journal des sciences militaires* IX (1883) 5-39 and 161-191, www.petitcolas.net/kerckhoffs/crypto_militaire_1_b.pdf, www.petitcolas.net/kerckhoffs/crypto_militaire_2.pdf, part III.

Programming tasks

1. Implement a cipher in which the key is a set of key alphabets. Use your functions for monoalphabetic substitution if you wish.

2. Write a function that takes a plaintext/ciphertext pair and a period, and returns a list of alphabet keys. If there is an inconsistency (i.e., it tries to fill one place with two different letters), then it should return `False`. If a place in an alphabet is left unfilled, use a dummy character, such as a period.

Exercises

1. Take these three key alphabets and encipher the following text.

```
LBRUVCJAWZYSHXINOQEPFTGKDM
SLNAXDIGOBKCEYQHTMWJFUVPZR
IFWVBXNGKHZQYOELPCDTJRUSAM
```

   In his book published in eighteen sixty-three, Kasiski presented a method for finding the period of a polyal-phabetic substitution cipher. The method uses the positions of repeated sequences of letters in the ciphertext.

2. Decipher the following ciphertext with the same key as in the previous exercise.

```
IYBIDTAXYIWTNQLFCIQHESZISHGLLBPOWROLAXCGSDPGIPQXBCIWWBUXRWI
BXXCVOTGSDCOCEJLFLQWWGVAKXDKCJBVYBWIGPZXWUBBFTGSDRQOEOVVMBU
FOBMBLKIBCBFYTWCBWIGPXBXWKKJAPGCVX
```

Unit 38
Finding the period: Kasiski method

The *Kasiski examination* is a method for finding the period of a periodic substitution cipher. It involves finding repeated sequences of letters in the plaintext. When more than one repeated sequence can be found, the period is likely to be a common factor (possible the gcd) of the distances between them.

Let's look at an example. Consider this ciphertext:

```
THZBAROLASYZFKHFNYCEYXOQMWHXLELXLAUHNPMIAZTLVDWNNHRDOW
SIHUCCMGNTTTCWSIHUCCMHTEEDCBUGMHZBAROLTSONNSHUDWQFZXRP
NABMHTZDPRYHUCMMNTWADUBUKAOCCMUKELRSDREHULXIAYPECDPNZR
OFVTRTWOCCMUKLAWGILYHNLCBRGWYNYCEYXTLVSGUFIDDMEKW
```

Notice that the sequence `ZBAROL` occurs twice. The distance from the `Z` of the first occurence to the `Z` of the second is 84 letters. The sequence `SIHUCCM` also occurs twice, with a distance of 14 letters. The sequence `OCCMUK` occures twice, with a distance of 35 letters. The greatest common divisior of 84, 14, and 35 is 7, so the period is likely to be seven.

Reading and references

Friedrich Kasiski, *Die Geheimschriften und die Dechiffrir-Kunst*, 1863; digital.onb.ac.at/OnbViewer/viewer.faces?doc=ABO_+Z224431001

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; chapter XIV.

Fletcher Pratt, *Secret and Urgent: The Story of Codes and Ciphers*, New York: Bobbs-Merrill, 1939, chapter IX, section II.

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, pages 207-210.

Programming tasks

1. If you think that it will be possible to write a function that employs this method, go for it.

Exercises

1. If you wrote a script for this method, test it with the example ciphertext.

Unit 39
Finding the period: Sinkov method

A ciphertext that has a period *m* is like mixing *m* different ciphertexts, each enciphered with a different key. As a result, the chance of picking two characters randomly and obtaining identical letters is reduced, as compared to an unencrypted English text. Without proving it, we state that the measured index of coincidence of the entire ciphertext is related to the period in this way:

$$\mathrm{IoC} = \mathrm{IoC}_{\mathrm{random}} + (\mathrm{IoC}_{\mathrm{English}} - \mathrm{IoC}_{\mathrm{random}}) / m$$

Solving for the period and using $\mathrm{IoC}_{\mathrm{random}} = 1$ and $\mathrm{IoC}_{\mathrm{English}} = 1.75$ (in the normalization that we use), we obtain

$$m = \frac{0.75}{\mathrm{IoC} - 1}$$

In tabular form:

IoC	period
1.75	1 (monoalphabetic)
1.38	2
1.25	3
1.19	4
1.15	5
1.13	6

As we can see, as the period increases, the values of the IoC get closer together. Since these numbers are approximate and there is a lot of variability in the IoC, *we should not rely on this method for finding the period*.

A better way to use the IoC to find the period is to guess a value for the period *m* and then slice the ciphertext into *m* slices and find the IoC of the slices. The nth slice is composed of every m^{th} character, starting with the nth character. Partitioning the text this way gives us slices that would each be encrypted with one key alphabet, if we guessed the period correctly. The average of the IoCs of the slices serves as a good measure of the IoC of the plaintext that you would obtain by deciphering the ciphertext with the guessed period. If this IoC is not close to the IoC of typical English text, then the guess is a bad one. By guessing periods until the IoC calculated in this way is close to that of English, we can find the period, and we are usually correct.

Let’s look at this ciphertext, for example:

```
BUHMKLRASCKBLRZQQHRZMVVZBZLXWBNHOMKKEBTQWTUEMPLWLBQAGI
UWSFSFVPLHBVPHGXVOHYPMQWSCQAGXEMCHVFWQRJXXRUMLLVFTLTLD
PMPNEIQQPVYYAGLXRBVRRZQCKIOBUHFBAGZEVBBXWBBUHMKLRASCKB
LRZQQHRZMGRJFVQWLBXRUMLLVVXLKHWXEMPLTEMEWIUBVQXLAYLGBA
NQHXDRUEDMGKIFWPRJQPRVPFKRVMCBUHESMEDKBQBFMPKYRWBBBWLX
BBIIKOYLWEBUHRQPRQYJJRUSCAYLGBAVVPFSROCQWOHXEMCHVFWQ
```

And here is a graph of the IoC averaged over the slices, for periods one through ten. The measured IoC is in blue and the theoretical values for a period of five is in pink. The peak at ten is due to the fact that when we take ten slices of the ciphertext, then each slice is half of one of the slices that we took for period five. We conclude that the period of the cipher is five.

But the astute reader will notice that five is a prime number. What happens if the period is not prime? Below is a similar graph for a text with a period of 15. Again, the measured IoC is in blue and the theoretical values are in pink. The secondary peaks at periods 5 and 10 and the tertiary peaks at 3, 6, 9, and 12 are due to the fact that when we take a number of slices that shares a factor with the true period, then each slice contains some letters that were enciphered with the same key alphabet, i.e., that are in the same “true” slice. The expected theoretical value at period *n* when the true period is *m* is

$$\mathrm{IoC}_{\mathrm{theory}} = \mathrm{IoC}_{\mathrm{random}} + (\mathrm{IoC}_{\mathrm{English}} - \mathrm{IoC}_{\mathrm{random}}) \cdot \gcd(n, m) / \mathrm{m}$$

Reading and references

William F. Friedman, The Index of Coincidence and Its Applications in Cryptography, Riverbank Laboratories Department of Ciphers Publication 22, Geneva, Illinois, 1920, www.marshallfoundation.org/library/methods-solution-ciphers

William F. Friedman and Lambros D. Callimahos, Military cryptanalytics, Part I, Volume 2, Aegean Park Press, 1956, reprinted 1985.

M. Mountjoy (1963) The bar statistics, NSA Technical Journal VII (2, 4).

Abraham Sinkov, *Elementary Cryptanalysis: A Mathematical Approach*, 2nd edition, revised by Todd Feil, published by Mathematical Association of America, 2009; www.jstor.org/stable/10.4169/j.ctt19b9krf; section 3.3.

Practical Cryptography: practicalcryptography.com/cryptanalysis/stochastic-searching/cryptanalysis-vigenere-cipher

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, pages 376-380.

Programming tasks

1. Write a function that cuts a ciphertext into a number of slices as described above. If the number of slices is n, then the first slice contains every n^{th} letter, starting with the first letter of the text; the second slice contains every n^{th} letter, starting with the second letter of the text; etc.

2. Write a function to find the period of a ciphertext with the method described above. You will need to decide on an appropriate cut-off, which you can base on the results of Exercise 1 in Unit 12.

Exercises

1. Use your function to find the period of the example ciphertexts from this unit and the previous unit.

Unit 40
Finding the period: twist method

The *twist method* was published in 2015 by Barr and Simoson. Let's see if we can understand it. First, look at the monogram frequencies of English, sorted in ascending order:

Now let's look at this ciphertext, which was enciphered with period five:

```
ZTSCALWAVEXTWAEJSSCASNSVSGSAUQSALEVSUTQJKDMGOACDCWEPLN
LGKETGJPFVEJIHWVYJEUWWSIVWZTDCHABPQSDDFNMYWTCVGJNFMLIH
GQSCLQSCTDCXSGTJYJHAFPQXTUIWBEFCGJCJSGHECGGADPMYWTVSNX
SKXGJRFISSPNEFTTJIKPIFAZDWJSSGEASMHTCQETRGHSGTJYJIHGQS
CLQSCTDCASNAIEACAMMFSOHWSSNGWKHEGQWSTQGJDSULAHFCGWBYPE
ETIURGIIOTGGTCRLWEUEASHGWWTMGHLDHCZWHOOILWIPKGCHKWEXNF
GGCVGVKPTKSFLAUGDTATPQHOOILWIPKZTFGPLWEFMVCTJENTTQVMHH
CXSGTJYJUENXSLKYEJSIGVQDUUXSGTNIVBEJIKPIFPSBENCLWEOEFA
OQOWSRQYFSTQLABAIEACAPHKAIILLAYTEAHEFLAHEAITGOYWZBMOQZ
TSCMVXSCMVNOWW
```

Here are the monogram frequencies for it:

Notice that it is much flatter than the previous graph. But, now, here is a graph for every fifth letter of the ciphertext:

The slope of this graph resembles that of English. This is the basic idea behind the twist method: If we divide the ciphertext into slices so that each slice has been enciphered with the same key alphabet, then the sorted monogram frequencies of each slice will resemble English.

Now we need to develop this into an algorithm that we can use in a program. Following Barr and Simoson, we define the *signature* of a set of of letters (a text) as the sorted list of its monogram frequencies. The *twist* of two signatures $A = \{A_i\}$ and $B = \{B_i\}$ is defined as

$$A \lozenge B = \sum_{i=0}^{12} (A_i - B_i) + \sum_{i=13}^{25} (B_i - A_i)$$

Notice what this does: it adds up the amount by which A exceeds B in the lower half and the amount by which B exceeds A in the upper half of the graph.

To find the period, we try various periods n. For each trial period, we slice the ciphertext into n slices, where each slice takes every n^{th} letter from the text starting from a different point. For example, with the sample ciphertext above, if we try a period of five, then we assign letters to slices like this:

```
ZTSCALWAVEXTWAEJSSCASNSVSGSAUQSALEVSUTQJKDMGOAC...
0123401234012340123401234012340123401234012340123401...
```

For each slice, we find its signature. Then, we average the signatures and take the twist of English monogram frequencies with the average signature. The trial period for which the twist is the greatest is likely to be the true period of the cipher.

Python tips

Arrays (lists) can be sorted with the `sort()` function, like this:

```
myArray = [1, 3, 2]
myArray.sort()
```

Reading and references

Thomas H. Barr and Andrew J. Simoson, “Twisting the Keyword Length from a Vigenère Cipher,” *Cryptologia* 39:4 (2015) 335-341, DOI: 10.1080/01611194.2014.988365

Seongmin Park, Juneyeun Kim, Kookrae Cho, and Dae Hyun Yum, “Finding the key length of a Vigenère cipher: How to improve the twist algorithm,” *Cryptologia* 44:3 (2020) 197-204, DOI: 10.1080/01611194.2019.1657202

Programming tasks

1. Write a function to find the signature of a piece of text.

2. Write a function to find the twist between two signatures.

3. Write a function to find the period from a ciphertext using the twist method. Feel free to use the function that you wrote for slicing a text in the previous unit. You might want to write a separate function for averaging signatures.

Exercises

1. Use your function to find the period of the example ciphertexts in Units 38 and 39 and in this unit.

Unit 41
Fuer GOD cipher

We now turn to a periodic polyalphabetic substitution cipher that was used in the real world. The *Fuer GOD cipher* was used by the Germans to communicate to forces in the Mediterranean Sea during World War I. Its name comes from the fact that its messages were addressed to radio receiving station with call letters GOD ("für GOD" means "for GOD"). Cryptanalysts in Europe and America also called it the *Welhelm cipher*. Unfortunately, we do not have any primary sources about this cipher and have to rely on what cryptanalysts were able to learn about it from intercepted messages.

We are studying this cipher not only for its own sake, but also because it serves as a model for other polyalphabetic cipher for which we (mostly) know the tableau of alphabetic permutations that it employs. In the next few units we will look at attacks that can be specialized for other such ciphers. Later we will see ciphers whose tableaux we do not know.

Sender and receiver both have a copy of the same tableau of twenty-two alphabetic permutations, each of which is labeled with a letter. This allows them to use keys that are words or phrases rather than a full list of scrambled alphabets. Each letter in the keyword indicates a permutation in the tableau, in the same way that the numbers did in Unit 37. The tableau, to the extent that we believe we know it, appears on the next page. As far as we know, there are no permutations for `J`, `Q`, `X`, and `Y`. The tableau contains holes in rows `F` and `W` where the literature about the cipher has inconsistencies. Table 41.2 shows the tableau with educated guesses to fill the holes.

The Germans used the cipher less securely than they could have. Rather than use a new keyword for each message, they cycled through a list of thirty words. They are listed in Table 41.3.

Here is the requisite example of encryption with keyword `ITALIAN`:

```
THE VEGAN PIZZAS ARE PREPARED IN PISA MEET THERE AT SIX
ITA LIANI TALIAN ITA LIANITAL IA NITA LIAN ITALI AN ITA
NRA NYJRH YGLPEJ XBA JMAGXBAW BN GBAE BYAH NRAIY EH OPV
```

key letter	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
A	ECDBAFJIGHLKONMSPQTRZUXVWY
B	BADCFGEIHJNOKMLSRPTQWXVYUZ
C	CDABGHEJFIKMPOLNTRQSXUZVWY
D	BACHDJFEGILONPKMSQRUZTYVWX
E	ACDBHJFIGEMNLKOTRSPQYZVUXW
F	ZVXABCEFD JIGK PLNSRO UT
G	VYXUZWCABEDIHGFLKNMJQOTPSR
H	UXZWYVAEBCFDIHGJNKMLSPORTQ
I	XUVZYWACBEDGIHJFKMONLTRQSP
J	
K	SQRYVXUZTWBDCAEJHKIFGPMONL
L	RTSWVYZUXFACBEDJKIGHONMPQL
M	TRSQYWXZVUEBACDKFJIGHMLNPO
N	RQPSZWTVUXYDBCAGIEJHKFONLM
O	LOPNMQSRTUVZXYWCABHEDGJFKI
P	MOLNPSRQXTYWZUVADCBHFIKEJG
Q	
R	PONMRTSQWYUXZVCABEDFJGKHIL
S	LMONTQRPSZXUYVWBACDEGJHFKI
T	MOKNLQSRPWZTVUXYDBACEFJGIH
U	HFIGNMJKOLQPSRVTZUWXYBEDCA
V	EDIGHFLMKPONRQJSUXTZWVYCAB
W	IEHFGLOMJKNQPTRSX YUZ BADC
X	
Y	
Z	IFHJGNKLMPOTSRQVYUXZWDBCAE

Table 41.1: Tableau for the Fuer GOD (Wilhelm) cipher. The tableau contains the best of our knowledge when this chapter was written. There is no information about rows J, Q, X, and Y, and no reason to believe that they were ever used, or even existed. Plaintext letters appear at the top of the tableau, and key letters along the left edge. Ciphertext letters are in the bulk of the tableau.To encipher the letter E *when the key letter is* S*, for example, we look to the place where the column headed by the plaintext letter* E *intersects the row labeled by* S *to find the ciphertext letter* T.

key letter	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
A	ECDBAFJIGHLKONMSPQTRZUXVWY
B	BADCFGEIHJNOKMLSRPTQWXVYUZ
C	CDABGHEJFIKMPOLNTRQSXUZVWY
D	BACHDJFEGILONPKMSQRUZTYVWX
E	ACDBHJFIGEMNLKOTRSPQYZVUXW
F	WYZVXABCEFDHJIGKMPLNSROQUT
G	VYXUZWCABEDIHGFLKNMJQOTPSR
H	UXZWYVAEBCFDIHGJNKMLSPORTQ
I	XUVZYWACBEDGIHJFKMONLTRQSP
J	
K	SQRYVXUZTWBDCAEJHKIFGPMONL
L	RTSWVYZUXFACBEDJKIGHONMPQL
M	TRSQYWXZVUEBACDKFJIGHMLNPO
N	RQPSZWTVUXYDBCAGIEJHKFONLM
O	LOPNMQSRTUVZXYWCABHEDGJFKI
P	MOLNPSRQXTYWZUVADCBHFIKEJG
Q	
R	PONMRTSQWYUXZVCABEDFJGKHIL
S	LMONTQRPSZXUYVWBACDEGJHFKI
T	MOKNLQSRPWZTVUXYDBACEFJGIH
U	HFIGNMJKOLQPSRVTZUWXYBEDCA
V	EDIGHFLMKPONRQJSUXTZWVYCAB
W	IEHFGLOMJKNQPTRSXVYUZWBADC
X	
Y	
Z	IFHJGNKLMPOTSRQVYUXZWDBCAE

Table 41.2: Tableau for the Fuer GOD (Wilhelm) cipher, with our best guesses to fill the holes in Table 41.1.

1	SPRINGBRUNNEN
2	VOLLWICHTIGES
3	PROBIERSTEIN
4	MARIONETTENSPIELER
5	BESTIMMUNGSORT
6	FAHRTUNTERBRECHUNG
7	WAGENWECHSEL
8	WOLLENSIEFAHREN
9	NOCHETWASMILCH
10	ESSCHMECKTMIR
11	KORKZIEHER
12	ZUCKERZANGE
13	SCHWIEGERVATER
14	WIELANGEDAUERTES
15	WIEVIELKOSTETES
16	FREMDENBUCH
17	BENEDIKTINER
18	SELTERWASSER
19	BEIWEMKAUFENSIE
20	PAPIERHANDLUNG
21	PFERDEVERMIETER
22	GOLDARBEITER
23	HANDLUNGSGEHILFE
24	HANDSCHUHMACHER
25	INSTRUMENTENMACHER
26	KAMELTREIBER
27	RADIERGUMMI
28	BESORGEDIEPFERDE
29	DUNKELKAMMER
30	RECHTSGELEHRTER

Table 41.3: Keys used by the Germans with the Fuer GOD (Wilhelm) cipher. Keys were used in sequence.

Reading and references

J. Rives Childs; German Military Ciphers from February to November, 1918; 1918; https://www.nsa.gov/Portals/75/documents/news-features/declassified-documents/friedman-documents/publications/FOLDER_268/41784789082381.pdf

> The tableau in this paper has duplicate letters in some rows, and this should not be allowed, since the cipher needs to decrypt texts unambiguously.

Some typed manuscripts by J. Rives Childs in a special collection at the University of Virginia library.

> To be honest, we have not read these documents. Perhaps it contains more complete and consistent information about the Fuer GOD tableau.

Greg Mellen, "Cryptanalyst's Corner," *Cryptologia* 8:1 (1984) 55-57, DOI: 10.1080/0161-118491858773

> This paper contains a ciphertext from World War I as "Problem No. 11." This is the same ciphertext solved in the following paper.

Hans van der Meer, An old challenge, 2010, https://staff.fnwi.uva.nl/h.vandermeer/pubs/fuergod.pdf

> Note that this paper has some typographical errors. The block `GBVRT` of the ciphertext should be `GBRVT`. Some software that prepared the PDF of the document removed the German letters `Ä`, `Ö`, and `Ü` from the plaintext. Also there are a few misspellings, and posslibly other typos in the ciphertext that were inherited from the paper in *Cryptologia*. The plaintext should read:
>
> > ganz zuverlässiger vertrauensmann meldet unterm dritten august: italienischer dampfer sumatra soll demnächst rest des dritten erythraeischen bataillons und das sechszehnte bataillon und urlauber von lybien nach port said überführen sowie von port said das erste elfte und zwölfte bataillon nach lybien bringen. das 16. batl. ist augenblicklich in benghasi. es soll durch das 12. ersetzt werden. die erythraeischen bataillone in lybien sind sehr geschwächt, das in lybien befindliche 15. batl. zählt nur noch 250 mann. zwischen lybien und port said verkehrt alle zehn tage ein postdampfer. der auch für truppen transporte benutzt wird. pol. 30 07 4
>
> (The Germans used the substitutions `Ä→AE`, `Ö→OE`, `Ü→UE`, and `ß→SS` before enciphering with ciphers that did not use those letters.) The tableau in this paper also has duplicate letters in some rows, and this cannot be correct for a cipher that deciphers texts unambiguously.

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Macmillan, 1967, pages 310-311 and 337, archive.org/details/B-001-001-264

Programming tasks

1. Implement the Fuer GOD cipher. Allow for the possibility that the key is given as a keyword or as an integer indicating one of the standard historical keys. Use the tableau in Table 41.2. Do not allow keys to have letters `J`, `Q`, `X`, `Y`.

Exercises

1. Encipher with key `BARONESS`:

   > The paintings on the walls recorded that the priest had been a good and pious man who had offered sacrifices to the gods, and who, I concluded, would be above leading his fellow-men astray.
   >
   > (from *By the Gods Beloved* by Baroness Orczy)

2. Decipher with key `MARTIAN`:

```
TNMPHRVYTPVYXRPXPANIZWMXPXJZTTSMSECQUPBBATTRRNXTHZADCYOJLIW
TYRVYCXXJOJGIPCVKKIRRBYBHZGNZLSACRQLIORPNCCUASYTNBBCZQGVMHW
ZTQFRGWHDNSEYECQGVNYASAGSRNDVTKXLHJZGIRTXNTHESLJFHZASXZT
```

Unit 42
Brute-force attack on the Fuer GOD cipher

To perform a brute-force attack on a periodic polyalphabetic cipher whose keys are strings of letters, we need to try all one-letter keys, then all two-letter keys, etc., until we find an acceptable plaintext. A plaintext is considered acceptable if its fitness is high enough; for this attack on these ciphers we use a fitness function based on *n*-grams, such as tetragrams, and not monograms.

If the length of the key is longer than three, then this attack is not the best choice, due to the amount of time it would need. Nevertheless, implementing it is a learning experience.

Python tips

The `product()` function from the `itertools` module can generate all possible tuples containing items from a set. For example, this block of code will create a list of all possible three-letter combinations:

```
from itertools import product
letters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
combos = list(product(letters,repeat=3))
```

The length of the combinations (3, in the example) can be replaced with a variable.

Programming tasks

1. Implement the attack. Use tetragram fitness to decide when you have found the correct plaintext. Exit then.

Exercises

1. Break this ciphertext. The plaintext is in English.

```
EMBRGZQKBNCNZNZPEVHWCGHADYZTMNNLDDHBURBROQRTGDHMUWBPAQGBZYR
ZURZQKFZLNDTGURZQTCEEFQTGURZYBFPSGDGZQOFURZQHVREFYRZURZQKFZ
LNDSFGYCZCFQNFYYQMWGOTWDHIBNUDBGDGZQLMLHTEMJEMNWWQBBZQTXEWN
MWFQHBOZSKBTGHZRTGPMNKBMBTGU
```

Unit 43
Attacking the Fuer GOD cipher with cribs

We have seen crib-based attacks before, but polyalphabetic ciphers add some complications. There are three-ish overlapping and interrelated tasks of such an attack, and each of them is affected by the nature of periodic polyalphabetic ciphers:

0. finding the period of the cipher
1. placement of the crib against the ciphertext
2. extracting information about the key from the crib and the matched segment of ciphertext
3. expanding the information that we have about the key and/or plaintext

Finding the period

We have studied this already, and recommend the Sinkov method.

Placement of the crib

We try placing the crib at every possible position against the ciphertext and checking for consistency. In other words, we check to see that for each key alphabet there is only one letter for each of the twenty-six slots and that no letter is duplicated. We can also check that a partial key matches any row of the cipher's tableau. For other ciphers there can be more or different constraints. There will often be several possible placements of the crib that are consistent and have to be tried. Ultimately, textual fitness will determine success.

Extracting information

This step overlaps the previous one. By comparing the crib to the corresponding segment of the ciphertext, we find some of the letters in the key table. Using those letters, we can also find the plaintext letters for parts of the ciphertext outside the segment we used.

Expanding the information

There are two ways to increase what we know about the keys and plaintext. One is to look for probable words in the text where some letters are known. If we are correct, then we can add more letters to the key table. The second way is to use the structure of the cipher to deduce more information about the

keys from what we have. With the Fuer GOD cipher, since we know the tableau, once we know one or two letters in a key alphabet, then we can fill in the rest of that row in the key table.

To summarize: The placement of a crib can fail for any of these reasons:

- the key table gets two of the same letter in different positions on the same row
- we try to put two different letters into the same position in the same row of the key table
- a row of the key table is not consistent with any of the rows of the tableau
- the resulting decrypted plaintext does not resemble text in a human language

An example would be nice, eh? Consider this ciphertext:

```
JROQLDBCVZTQRSGMXQHHZGESKLOSOTLOHEIFZOFOYJLENIFTHQFFBPHOHFC
FDUSFDNKBGQLZBDIRHMKXTMDQSQYELDBHRYCHEFJRMTFCJCXRBUGTRCTHCV
BSQJWEUIGNJWOWQCUYIHGPJPMFLYQJPNBTMHMXXTBEOSQYPMDCXFNIFUOMX
SBMWGUIXMCHIGNXMCCHCVZQIVDRWUBMWKLKOIFUABDYQIVSHPVFBZAFVOQQ
UHPPGOBEBFFRLWCBOFUFDUSFDNKBGASPGOIVQJCLEIHVFFCLWHQJRI
```

Let’s see if we can break it with this crib:

```
IVYANDTHECAN
```

The zeroeth step is to find that the period is seven:

So we will build a key table with seven alphabets:

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	.....
1	.....
2	.....
3	.....
4	.....
5	.....
6	.....

Next, we must find the correct placement of the crib. We will show two positions that fail, and then move on to the positions that give consistent key tables, one of which fails in comparison to the tableau and one of which succeeds. Start at position zero, i.e., at the beginning of the ciphertext. Is this placement possible?

```
0123456012345601234560123 45...
IVYANDTHECAN
JROQLDBCVZTQRSGMXQHHZGESKLO...
```

If we fill in the key table, we get to the following state. But at position 10, we need to put a `T` under the plaintext `A` in row 3, yet that spot is already occupied by `Q`. Therefore we must reject this placement.

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	.....CJ.....
1	....V.....R....
2	..Z.....O.
3	Q.....
4	.....L.....
5	...D.....
6	.....B.....

We get a similar inconsistency until we reach position 12:

```
...3456012345601234560123 45601...
   IVYANDTHECAN
...RSGMXQHHZGESKLOSOTLOHEIFZOF...
```

We fill in the table until the following state. But then at position 21 we want to put a `G` under the plaintext `C` in row 5. But there is already a `G` under `Y` in row 5. So we must reject this placement of the crib.

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	.....X.....
1	...Q.....
2	.....H.....
3	.....HR.....
4	....Z.....S....
5	.....G.
6	M.....

At position 118 we find a placement that passes the tests of internal consistency of the key table.

```
...60123456012345601234560123 4...
   IVYANDTHECAN
...BSQJWEUIGNJWOWQCUYIHGPJPMFL...
```

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	....G.....S....
1	..N.....Q.
2	J.....
3	.....W.....
4	...E.....
5	.....U.....
6	.....IB.....

The problem now is that there is no row of the Fuer GOD tableau that is consistent with row 0 of our key table. If G comes under E, then S must be under T, P, or M in rows C, W, and Z of Table 41.2. But here it appears under V. So, once again, we reject this placement of the crib. We have better luck at position 181:

```
...60123456012345601234560 1...
   IVYANDTHECAN
...GUIXMCHIGNXMCCHCVZQIVDRW...
```

We get this partial key table:

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	....G.....U....
1	..N.....I.
2	X.....
3	.....M.....
4	...C.....
5	.....H.....
6	.....IG.....

Now comes the the time to expand our knowledge. This will be very easy, since we know the tableau of the Fuer GOD cipher. By comparing the zeroth row of the table above to the tableau, we see that it matches key letter `C`, the first row matches `R`, etc. Row 5 matches tableau rows `L`, `N`, and `P`, so we will have to do more work. Row 6 matches `A` and `E`. The others have a single match each. Suppose we decrypt with key `CRIBB..`; we get (after inserting the missing bits of the crib)

```
HESTO..ANDST..EDATI..ERYMO..NFULI..IMSEL..ORTHE..STPAR..FTH
EC..ISTMA..OMICH..LWASN..THEGO..FEEDI..HERSE..ALWAY..ROVID.
.THOUG..ECOUL..AKEHI..HAREO..HATAS..LLASA..THERN..UTTHE..LL
YAN.IVYANDTHECAND..ANDTH..RIBAN..OWSHE..DSETH..FACEA..INSTT
..MALLA..ITWOU..NTBEC..ISTMA..TALLH..HOUGH..ITHOU..HEM
```

We can see some probable words peeking through, such as `ALWAY.` which looks like `ALWAYS`. If we look at the ciphertext, we see that that `S` was enciphered to `G` in row 5. Now we have

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
5	.....GH.....

This breaks the ambiguity for row 5, since now it can only match row `L` of the tableau. With this improvement, the plaintext has fewer holes:

```
HESTOO.ANDSTA.EDATIT.ERYMOU.NFULIN.IMSELF.ORTHEB.STPART.FTH
ECH.ISTMAS.OMICHA.LWASNO.THEGOO.FEEDIN.HERSEL.ALWAYS.ROVIDE
.THOUGH.ECOULD.AKEHIS.HAREOF.HATASW.LLASAN.THERNO.UTTHEH.LL
YANDIVYANDTHECANDL.ANDTHE.RIBAND.OWSHEH.DSETHE.FACEAG.INSTT
H.MALLAN.ITWOUL.NTBECH.ISTMAS.TALLHE.HOUGHT.ITHOUT.HEM
```

It looks like `.ROVIDE` could be `PROVIDE`. That adds a letter to row 6:

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
6	.....IG.....T.....

This fixes the last letter of the key to be E, and the key is CRIBBLE. We can now completely decrypt the text, which comes from *Candle and Crib* by Katherine Frances Purdon:

> He stood and stared at it, very mournful in himself. For the best part of the Christmas to Michael was not the good feeding Herself always provided, though he could take his share of that, as well as another; no, but the holly and ivy and the Candle and the Crib; and now she had set her face against them all. And it wouldn't be Christmas at all, he thought, without them!

Programming tasks

1. Implement the attack. The function you wrote in Exercise 2 of Unit 37 should be useful in checking for consistency and building the partial key table.

Exercises

Break these ciphertexts. Plaintexts are in English. What are the keywords?

1. Crib: UNMARRIEDGERMAN

```
KXQQRMXXHXWXDNRZXQQCFXOHNQXXFRCWZIRNMKCPVLQLPEMKHMSYXLPVFXH
TRMXHMCHHVQCVEKNCIHVRXGXHNCIRHVMYGAVMAKZROJQBTCMFAWFCFYXAYX
JCEIEKNRJFBJFSYGXXMHDRWTHAXWUHSPXOENPESETAXSKLZRZKAFRGAQCFC
HCCPMGOTALDNWNRJSUDFWLAWHLBRAXXQZRHZ
```

2. Crib: THREEOREVENFOUR

```
AQWAYEXHFTAQBOTSFKIBCEAHHMEUTMKNCWRQMRZOXJQGJOTXCKDKYAKNCWR
HYWGIRARKNAMHOFCZNSYSQMVRBNGIGKPUJAJDCTRUTMKNCWWPCEXAFNDUJX
DAQEFYAGIQ
```

Unit 44
Dictionary attack on the Fuer GOD cipher

We have seen a dictionary attack before. We need to check all keywords from our word list until we find one with an acceptable fitness.

Python tips

This statement will open a text file read-only (`"r"`), read its contents into a string, automatically close the file, split the string at end-of-line characters, and put the pieces into a list:

```
words = open("words.txt","r").read().splitlines()
```

Programming tasks

1. Implement the attack. Allow for the possibility of choosing the file containing a word list. The default file should be the one you generated in Unit 4. If you are clever, and I'm sure you are, then you will take the attack you will write a function that takes the decryptor as one of its arguments, so that you can reuse it with other ciphers.

Exercises

1. This English text was encrypted with one of the thirty German keywords used in World War I. Break it.

```
VRQJDTICXJBLYBKZLIIAXWLQHMEQUQFGVDOMYNZXBUOULQHTXNLOPFRAKJY
SHBQYDWUYQUAFLGXGOZAOTHHWNXKSHTYNNYLCBCESCDVIHIPPHKFOJUQRPQ
USUNHEBFDJCXBAYUNQVTKJSCBFDAPMYBXKBYJEVWCIHPQEVZPKTXQVUVP
```

2. This ciphertext was encrypted with an English keyword. Break it.

```
WOHHNLMWRPTYPXHVPRVYBPVMBOHKAAUNMHAUNIVZJXCXMONLXXKMPMBSMWN
BVVZVKCQNRCOEUGSWOXCEXHIMHOUJVPUGVKYUNNCHILBXMXSJMCHQHQFUUX
QECNLVYTNRVHIMINOPHUZYLQPPZTWPSMINCJOTNIQMGBMUXLMLHCVWBOPGM
UBMHYZONLNVKUQMOPPMJPCNAWAKPMFWNXSFFCKPUJWVFAKHBAAPUXPNOUHK
PKATWVAAOAXXPHILNVVCOFPUHQHQXUPHILCNKMPYMOUHHNMTVHJEWPVSPXZ
NHQGUREQXDRYNAHCXXPXUFRMTAXNIMEMJAUNWPUBPUJFADTVYNBOQKKMHNX
VQPNNPYNCPATBXPML
```

Unit 45
Hill-climbing attack on the Fuer GOD cipher

As usual, the first step is to find the period of the cipher. We recommend the Sinkov method (see Unit 39). Then we take a key that contains all A's. We start with the first letter of the key and replace it with each of the letters of the alphabet. The choice that gives the best textual fitness for the deciphered plaintext is kept. Then, with the new first letter in place, we move to the second letter of the key and do the same to it. We continue until all letters of the key have undergone this process. Then we repeat again from the beginning of the key. We continue to loop through the key in this way until the fitness can no longer be improved. This attack is very reliable, even for shorter ciphertexts.

The algorithm:

1. find the period m
2. set the key as m copies of the letter A
3. set the "current fitness" to the fitness of the undeciphered ciphertext
4. set a flag equal to FALSE (to indicate that we have not finished)
5. while the flag equals FALSE
    a. set the "old fitness" equal to the "current fitness"
    b. for each position i in the key (i from 0 to $m-1$)
        i. set "maximum fitness" equal to "current fitness"
        ii. for each letter x in the alphabet
            - set the ith letter in the key equal to x
            - decipher the text with the key
            - calculate the fitness of the new plaintext
            - if the new fitness is greater than the "maximum fitness"
                · set the "maximum fitness" equal to the new fitness
                · set the "best letter" equal to x
        iii. set the ith letter of the key to the "best letter"
        iv. set the "current fitness" to the "maximum fitness"
    c. if current fitness" equals old fitness" (there has been no improvement)
        i. set the flag equal to TRUE (to indicate that we have finished)
6. output the key and plaintext

Two things to keep in mind: By "alphabet" we mean all allowed key letters; for Fuer GOD there are only twenty-two. With the Fuer GOD cipher, slices of a ciphertext typically have a lower index of coincidence than for many other polyalphabetic ciphers; this is an effect of the fact that most columns of the tableau have repeated letters. You may have to adjust for this when looking for the period.

Reading and references

Practical Cryptography,
practicalcryptography.com/cryptanalysis/stochastic-searching/cryptanalysis-vigenere-cipher-part-2

> This webpage presents the attack (with slight differences) for the Vigenère cipher, which we will see soon. We can generalize the attack to any periodic polyalphabetic substitution cipher whose tableau is known.

Programming tasks

1. Implement the attack. Use tetragram fitness.

Exercises

Break these ciphertexts. Plaintexts are in English. What are the keywords?

1.
```
XZSWUBIVJLCYPXBFLWOABHPNPVHUNRBVVXXFEWWWGUIWWWUQLUCEHBKSQWZ
WDOXYJCUBFRHXKHGQRMFCPUMVJNRXDBJQRBPBQTXTEBBQAPXQMIOURFRPMQ
MCDHVUKKXRYHMOTVVMWTPXFVDHLHBMOZJHOCHWYRLILQJXMHSMLREVOIPLQ
HVIAUJCQZTSLMMMCMCLWCYREAAORMUJNMJWYRUQKQWZNKHUBSLZYPQCCXHE
HOPSTUHGURPYNOGTWABVXAOWINVBQCXXZHEHLUMJQBDRXDLUSGAMVPPVOBH
QMSYVYPCCWYVRVSBVOYY
```

2.
```
GCTKAAPVMKQAYJQRPXNLLUYQVSCHXAPHNMMHYNOHKUOOTBILERZQVYUWRIL
UQXDDTTOHPHZXDMZMAVUKOGBLUOPPQXGAHWHSXSTGTHORBWYPPSQYVGSUFA
UEZEQHLHZGSBIVUERLBAVSGYNKILHUYHPKCUMXKWGAJVNPKATFVZHXQREXI
GAQRYUBUWAAVRMLBKMKALVNAFYISECOEXCYNMKNPVMKZGAYIXMWH
```

3.
```
NQGJCKNXVMKEIBPROUQHNESMJLYDXWNHLLKKRQJLKLHQGJSOCUZWBCGNXVN
RPHHWDFECKOXDIUMN
```

Unit 46
Attacking the Fuer GOD cipher with monogram frequencies

In Figure 46.1 we see a histogram of monogram fitness for a 1000-letter piece of text that was enciphered with Fuer GOD and single-letter key A, then deciphered with one of the the twenty-two allowed key letters. Only when the decipherment was done with the same key as encipherment was the fitness above about 0.7. This suggests a method of attack. Suppose we have a ciphertext with period m, and we have determined that period from one of the methods we have seen. We cut the text into slices so that there are m slices and letters that are m places apart are put into the same slice. In other words, if the ciphertext is an ordered set of letters $\{c_i\}$, then slice number j contains all letters such that $i = j$ modulo m. For each slice, we decipher with each of the twenty-two possible key letters of the cipher and calculate the monogram fitness of the result. The highest fitness among the twenty-two tells us which key letter is correct for that slice. In the end, we can put the key letters for all slices together to get the keyword used to encipher the text.

We can make the attack faster by finding the monogram frequencies of the ciphertext once, and then rearranging them according to each row of the tableau. We wrote a function to do the rearranging in Task 2 of Unit 25. Each rearrangement can then be compared to our table of English monogram frequencies as in Unit 10.

Figure 46.1: Histogram of $\cos\theta$ *between the vector of monogram frequencies of English text compared to text enciphered with Fuer GOD and single-letter key* A *and then deciphered with one of the twenty-two possible key letters. Only a single datum (deciphered with key* A*) has a* $\cos\theta$ *above about 0.7.*

Programming tasks

1. Implement the attack.

Exercises

Break these ciphertexts. Plaintexts are in English. What are the keywords?

1.
```
SEGNMOVUMVHBXCCAVXBBRJWPJCURSRDSPWFVORNPGFZAGJWOFVDRGXSADQH
XUIRDFZZFIPGGAHMRMYNDBRKFYNBYUXPAMCAHKRZGDGANCKKZXUGWGGJYIO
MYNHNQHFZAVXBECJLZZZVDGHKZXIWDZJGJHVQTKVLNDIFKGWCWCRVBBGWJA
PNXVV
```

2.
```
JNRCEGGKGREDANBNXABOMYTQVXZYACHKGCGTUECGKJDMSCADRRGUAWDUGBR
JWNQVACHIGXTZWFXNIZHKRBKDVGZBRVHKGOACGNLWIRNOOAHAOUOYWZWBGM
WWRUCGKHUNMJNTMHWWQABXJYLCDESJRVVYODNAMRJYIRWILIARGOSRAEVOO
NMHUNDJRHDSLWEQDEMXOAEJQNRVVKFGKJRRBUKRSQZJMSGWJLRAOXSCAODU
MZ
```

Unit 47
Vigenère cipher

The *Vigenère cipher*, which was actually invented by Bellaso, is our simplest and one of the most constrained periodic polyalphabetic substitution cipher. Essentially it is a periodic Caesar shift cipher. The key alphabets are shifted versions of the regular alphabet, and the key is the set of shifts, which is usually expressed as the equivalent letters by a keyword. For example, if we want to use the keyword `SPACE`, then the key alphabets are

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	STUVWXYZABCDEFGHIJKLMNOPQR
1	PQRSTUVWXYZABCDEFGHIJKLMNO
2	ABCDEFGHIJKLMNOPQRSTUVWXYZ
3	CDEFGHIJKLMNOPQRSTUVWXYZAB
4	EFGHIJKLMNOPQRSTUVWXYZABCD

And here is how we might encipher a secret message:

```
PSST THE UNIVERSE IS REALLY BIG PASS IT ON
0123 401 23401234 01 234012 340 1234 01 23
HHSV XZT UPMNTRUI AH RGEDAY DMY EAUW AI OP
```

We can also understand the Vigenère cipher in terms of modular arithmetic. If we express the key as an ordered collection of L shifts $\{k_i\} = k_0, k_1, k_2, ..., k_{L-1}$, then encipherment of the plaintext $\{p_i\}$ to a ciphertext $\{c_i\}$ with a key $\{k_i\}$ is done with this equation:

$$c_i = p_i + k_{i \bmod L} \bmod 26$$

and decipherment by

$$p_i = c_i - k_{i \bmod L} \bmod 26$$

Note that if the period is one, then the Vigenère cipher degenerates to a Caesar cipher.

Some prefer to use a full table of all twenty-six possible ciphertext alphabets. This table is called a *tableau* (plural *tableaux*) or *tabula recta* (“right table” or maybe “square table”). The tableau for the Vigenère cipher is on the next page.

key letter	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ	label for permutation
A	ABCDEFGHIJKLMNOPQRSTUVWXYZ	$R_0 = e$
B	BCDEFGHIJKLMNOPQRSTUVWXYZA	R_1
C	CDEFGHIJKLMNOPQRSTUVWXYZAB	R_2
D	DEFGHIJKLMNOPQRSTUVWXYZABC	R_3
E	EFGHIJKLMNOPQRSTUVWXYZABCD	R_4
F	FGHIJKLMNOPQRSTUVWXYZABCDE	R_5
G	GHIJKLMNOPQRSTUVWXYZABCDEF	R_6
H	HIJKLMNOPQRSTUVWXYZABCDEFG	R_7
I	IJKLMNOPQRSTUVWXYZABCDEFGH	R_8
J	JKLMNOPQRSTUVWXYZABCDEFGHI	R_9
K	KLMNOPQRSTUVWXYZABCDEFGHIJ	R_{10}
L	LMNOPQRSTUVWXYZABCDEFGHIJK	R_{11}
M	MNOPQRSTUVWXYZABCDEFGHIJKL	R_{12}
N	NOPQRSTUVWXYZABCDEFGHIJKLM	R_{13}
O	OPQRSTUVWXYZABCDEFGHIJKLMN	R_{14}
P	PQRSTUVWXYZABCDEFGHIJKLMNO	R_{15}
Q	QRSTUVWXYZABCDEFGHIJKLMNOP	R_{16}
R	RSTUVWXYZABCDEFGHIJKLMNOPQ	R_{17}
S	STUVWXYZABCDEFGHIJKLMNOPQR	R_{18}
T	TUVWXYZABCDEFGHIJKLMNOPQRS	R_{19}
U	UVWXYZABCDEFGHIJKLMNOPQRST	R_{20}
V	VWXYZABCDEFGHIJKLMNOPQRSTU	R_{21}
W	WXYZABCDEFGHIJKLMNOPQRSTUV	R_{22}
X	XYZABCDEFGHIJKLMNOPQRSTUVW	R_{23}
Y	YZABCDEFGHIJKLMNOPQRSTUVWX	R_{24}
Z	ZABCDEFGHIJKLMNOPQRSTUVWXY	R_{25}

Table 47.1: Tableau for the Vigenère cipher. Plaintext letters appear across the top, key letters on the left, and ciphertext letters in the bulk of the table. Each row of the bulk is a permutation of the alphabet; along the right side are labels that we have chosen for these permutations; "e" represents the identity permutation. This tableau is the same as that for the Caesar cipher, with the one modification that letters are used for the key instead of numbers.

If we encipher a text with two Vigenère ciphers, then the result is equivalent to a single Vigenère. This is easy to see, since the alphabetic permutations of the Vigenère are rotations of the alphabet, and the composition of two rotations is another rotation:

$$R_m \circ R_n \;=\; R_{m+n} \tag{47.1}$$

where the arithmetic in the subscript is done modulo 26. The period of the combined cipher is the least common multiple (lcm) of the periods of the two factor ciphers. For the greatest impact, we would want to choose periods that are coprime, for a large lcm.

The collection $\mathcal{V} = \{R_n\}$, together with the composition of alphabetic permutations, forms a group (see Unit 17). The action of the binary operation is given by Equation 47.1. Furthermore, there is a natural isomorphism $R_n \to n$ between this group and $\mathbb{Z}_{26}$:

$$(\mathcal{V}, \circ) \;\cong\; (\mathbb{Z}_{26}, +_{26}) \tag{47.2}$$

We will see that this is a powerful result when we study quagmire ciphers later in this book.

Reading and references

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; chapters XII and XV.

Blaise de Vigenère, *Traicté des chiffres ou secrètes manières d'escrire*, Paris: Abel l'Angelier, 1586, HDL: 2027/ien.35552000251008, gallica.bnf.fr/ark:/12148/bpt6k1040608n, gallica.bnf.fr/ark:/12148/bpt6k94009991

Wikipedia: en.wikipedia.org/wiki/Vigenère_cipher

Practical Cryptography: practicalcryptography.com/ciphers/vigenere-gronsfeld-and-autokey-cipher

Crypto Corner: crypto.interactive-maths.com/vigenegravere-cipher.html

Fred B. Wrixon, *Codes, Ciphers & Other Cryptic & Clandestine Communication*, New York: Black Dog & Leventhal, 1998, pages 207-211.

Giovan Battista Bellaso, *La Cifra del Sig. Giouan Battista Belaso* [sic], 1553.

Paolo Bonavoglia, "Trithemius, Bellaso, Vigenère: Origins of the Polyalphabetic Ciphers," Proceedings of the 3rd International Conference on Historical Cryptology, 2020, ep.liu.se/ecp/171/007/ecp2020_171_007.pdf, DOI: 10.3384/ecp2020171007

Fletcher Pratt, *Secret and Urgent: The Story of Codes and Ciphers*, New York: Bobbs-Merrill, 1939; chapter VI, sections I-III; chapter XI, section I.

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, pages 148-150 and 240-242.

Programming tasks

1. Implement the Vigenère cipher. Feel free to use your script for the Caesar cipher, or to use the equation, or to use the tableau.

Exercises

1. Encipher this text with the keyword `PACMAN`.

```
DOOT DOOT DOOT DOO DOO DOO DOOT DOOT DOOT DOO DOO DOO DOOT DOOT
DOO DOO DOO DOO DEET DEET DEET DEET DEET DEET DEE WOCKA WOCKA
WOCKA WOCKA EEEEEEEOOOOP GAME OVER
```

2. Decipher this text with the keyword `VIGENERE`.

```
OPKZVKVRZZKGVTYIMEGWSMIWOLKWPVZFZLHCTMFZVVHEGXZWOIHIYPRWJQT
JVJKIZVLMSXPXCZKINRUMOZKQNMEIYCTFESBIICTXVPKLZUOHRMXLOMKRUY
EHMMJWVXJVZXAXNXZSIMGVAIUMOBNIAMTOIISIYITLDNLVRMEHZKNMSJIEW
TKAUMTLDALVRRTLAWXXUIZRYMIMCLVVVJRIPMGLZZ
```

3. Suppose we encipher a text with two Vigenère ciphers with periods 12 and 14. What is the period of the result?

4. Review Exercises 4 and 5 from Unit 20.

Unit 48
Brute-force attack on the Vigenère cipher

Review Unit 42 which contains our first brute-force attack on a polyalphabetic cipher. To perform attack on the Vigenère cipher, we need to try all one-letter keywords, then all two-letter keywords, etc., until we find an acceptable plaintext.

Python tips

The `product()` function from the `itertools` module can generate all possible tuples containing items from a set. For example, this block of code will create a list of all possible three-letter combinations. The length of the combinations can be replaced with a variable.

```
from itertools import product
letters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
combos = list(product(letters,repeat=3))
```

Programming tasks

1. Implement a brute-force attack on the Vigenère cipher. Use tetragram fitness to decide whether you have found the correct plaintext.

Exercises

1. Use your implementation to break this ciphertext from the 2015 British National Cipher Challenge:

```
WLHJLVVXLXHQLRRYUPLXWPHEXGWMRRZMOPEIWLHPRGDXLSQSIEVEIIKSXWH
MQXKIXOVIFXRVRJEIUPLRLXLWDQLRRVVXRTRZHVRRWLHVDXOMQIVFXXBSXR
HZHVNRRABSXQLKKXJIWPXGNCDRGJLRGWRQHSQILRWIUIVXLRJLLHLRJXKIU
IDXWLHZHVBPHEVXBSXALPOMQGRRYIQMHRFIWLHVHMFLVHROWSUMICRYWENI
SSVWHWVMRRRJLXKSZQXGKARYOHWLDXEIZSUXKXRCRYGSLLHEUEEMGSIJLZH
LXRGVHHWLRYVEQHIVDRFWIVRQRYUJUIQGKJUMHRGWSIULDTVXKIEVLXLWKA
RYOHSEBQRVHSUQDCEIWLHCFEQRRXDJISUHWSLARRGIULRAWLHCIIHPDFRYW
XKEWTHVKESWBSXWKSXPGEVOWLHQLJBSXADRWXRSXXEMGCRYUWRGDPOIGJUM
HRGWWLHROIDZHXKIPSQIBMQYQQDVNIGXUIDWXVBFLPOWLROSFOHVDXWLHJD
VHRGSIXKISPDXISUQLRIVLIGVLGKWWVDWVILALPOPHEYIWLHHHXDMOWLROS
FOHVBSXALPOJLRGXKINIBMQHRRRXWVBXRHRYEPHGUSVWPILXZMOPQSWARVN
EQHRYUPLXWPHKDQHALPOIQHEIISUILXKEVIYIQTUSSIUPBFHKXR
```

2. Now try another ciphertext from the BNCC (2002). This one has a slightly longer key, but it may take a lot longer to find it. The time it takes grows exponentially with the length of the key.

```
BPPOFDATNWBDLJZOIACTQJJXTJZOTSIUQTPZLPZAJDQUUAXUBIBTFMAVDMU
TIUUIDOMQFGPGZPRNFDBPPQTOCTEBIQDBXCFANUTMNMBFDQBXKPZKFDVJZO
UTMFZOMUAIQVDDGQFQPZMOSQOQWONMIMTGANNKUBEBFDKEMAZACDMVTQMJT
IWQUPHMERZPYAPGBIMUQFWOFWBZIEPZFEAJZTPZLPZIOPAVSOFEBZACTTWV
XLNQMUYMUTMSQIUZWPZIXQMLAVXQLOQAEMGQXMBEMDAUFMTPZMBEQGQISFP
FYIUQZJMTJEAIMTMIMTMGJZNMMUNMJMQQUIWVXLCQUPEBVZNPDBVZIUQQGM
ABDMTGTUANBZIUFIDWWGZMSHMTUEFDMUASOAKLADFDIMMVUQZOAZQQZIMXT
PZPBIMUOIFMEOQHMZJZBIQDJOQOUBZANUTMTFWDWWGOPFYQDMTTUPBHMTFW
SQLIQZFOPFYQTFZZUATGKIMXMQITMVUPQWQZTUWORWSFPFOCSUWVEUJZLBZ
LXUBIEWNMVZRITOQOMBJZOBZLSMBIQZQXMBEQOSIQBTJOIUUWOEQOFPFYWE
QZOIWSXLJYAVDMZACXUTMMOSQMPRKPGZTQBIQXMQITGZFUVTGKIMBSMVTMK
UUWOUAOQKFEABDQMKUVFCBXIOPQIAXFFPBFIMXCOBTFMABZBOQATMDPULFP
QXUTMNMBNTFFWTMBJENZKWVDKVDQPEQUKKPZKFDVJZOQDWGNICNIHQAEQDJ
OMBZLZACXUTMNMBNTFFWTMBJENZYGNMAUQZTZMFPNPDKVDZFZKZMABSMOFT
FYIOUINECSQPFRMMFCOMJMQBPPQTOCTEBFDUTIQUTGPGJVFITTQTRIJFPGG
TTQZWMVUUUVEBPRKPGZTQMOPMBHWVDBPMZSUDFMBBZIHDMFYMOFNBHWVDIC
XMUAPJYQUDCTFEFGVEQZTFIOPWOQIOABIQZBZLUTIUACSYMFFQOSEJXTJZL
FQLCQIQXMBECSQNPDCTNWUTXMQITQZFBTZUVUTMVECBXNBEPJAVZACSEMT
```

Unit 49
Attacking the Vigenère cipher with cribs

In Unit 43 we saw how to attack a periodic polyalphabetic substitution cipher with a crib when we know the permutations in the cipher's tabeleau. The tableau of the Vigenère cipher is far more orderly than that of the Fuer GOD cipher, and this will make the task easier. Since the Vigenère is really just a periodic addition cipher, what we will do is subtract the crib from the ciphertext at each position until we find a result that is consistent with the tableau and which reveals the key. We will know by sight that we have succeeded when the result of the subtraction repeats with the same period as the cipher.

We now work through an example. Below is a ciphertext encrypted with the Vigenère cipher. We will break it with crib `CELESTIALFIRE`.

```
XSIDIJAICIFLRVIWIEXVTPQMDIVLEWPOPRTVLRPWUBRTRSEFTPLQBAUHWPJ
XEZXWSSGPQUIPRAYEBWSIPAVMLHEFIEHJWMRIZBRRPQHJBXSEOIEMETRQGB
FTZWUXVHRZJOEZILTVQWNERTXDINGHZXTIEBRQPMQNMMGIYEGMICW
```

The first thing we should do is find the period. By the method of Sinkov, we see that it is 7:

Next we subtract the crib from the ciphertext at each position until we find a result consistent with this period. At position zero we see that the subtraction gives both `V` and `I` for the first key letter, and this inconsistency leads us to reject this placement of the crib.

position in key:	`0123456012345601 23...`
ciphertext:	`XSIDIJAICIFLRVIWIE...`
crib:	`CELESTIALFIRE`
result of subtraction:	`VOXZQQSIRDXUN`

The first position at which we have a consistent result is 13:

position in key:	`...601234560123456...`
ciphertext:	`...VIWIEXVTPQMDIVL...`
crib:	`CELESTIALFIRE`
result of subtraction:	`TELEMENTELEME`

We can immediately read off the key `ELEMENT` and decrypt the text to find this passage from *A Discourse of Fire and Salt* by Blaise de Vigenère, translated by Edward Stephens:

> There where the Celestial fire is all clear and shining as a lamp, whose flame should bee nourished with water of Life, mingled with a certaine composition of Camphire, Salnitre, and other inflamative matters.

Of course, we should check the tetragram fitness of any decryption before assuming that we have found the correct key.

Programming tasks

1. Implement the attack. Some of it can be copied from the attack in Unit 43.

Exercises

1. Break this ciphertext with the crib `NATIONALSECURITY`.

```
JGVROAEAQNFRFEZGUFQAEAQNNOSPUWRVWYLYGNPBSAJKSZSRZYTAZFOYLKN
HHTZZCVRZOCDFWVGECARPYHEQXGCGVNPSTLLWWZEQNGKSLXVEEXSQWFEEDL
AJQSRFUEGTSPKACYGDAVAHZKSGOEMDQWRUEOOCRQVNZOFEAZIEZGSCLOYSI
ENQDEZGFGRFRGXEEQMPFVPERPPJVYSRRMDQWVQGEZGLVGOQXQFGKENDCNQH
SEAPEFXRGWKLYDNNWRRBJRLEVHRFOXHCNLWHLLUEPXRPVUNBZDPFUEYHCEJ
QNVFCZEOUALCLLKOAVWTLJJBXRYSNIFWSLFFIAWECFCTVRNLDQFSLCTSNSU
DSDZWTQRWYAVSRQCCQRTRGEXSKLFHRGAEEF
```

Unit 50
Dictionary attack on the Vigenère cipher

If we have reason to believe that the keyword used to encipher a particular ciphertext is an actual English word, then we can perform a dictionary attack. Of course, the keyword might be a name, so we could use a list of known names and previous keywords as our dictionary.

Reading and references

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; pages 112-116.

Python tips

One way to read a file of words with one on each line is this. The flag `"r"` indicates that the file is being opened for reading only. When you use the `read()` function, there is no need to close the file; it is done automatically after reading.

```
words = open("words.txt","r").read().splitlines()
```

What this statement is doing is opening a text file with the function `open()`. The result is a file object. The `read()` function reads all of the file into a string object. Finally, `splitlines()` takes that string, splits it across end-of-line characters, and puts each line as a string into a single list.

Programming tasks

1. Implement the attack. Use tetragram fitness to determine when you have found the correct plaintext.

Exercises

1. Break this ciphertext. The keyword is a common English word.

```
ARVYIMZVGVQHFXJJWBFGGVHPSVFJRHOHGEFXYEPYMLOLKJXEWFNFABFKERQ
ISMIEEZSMPJXMZIPRXBGRCCWXUYTZXRSKGEGRLLGWSKEITXSOWVPDIGLGQE
XKSGVFVASWGOTHKIFKLXSKGEGRKQCJWBNIQEPBFIGRZXKHTFTIARIVJYGVV
JEGVEVKIFHXUKSVAVELQOWRVVRVJCRKMHFYUVHMGWGTYKWHKXMMSPEFQFMR
KTEMASPJXAWPCKILLENCIZSXKFRLARFZGTLIVYIGKEORRBHYNRXXVEPUAXS
OGEIWSGTPTMGKTRTAQWVRRWSVFKLXFEVZSGSPKEWMAXWKIHXWVRRWSVFJEG
VXGTLGGPQXCASHKJWNWHUVZXJENPITJWCXSTKETVWNDXPZWMDEUKAXWORLF
EAGNPHBKGQLVTYIFKIVZGQDTTFMGJJKGQWJMGYXJRXVJCRKSZJERYMVSTRI
STULCEHIJSOZWXVXQXMOWXJVTNTPKTEGGTRFVMMRKKCMGAGZKAARQEEKWZK
JIWKXCEHTJHVYIYABORCGGXDVEEDXJRXWAJHZGNDXVYIMSMPKIWHETKSYLL
GJXTFHCIHBKEJZKADCKEIYXMEZIGLENXSKAXJDXASXUVGNJMVPIQHITKWBV
IPKMYAIFRWTHVQSPXEPQEKTYSKEJTUXVYIUAKIVWMECUKIKQXJFWXWBRVVM
KWCPMLOLAKLXFWCKLHMKJKEGQGQDTTFCQIKHNITEQXFXCXIGUCYFYEVAKCP
BFKNPYLWXJRXISVVZGNDETRPZGVKKLFLSRISMWGVKLXAVFRXT
```

Unit 51
Hill-climbing attack on the Vigenère cipher

This is the same attack that we saw in Unit 45. The only differences are that the decryptor we use is now the one for the Vigenère, and that we try all twenty-six letters of the alphabet.

Reading and references

Practical Cryptography,
practicalcryptography.com/cryptanalysis/stochastic-searching/cryptanalysis-vigenere-cipher-part-2

Programming tasks

1. Implement the attack. Reuse most of the code that you used in Unit 45 but replace the decryptor. If you are clever, then create the attack as a function that takes the decryptor (or the cipher, if defined as a class) as an argument; you can reuse this function for any cipher whose tableau you know.

Exercises

1. Break this ciphertext.

```
HVSWYTMSBBDIYGKDIJSWNJVCWITWDMTIHSCIJHVFIRFSTCCMOKTNHWRTJHV
JAVBUYWFTHMTBSM
```

2. This ciphertext was encrypted with two Vigenère ciphers. The order in which they were applied is irrelevant. The result looks like it was enciphered with a single long key. The period is the least common multiple of the lengths of the two keywords.
    a. Find the period.
    b. Use the hill-climbing attack to find the plaintext. At the same time, you will find the combined key that looks like gibberish.
    c. Use the hill-climbing attack to “break” the key that you found in part (b), and recover the two keywords for the two Vigenère ciphers. It may be helpful to know that the lengths of the keywords are factors of the overall period.
    d. Now that you know the two keywords, decipher the text with two Vigenère ciphers.
    e. Repeat part (d) with the order of the two Vigenère ciphers reversed. Verify that the resulting plaintext is the same.

KBLFZROYITGKACGXWGSWSYOKTSYMRQZEPCZSRLAOWXUYRHVMTEFIQYZNVGU
LHCBZVBOKCJWVLKDCMNEXIYFNZLWFLTJBQFCUBFTEDBCXZDLZIMJFLAFSQZ
ROCMNUKISZGOWOBWZLGVIIICTXMOZXCFRHKCWRZSPYAXLJOIVPMAOLVRNUB
FXEBIBWOGZGCIYZLJKHGLWYQWCRXHRUHAHQMFOBSUFGQKBMCOCUNFQFERDR
QLRDMXLRCTFQXEPLNHCYACOHJFBEDDERTIJDOBUOLLNERICAMBSDVINVIZJ
HUJBRTGKAVXOOCHTXMUWGSOIIBXLRGAZAVNCJMBOYRYJXXFBTMDD

Unit 52
Attacking the Vigenère cipher as a periodic Caesar cipher

In Unit 46 we saw how to break a periodic polyalphabetic cipher by slicing the ciphertext so that we have collections of letters, and each collection was encrypted with one key letter. Then we can break each of the collections by comparing its monogram frequencies to the frequencies we would get by enciphering English text with each of the possible key letters.

Since the Vigenère cipher is a periodic Caesar cipher, we can use the technique of Unit 24 to break each slice of the ciphertext. First, we find the period *m* with the index of coincidence (Sinkov's method) or, if you prefer, one of the other methods. Then we cut the ciphertext into *m* slices or collections or columns (call them what you will), where the nth slice includes every m^{th} character of the text, starting with the nth. Each slice contains characters that were enciphered with one Caesar cipher. Therefore, for each slice we can find the shift using the technique that we used to break the Caesar cipher and convert that shift into one letter of the Vigenère's keyword.

Note that this method requires a longer ciphertext than the hill-climbing attack of the previous unit.

Here is an example to quell your fears. This text was encrypted with a Vigenère cipher and a key of length four:

```
KVQVFAMRVABMISMWZHQBZGFIUWZXYCEIUOKWDIEXECFFVQARTSUZVRAJSMF
LVFQEUSDEJIZMKSPXFUQXYSDYERQVFBQGFABETHMRUQARJCXMUOFIUUAZVF
ZQVBFMKKMWFBFLVCFLVFTEERMZRGFGFBSIIWQWFTZEKWARJKUHVZKHZGEMD
WXEIWZIMSDCISETVQFJICYIRQTSKVQVJDQEBWZKMODMFIEPRBSYRUQWRBPL
RJURXJMVZCGWTIEXFAEEERXENGFLVMIIISMPCVAAVJQVDCDIFFXIJGPIGSZ
HVBFYGCZEEROSEBQGKSPAZHTXYSSVVOFGVBFVRZBSNSD
```

We break it up into four slices:

```
0:    KFVIZZUYUDEVTVSVUJKFYEFFTUJUUVVKFVVERFIFKJVZDIMIVIRKJB
      MFRRRRXZTFENVICVDFJGVGEEKZYVVRN

1:    VAASHGWCOICQSRMFSISUSRBAHQCOUFBKBCFRGBWTWKZGWWSSQCQVDW
      OIBUBJJCIARGMSVJCFGSBCRBSHSOBZS

2:    QMBMQFZEKEFAUAFQDZPQDQQBMAXFAZFMFFTMFSQZAUKEXZDEFYTQQZ
      DESQPUMGEEXFIMAQDXPZFZOQPTSFFBD

3:    VRMWBIXIWXFRZJLEEMXXYVGERRMIZQMWLLEZGIWERHHMEICTJISVEK
      MPYWLRVWXEELIPAVIIIHYESGAXVGVS
```

Concentrate now on slice zero. Its monogram frequencies are

A	0	J	0.059	S	0.012
B	0.012	K	0.071	T	0.035
C	0.012	L	0	U	0.071
D	0.035	M	0.024	V	0.176
E	0.071	N	0.024	W	0
F	0.118	O	0	X	0.012
G	0.024	P	0	Y	0.035
H	0	Q	0	Z	0.059
I	0.071	R	0.082		

We can shift this frequency distribution from 0 to 25 steps and calculate its monogram fitness as the cosine of the angle it makes with our known frequencies of English in some 26-dimensional vector space. The clear winner is a shift of 17, corresponding to key letter R.

shift	**key letter**	**cos θ**	**shift**	**key letter**	**cos θ**	**shift**	**key letter**	**cos θ**
0	A	0.468	9	J	0.423	18	S	0.510
1	B	0.628	10	K	0.482	19	T	0.406
2	C	0.644	11	L	0.392	20	U	0.441
3	D	0.529	12	M	0.518	21	V	0.608
4	E	0.557	13	N	0.671	22	W	0.364
5	F	0.522	14	O	0.497	23	X	0.471
6	G	0.562	15	P	0.394	24	Y	0.464
7	H	0.495	16	Q	0.565	25	Z	0.469
8	I	0.471	17	R	0.941			

If we repeat the process for the remaining slices, we find that the key is ROME. We then decrypt the text and find this passage from *History of Julius Caesar* by Jacob Abbott:

> The Roman empire, as it existed in those days, must not be conceived of by the reader as united together under one compact and consolidated government. It was, on the other hand, a vast congeries of nations, widely dissimilar in every respect from each other, speaking various languages, and having various customs and laws. They were all, however, more or less dependent upon, and connected with, the great central power.

Programming tasks

1. Implement the attack. Re-use pieces of attacks that you have already written.

Exercises

1. Break this ciphertext with the attack.

```
BNAGGUTAGGOTGNTALTNTVEWFZSMARGSGWGNWKTBTWEGQGQGWBTWAMEKTNBZ
KEHZRRGGMYOFXAFZZTVNTQFIAIGNTQHWTZEKKVCRSWUMSUJXQSGDEQAZGWM
FVSBZVRDLMAJSGABYLHBUKOHZYJAAWCKLAIGYGFMBTWZMGYERURYKTOROFT
JBZLEMNEWTZUGKIIFYWWAVTUXQJXGMMZEFHBROKAWHRVAIIKCGWJTLAQFXA
ZPGLJHUGNWLBNXLHVYEZHXRISGSRKHFMGUYXBUKJEWIKUTVZKFWGBAJEQSK
TNBYUNXKNTTKMNQQHCENWTZGCSESRJGNBGNALUBXFBVTOVHVGHWEQRBWPPN
ZALIJGZNVQXWWUVRDBWAHGMBYKKPIFNWWCCUFMPRYZHZRYWXUFOEGWGGDHV
ROFUMVTYTTBTWTPHTVKMQSAETVUFVIFZSPILYDHWXOFZNBXSAWZK
```

Unit 53
Gronsfeld cipher

The *Gronsfeld cipher* is the same as the Vigenère, except that the key is a string of digits rather than letters. Because the key is made from digits, the largest shift is nine.

The attacks that we developed for the Vigenère cipher also work for Gronsfeld, except for the dictionary attack. Notice that they only needs to try shifts zero through nine for each digit of the key, rather than run over 26 letters.

key digit	**plaintext alphabet** ABCDEFGHIJKLMNOPQRSTUVWXYZ	**label for permutation**
0	ABCDEFGHIJKLMNOPQRSTUVWXYZ	$R_0 = e$
1	BCDEFGHIJKLMNOPQRSTUVWXYZA	R_1
2	CDEFGHIJKLMNOPQRSTUVWXYZAB	R_2
3	DEFGHIJKLMNOPQRSTUVWXYZABC	R_3
4	EFGHIJKLMNOPQRSTUVWXYZABCD	R_4
5	FGHIJKLMNOPQRSTUVWXYZABCDE	R_5
6	GHIJKLMNOPQRSTUVWXYZABCDEF	R_6
7	HIJKLMNOPQRSTUVWXYZABCDEFG	R_7
8	IJKLMNOPQRSTUVWXYZABCDEFGH	R_8
9	JKLMNOPQRSTUVWXYZABCDEFGHI	R_9

Table 53.1: Tableau for the Gronsfeld cipher. Plaintext letters appear across the top, key digits on the left, and ciphertext letters in the bulk of the table. This is merely the first ten rows of the Vigenère tableau.

Reading and references

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; pages 117-118.

Practical Cryptography: practicalcryptography.com/ciphers/vigenere-gronsfeld-and-autokey-cipher

Fred B. Wrixon, *Codes, Ciphers & Other Cryptic & Clandestine Communication*, New York: Black Dog & Leventhal, 1998, pages 213-214.

Fletcher Pratt, *Secret and Urgent: The Story of Codes and Ciphers*, New York: Bobbs-Merrill, 1939, chapter VIII, section I.

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, chapter 4 and pages 245-46.

Python tips

When you define a class of objects in Python, it is possible to do so as an extension of a previously defined class. The new class inherits the things in the original class. For example,

```
class oldClass():
    def __init__(self,number):
        self.value = number
    def printNumber(self):
        print(self.value)

class newClass(oldClass):
    def doubleNumber(self):
        print(2*self.value)

x = newClass(5)
x.printNumber()  # expect 5
x.doubleNumber() # expect 10
```

Programming tasks

1. Implement the Gronsfeld cipher. Feel free to simply make wrappers around your functions for the Vigenère cipher, or, if you implemented the Vigenère as a Python class, then define the Gronsfeld as a class derived from it.

2. Write a function or script to brute-force a ciphertext enciphered with Gronsfeld. Feel free to copy your work for the Vigenère and make the appropriate changes.

3. Make a copy of your hill-climbing attack on the Vigenère and modify it for Gronsfeld.

4. Make a copy of your attack on the Vigenère as a periodic Caesar cipher and modify it for Gronsfeld.

Exercises

1. Decipher this ciphertext with key 78345024.

```
VCWHFMPIKASSYOWXPADCTNGXDWZLDTJIUQWMXTKQLBRHTIVLLTOMXMWVRGI
MJMAPVZGJNECWVTGMJRCRKIIIFRFAOIWRJEFALNHEWWJSRVRAXIVAOMQRTN
GGHVFEQLQYYXRAJRVSHKFSZNVCLBZLTWQYSLKEAEVLVCJLYTJIVTGQFNVSO
IYIMAFWVUXGMBNSVLLRMIO
```

2. Brute-force this ciphertext.

```
TNMDFHBUXUYQLOXBWWLDXMTDYQRMJRMTDUIVFXNSKFPYQPSJMTALXCKFARK
XYYQLXCYFRNMCMTADFAKUJAMFHDQHIKLJWSTBAFQTJQVBQHWMHYQPSPPYRZ
YXZFHDMJABJZYQPXOVWNZYBHAJNJAVZPOFWKXCLWWDMRJMRUYQLANYDCOTD
NMCYJWLBBAMNMJJYXXINCAJAPXRAINHYQPXUPYCSJVVWNIZCVKCOJPVTMAT
CYJJABQPHQAMNYJRMTDUIBWJJRBRSQRVKCOJXAMNYYQPSPZNBHBCOJALNLH
SWVYFLQUYJYLFCOTFAMNYJRLSCLWNKXXMZUSBJZNXMXUBRKLWJAYQLRXTJW
ANWDMRJMROFMHGJUIXUJMAMNAWDLBJF
```

3. Break this ciphertext however you please.

```
CVPJIWAGLSMGXWVLNLKXAQSEYNHBRHCYXHXHFORSXWKVMFROCJHPTCHAURF
TRGNWUOUKKJXHESZHAGGISZRJJRLDGOUKDNHKWXGAOWILBGWHNYVYGFTWKS
XRHASJFICUMJZWNTQIHNDWFJUNFRCHUESIKACGIHGOBCQDLKUVCQPTRLLGV
PNKVFJHRJOVCJHPFXWHKQOUVRBKWITTWQCWHFYQXVEEJTGNEUIJJDBERMRA
QRUWIHHBRVVOUVRWGQTXOQYQZEWCKNTHIXZKNQSPTYLCKRNYNDCJDSSUWQW
ULJJEJENTMKEACQDNTJAGSRTGFQQICTSPDPLSRGJJKQSYZKNORRJGGECQCJ
JRYRRSNZLXPSAWZLNUDSBKOUCVALGLWUWIYYUNCFTNUQJTBAIBHAUDRNKV
```

4. Break this ciphertext.

```
NQRTSGRRTSGFQHCWYWSNBFLPWPTZIRMLRSDTNWXLMGXRNQJVAEVYGSWVYKI
LISHXOOFWLEZGOECSGHVRQJVFUBUHXEZBKSWIXYIRGSDYGPYBVAVXWVAQLK
XAEFBXOUWPGGZJJSRALDVDMSDRDAVXIEHJGYNLJUXHMLDXENNUWTPJDXEAT
PILMYWYCMXDRDATPIFZJQGHJJDRSISGXHMSKIABJVSMMWDHIAMHWAVIWLEV
KHILQSJVABMHVSQHNLEEJQXTWQRSKNTUWOUJSERAQHC
```

Unit 54
Beaufort cipher

The *Beaufort cipher* is a periodic polyalphabetic substitution cipher in which the key alphabets are shifted and reversed versions of the regular alphabet. Encipherment and decipherment are the same operation. The key is series of shifts, which is typically represented as a keyword. Each letter of the keyword gives the first letter of the key alphabet that it represents.

For example, let's encipher a short message with the key `BEAU`. The key alphabets are written here under the plaintext alphabet:

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	BAZYXWVUTSRQPONMLKJIHGFEDC
1	EDCBAZYXWVUTSRQPONMLKJIHGF
2	AZYXWVUTSRQPONMLKJIHGFEDCB
3	UTSRQPONMLKJIHGFEDCBAZYXWV

Here we have labeled each letter of the text with the key alphabet that is used to encipher it:

```
HARRY SAYS BEAUFORT CIPHERS ARE BEAUTIFUL AND STRONG
01230 1230 12301230 1230123 012 301230123 012 301230
UEJDD MAWJ DWUHZMDI CSFUAJC BNW TXEGBTZGJ BRX CINMHV
```

We can also understand the Beaufort cipher in terms of modular arithmetic. If we express the key as an ordered collection of L shifts $\{k_i\} = k_0, k_1, k_2, ..., k_{L-1}$, then encipherment of the plaintext $\{p_i\}$ to a ciphertext $\{c_i\}$ with a key $\{k_i\}$ is done with this equation:

$$c_i = k_{i \bmod L} - p_i \mod 26$$

and decipherment by

$$p_i = k_{i \bmod L} - c_i \mod 26$$

Notice that the two equations are the same, but with p and c exchanged. This means that encipherment and decipherment are the same process; a cipher with this property is a *reciprocal cipher*.

The attacks on Beaufort (brute-force, dictionary, crib, hill-climbing, using monograms) are in principle and structure the same as for any periodic polyalphabetic substitution cipher whose tableau is fully known. So we can take the attacks we made for the Vigenère or Fuer GOD and modify them to

attack the Beaufort. Keep in mind that the Beaufort permutations are rotated atbash keys rather than Caesar keys.

Here's a fun fact: The Beaufort cipher is the same as a Vigenère followed by an atbash cipher. Try it and see. The key for the Vigenère in this case is the Beaufort's key enciphered by the atbash. We can reverse the order of the atbash and Vigenère, so long as we also adjust the key. When we studied the atbash cipher, recall that we labeled its alphabetic permutation z:

$$z = \texttt{ZYXWVUTSRQPONMLKJIHGFEDCBA}$$

Then we can write the alphabetic permutations of the Beaufort cipher as the composition of z with a rotation:

$$b_n = z \circ R_n \tag{54.1}$$

where the numbering of the b's can be seen in the tableau (Table 54.1, below), and the R_n are the rotations found in the Caesar and Vigenère tableau (Tables 20.1 and 47.1).

Here is another weird fun fact: If we encipher a text with two Beauforts, it is the equivalent of using one Vigenère. To show that, we first need to understand that making a rotation of the alphabet is the same as reversing the alphabet, making the rotation in the other direction, then reversing the result. In terms of permutations,

$$R_n = z \circ R_{-n} \circ z$$

where any arithmetic in the subscripts in the above equation and in what follows is done modulo 26. From the preceeding we can say that

$$z \circ R_n = z \circ (z \circ R_{-n} \circ z) = (z \circ z) \circ R_{-n} \circ z = e \circ R_{-n} \circ z = R_{-n} \circ z$$

And so

$$b_n = R_{-n} \circ z \tag{54.2}$$

We also know that acting with one rotation followed by another is also a rotation (Equation 47.1):

$$R_m \circ R_n = R_{m+n}$$

Now, if we compose any two of the b's we get a rotation:

$$\begin{aligned} b_m \circ b_n &= (z \circ R_m) \circ (z \circ R_n) \\ &= (R_{-m} \circ z) \circ (z \circ R_n) \\ &= R_{-m} \circ (z \circ z) \circ R_n \\ &= R_{-m} \circ e \circ R_n \\ &= R_{-m} \circ R_n = R_{n-m} \end{aligned}$$

Since using two Beaufort ciphers enciphers each letter with two substitutions using two b's, this proves that the result is a Vigenère. What its key is is another story.

key letter	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ	label for permutation
A	AZYXWVUTSRQPONMLKJIHGFEDCB	b_{25}
B	BAZYXWVUTSRQPONMLKJIHGFEDC	b_{24}
C	CBAZYXWVUTSRQPONMLKJIHGFED	b_{23}
D	DCBAZYXWVUTSRQPONMLKJIHGFE	b_{22}
E	EDCBAZYXWVUTSRQPONMLKJIHGF	b_{21}
F	FEDCBAZYXWVUTSRQPONMLKJIHG	b_{20}
G	GFEDCBAZYXWVUTSRQPONMLKJIH	b_{19}
H	HGFEDCBAZYXWVUTSRQPONMLKJI	b_{18}
I	IHGFEDCBAZYXWVUTSRQPONMLKJ	b_{17}
J	JIHGFEDCBAZYXWVUTSRQPONMLK	b_{16}
K	KJIHGFEDCBAZYXWVUTSRQPONML	b_{15}
L	LKJIHGFEDCBAZYXWVUTSRQPONM	b_{14}
M	MLKJIHGFEDCBAZYXWVUTSRQPON	b_{13}
N	NMLKJIHGFEDCBAZYXWVUTSRQPO	b_{12}
O	ONMLKJIHGFEDCBAZYXWVUTSRQP	b_{11}
P	PONMLKJIHGFEDCBAZYXWVUTSRQ	b_{10}
Q	QPONMLKJIHGFEDCBAZYXWVUTSR	b_9
R	RQPONMLKJIHGFEDCBAZYXWVUTS	b_8
S	SRQPONMLKJIHGFEDCBAZYXWVUT	b_7
T	TSRQPONMLKJIHGFEDCBAZYXWVU	b_6
U	UTSRQPONMLKJIHGFEDCBAZYXWV	b_5
V	VUTSRQPONMLKJIHGFEDCBAZYXW	b_4
W	WVUTSRQPONMLKJIHGFEDCBAZYX	b_3
X	XWVUTSRQPONMLKJIHGFEDCBAZY	b_2
Y	YXWVUTSRQPONMLKJIHGFEDCBAZ	b_1
Z	ZYXWVUTSRQPONMLKJIHGFEDCBA	$b_0 = z$

Table 54.1: Tableau for the Beaufort cipher. Plaintext letters appear across the top, key letters on the left, and ciphertext letters in the bulk of the table. Each row of the bulk is a permutation of the alphabet; along the right side are labels that we have chosen for these permutations; the one called "z" is the permutation that reverses the alphabet.

Reading and references

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; pages 121-125.

Wikipedia, en.wikipedia.org/wiki/Beaufort_cipher

Practical Cryptography, practicalcryptography.com/ciphers/beaufort-cipher

Crypto Corner, crypto.interactive-maths.com/other-examples.html

Fred B. Wrixon, *Codes, Ciphers & Other Cryptic & Clandestine Communication*, New York: Black Dog & Leventhal, 1998, pages 214-216.

Fletcher Pratt, *Secret and Urgent: The Story of Codes and Ciphers*, New York: Bobbs-Merrill, 1939, chapter XI, section I.

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, pages 202-203 and 240-242.

Programming tasks

1. Write a Python script that can construct the tableau for the Beaufort cipher.

2. Implement the Beaufort cipher. For Tasks 2-6, modify as appropriate the attacks you built for the Fuer GOD and Vigenère ciphers.

3. Implement a brute-force attack on a ciphertext encrypted with Beaufort.

4. Implement a dictionary attack on a ciphertext encrypted with Beaufort.

5. Implement the hill-climbing attack on a ciphertext encrypted with Beaufort.

6. Make a modified copy of your attack that uses monogram frequencies on each slice of a ciphertext encrypted with the Vigenère cipher so that it can now attack a ciphertext encrypted with Beaufort.

Exercises

1. Decipher this ciphertext with key `HUSBAND`.

```
ZBKJAUMNBLHNFIDDABPCFHSIOMRSDRMXXUWHBSJSAXWQGBNFQSGAJWVLZGF
NVNXTGPWMWKNHOPGVKGQKOENQOGNBEFYDNEFWSZQJKIHCZXHEFNUWDPOXPF
QBCEKFFZLCEWITBAUGBNBDJTONNGVPPKKIUZUBOKSAXHHOTUGCTABUMZAON
KJHWJONKJIZHDJHWSQZEMFITJRZHPJMIKAQAHJWPNHPTNHYHIKQSJLONSIT
JVPSEOIFADDOYHGZQMMUHIJWFBNLJMOWEWIZRDGFXMWPONOKMIKAQKKXNJB
NZXJV
```

2. Brute-force this ciphertext.

```
QEPHSWKMTGEWHMMGACOEJAEUERWXAJGWNPXMWTMFYLBAYKMWHGTSJSFYPKL
HHAMBRMGQNOGSFSCDKJAAAFBNELTKRUDEJZWYYTABLMKRHONWALOOQNAXMB
ASZXMYYTZWEEKPPMAKTARQGZEPKLHHAAFBIOAHLAESLWBGLSNHEZHLAEENP
PWHKMAWECKXAXAMNBJKJSFYMNZHLAIDTWORAFBJOYGHEHKRUZXWQWJQKPSL
SERELANOJWBWVKRXGGIOTVMNMWWNMYJKSARQGZLTOSMYLTWXWFAFONSZWIS
BAGPXBWBTTCFQFOSZONSFSCAQGHITORWXAJKZSFBOUMWHZSFJMHKNZEJKTC
DEGAWNMDWNQJOCMNZSFIABATEKIOMAFBZBWNMWNMKLZXWBAABQVOJWBGVYR
WBEPKSWOLAFBWAPWQWAHTCWXWFAFONOUXCDQIMAHAKYLENYPLONTSRXEZOO
LTSLSZNWCKSBAISMHBQNMSMBEPDNSFCSDTWZQLBAFORHGAVBQOPAPKDWBEH
OTCALWDPSFYSFLMZXWALJOAHSRXGAHLQXKCAHTCIRMQUSFYLOQLHAILEHAQ
VNLTORSSCYEKNZWHLWULLSGAHEYWZLMAAASMIEQNSMSQEN
```

3. Break this ciphertext with a dictionary attack.

```
WWOQTJIVANTKVMCOEILZMBYIGISXAZHDITZIEQARODBLNMKTXFECYSJJVTQ
HOTILGSCMTFDVBAVMKTCGIJTXJRASNVOWTAWESFQVWVCRFDVERHWSMGFJVF
AUAGBWHPVQXFLKTXJTKKABVKVMAISHACLYSWJDDOIFDHYKUWBJLGSJENJUX
MCDLKJCGSKVZTLDOIFDHERFDSMMNUAAZZFVMDCSXYQBDHYKHPVFALWXXAUA
OUZNTKJEATVFAICXZWUCXIJJFQXTNOBHQWWIYQKHGKLTFKYNUAAFKAQAZLG
SSMTCENYQDEJETCSLTXJFKKAVEKUCGFQUEADHISTKWMRKUHVWVDCMMWIWIM
MLAYSJTGZLGSSERKISUEATOXTJWWMRUWVAQVHHINJIJFQBSZIELSZINSWLI
BFD
```

4. Break this ciphertext with the hill-climbing attack.

```
ZDJPPAIENXVBYODKGKPZNZKTDIGZUDUMSIKZCUYWNGZAXNGZVMZCEHLMXSO
DCWBWZYSMQZAWTTZIKVSQJWTJDFB
```

5. Break this ciphertext with the attack that matches monogram frequencies.

```
LTTFHWGKAFAASOUXHJRDSOZCNCWPRDJYSULDAGHGWZLKEKGKARKJHVUXLZM
VLQKKFEXJTHCJKOMVHXYTOBKGLXURWHMALXYFEQXJZVOJONPRGVYTZDRRSB
UELOEEIHYDSEWEQOLCSGTTKXYKICEZSKZKFEYDYKOQZGAMXXYCBTMNDBJDO
VQAYGPEATSNSGQGAAAZWKPRLZMOLQCRZAAPEKQCQHTGLNOTSOQANWOYOOWF
HWUXZAAODWKGQOWEFXOWGPEEIAAISPWDYORZUZRGXBWOSWUZPHUILHPCHOL
GPOQYLEYDKVWESAJDOWQVHWJVEQLJXBKWNOXNJGJMFTLKDBWIENTOHRAGOE
XJTKPGBBGVEKPRLDIDUEUXMCWYPOKQDTRKDCBQZUQVLBZKMHWELXYKQOWDY
SOQHEMNDPYDKVEEIJOTPHGZXBYXPAAVXURHHZAXWAPRLDMDGOUXZAAZDVJC
FBNDXCVKBETTRVJGBZRLLSOTPTNFXTUXMHSXTMRGNQQFZALXKUYYXBUELO
```

6. If we encipher a text with a Beaufort cipher using the key `BEAUFORT`, then what must the key be to get the same ciphertext with a Vigenère followed by an atbash cipher? What must the key be if the atbash is done first?

Unit 55
Variant Beaufort cipher

The *variant Beaufort cipher* (also sometimes called the *German Beaufort cipher*, or simply the *variant cipher*) is the inverse of the Vigenère cipher. Encipherment with variant Beaufort is the same process as decipherment with Vigenère. In fact, by modifying the key, a variant Beaufort cipher can be converted to a Vigenère. The modification to the key is to apply an atbash and a Caesar shift of one.

Reading and references

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; pages 121-125.

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, pages 202-203 and 240-242.

Programming tasks

1. Write a script that can construct the tableau for the variant Beaufort cipher.

2. Implement the variant Beaufort cipher. Feel free to make a wrapper around your implementation of the Vigenère cipher.

3. Wrap your brute-force attack on the Vigenère cipher so that it can attack a ciphertext encrypted with variant Beaufort. Really all that changes is the assignment of the key letters to rows of the tableau.

4. Wrap your dictionary attack on the Vigenère cipher so that it can attack a ciphertext encrypted with variant Beaufort.

5. Wrap your hill-climbing attack on the Vigenère cipher so that it can attack a ciphertext encrypted with variant Beaufort.

6. Wrap your attack on the Vigenère cipher that uses monogram frequencies so that it can attack a ciphertext encrypted with variant Beaufort.

Exercises

1. Decipher this ciphertext with key `TOWER`. What is the equivalent key for Vigenère?

```
YMTQWGQPCALIMJCVFLAVVEXXNHGXEOBXGDRSPYJMLDXDNZGRSQLZWDNDMWP
FLXZAHLMVOXSPXDNLZGDJUFVABZELQCOQVEWAAEPXDQVSQPOLHJFURWOVDI
OCHZHDJKZIECOQVOCHUVOWVDHKXYNYPZBUXAJAFLACVBAWBHXMPCSQAEWKA
ASQLZXDNLZGDJUFVABZIEJCLPXKPVUROQLBPWLLPLAAZQPBKLZIWCOUXWWK
OVENKDELDULIHAHBYJILXPACKAAJHVGVDJPD
```

2. Brute-force this ciphertext. What is the key? What is the equivalent key for Vigenère?

```
IHPHYYEHPRYKFNDELKOGYLUIPYZTIWNOLRCCHNAAMALMPOIZTBAFOHLFANA
PHJKRNNACPOZWYIQNAIAHKFYTTLWOLZIHWRSLELOOHWLVAAOPYUJDCJFLKN
NKFCPSIIEFETNHEXESNWNWAAQWYQWSMETNENAPHYWRNESNDIGOEFBBUOIFD
AFHWUNDQDOMASOZDYJDCOAJLEUNAHYEMKMYUEUNSUCOWWUMADUPTBATCIEM
QCBLUVHIWAXWETYIEHPAHZGURELESYPOMKMUJYMPRUJGYYOHFEWPULAS
```

3. Perform a dictionary attack on this ciphertext.

```
POOAQRXNAJHEAZXOHRHOZZDJLBOTWXVWUMWDDKWPBAAHMGCWUMYOTJRBZLD
OWTNNZROLITMOHWZMIXXKUBWNLHDYJJJEMXBWUMXUZONZNXHDITMWSUPHMU
UZYXIAVINNLCKLLKGWJCHYQTCDLJJCQKWPDJUCITYHLJOEIYVAAQAYXRNWZ
NZONUUZAQAWQYNNFXQNOYCAYBKFBUMWFBKBMOCWUMBATRXJPOJOBOOOADZI
WABUVDPHVUUKUPARKXJRLQESITLELWPAXVNPPCASNUACVCGIVMBPVWKRJGU
HHWPYVKCDLKNADKXNJXKPMXXBAQAWWUMWUMSADKBKINETIRBKHWZMIERWUM
WLTSHLPAWTMYBDHAATPKPNHEAWPKAAAQASMGWZAQAIZIAAHCEOVYUEL
```

4. Perform a hill-climbing attack on this ciphertext.

```
WXDJUOJOYHKPDGPTAWDWSIWPOAXVYSHXLGJGIGYONPAJPHTJCOAEGPLOXKC
YLDKCXEWALZUHSLZHTKRNEPVGJGDFKUTPAJOETZMSEPYCNLNLFALDTQPEGK
YJDIZCPUGLJASPDJWDKSLYEIZCUAGWUWIIALCOCLFASWALCLTOGHLNGSYOB
WYJDYGGJTWWBWNRW
```

5. Break this ciphertext with the attack that matches monogram frequencies.

```
ETWZQJLEGVORLBGWDJTPGEIUZXADQQTZSTAAPXQKAGEMYMUAQDGVFBQFZMO
BEFSOQJLESOMEOQFETRCQAVEGZAVBIBCAKMFEPQKWZRZROPUPSNGZQJSULM
MAOFZMAGSQJZQQCAKMEFSQZIPGHAUPUYODWVIUZIWZQYTWWBTREIGZAFPFJ
MQFLZVWZRHMKKMYWQVAZBHIZQFRLZVBTRZFZMDEZEWZQQETWGIRCQSASBZP
SVPULBHGMFMGKGMAOOZMQEQGDIERGQJBIBNTATPEPZAVFUPIGZXQHQJMAAW
KKVAJHTABQJLEEWDRBGAMFNYPYMZGWQLPMACAKMDRODGAQEPPDQWRONWBFR
CFGZGALNGCFVYFZMYRLPGEENYPXQQYOEKMQXTZYNXBHQJAMAOOSBOUTZYJG
GEQJNXVPETCFFYAOETVEQKIFNETGUQJTFZPQEXALPQELZVPQYAQVPQEHULP
TRCTGCERHAJSAECQSLFBSQJETRYFZMDRHMKVAGSUFOFBOA
```

key letter	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ	label for permutation
A	ABCDEFGHIJKLMNOPQRSTUVWXYZ	$R_0 = e$
B	ZABCDEFGHIJKLMNOPQRSTUVWXY	R_{25}
C	YZABCDEFGHIJKLMNOPQRSTUVWX	R_{24}
D	XYZABCDEFGHIJKLMNOPQRSTUVW	R_{23}
E	WXYZABCDEFGHIJKLMNOPQRSTUV	R_{22}
F	VWXYZABCDEFGHIJKLMNOPQRSTU	R_{21}
G	UVWXYZABCDEFGHIJKLMNOPQRST	R_{20}
H	TUVWXYZABCDEFGHIJKLMNOPQRS	R_{19}
I	STUVWXYZABCDEFGHIJKLMNOPQR	R_{18}
J	RSTUVWXYZABCDEFGHIJKLMNOPQ	R_{17}
K	QRSTUVWXYZABCDEFGHIJKLMNOP	R_{16}
L	PQRSTUVWXYZABCDEFGHIJKLMNO	R_{15}
M	OPQRSTUVWXYZABCDEFGHIJKLMN	R_{14}
N	NOPQRSTUVWXYZABCDEFGHIJKLM	R_{13}
O	MNOPQRSTUVWXYZABCDEFGHIJKL	R_{12}
P	LMNOPQRSTUVWXYZABCDEFGHIJK	R_{11}
Q	KLMNOPQRSTUVWXYZABCDEFGHIJ	R_{10}
R	JKLMNOPQRSTUVWXYZABCDEFGHI	R_9
S	IJKLMNOPQRSTUVWXYZABCDEFGH	R_8
T	HIJKLMNOPQRSTUVWXYZABCDEFG	R_7
U	GHIJKLMNOPQRSTUVWXYZABCDEF	R_6
V	FGHIJKLMNOPQRSTUVWXYZABCDE	R_5
W	EFGHIJKLMNOPQRSTUVWXYZABCD	R_4
X	DEFGHIJKLMNOPQRSTUVWXYZABC	R_3
Y	CDEFGHIJKLMNOPQRSTUVWXYZAB	R_2
Z	BCDEFGHIJKLMNOPQRSTUVWXYZA	R_1

Table 55.1: Tableau for the variant Beaufort cipher. Plaintext letters appear across the top, key letters on the left, and ciphertext letters in the bulk of the table. Each row of the bulk is a permutation of the alphabet; along the right side are labels that we have chosen for these permutations; "e" represents the identity permutation. The rows of this tableau are the same as that for the Caesar and Vigenère ciphers but in a different order.

Unit 56
Porta cipher

The modern versions of the *Porta cipher* (actually originally invented by Giovan Battista Bellaso) use a set of thirteen key alphabets. The key is again a keyword, but there are two common versions for assigning key alphabets to keyword letters. Below is their tableau. Notice that each of the key alphabets is *reciprocal*, i.e., it is its own inverse. Thus, the Porta cipher is a reciprocal cipher and is also its own inverse.

key letters ver. 1	ver. 2	**plaintext alphabet** ABCDEFGHIJKLMNOPQRSTUVWXYZ	**label for permutation**
A/B	A/B	NOPQRSTUVWXYZABCDEFGHIJKLM	p_{13}
C/D	Y/Z	OPQRSTUVWXYZNMABCDEFGHIJKL	p_{14}
E/F	W/X	PQRSTUVWXYZNOLMABCDEFGHIJK	p_{15}
G/H	U/V	QRSTUVWXYZNOPKLMABCDEFGHIJ	p_{16}
I/J	S/T	RSTUVWXYZNOPQJKLMABCDEFGHI	p_{17}
K/L	Q/R	STUVWXYZNOPQRIJKLMABCDEFGH	p_{18}
M/N	O/P	TUVWXYZNOPQRSHIJKLMABCDEFG	p_{19}
O/P	M/N	UVWXYZNOPQRSTGHIJKLMABCDEF	p_{20}
Q/R	K/L	VWXYZNOPQRSTUFGHIJKLMABCDE	p_{21}
S/T	I/J	WXYZNOPQRSTUVEFGHIJKLMABCD	p_{22}
U/V	G/H	XYZNOPQRSTUVWDEFGHIJKLMABC	p_{23}
W/X	E/F	YZNOPQRSTUVWXCDEFGHIJKLMAB	p_{24}
Y/Z	C/D	ZNOPQRSTUVWXYBCDEFGHIJKLMA	p_{25}

Table 56.1: Tableau for the Porta cipher. Plaintext letters appear across the top, key letters on the left, and ciphertext letters in the bulk of the table. The two versions are American ("1") and European ("2").

There is a way to generate the tableau by rotating the second half of the alphabet against the first half. For p_{13} there is no rotation:

```
ABCDEFGHIJKLM
↕↕↕↕↕↕↕↕↕↕↕↕↕
NOPQRSTUVWXYZ
```

For p_n, the lower row is rotated leftward by $n-13$ steps. Here, for example is p_{21} with rotation by eight:

```
ABCDEFGHIJKLM
↕↕↕↕↕↕↕↕↕↕↕↕↕
VWXYZNOPQRSTU
```

Let's work through an example of encipherment with each version. Here is a short message, which we encipher with the keyword `PORTA`.

plaintext:	`GIOVANNI DELLA PORTA PUBLISHED IN FIFTEEN SIXTY-THREE`
key letters:	`PORTAPOR TAPOR TAPOR TAPORTAPO RT APORTAP ORTAP ORTAP`
version 1:	`NPGMNGGQ ZRSSV GBKMV GHVSQJUYX QE SPZLNRG LQBGE MPIRY`
version 2:	`ZOJENHHN URRRS LBLAS LHURNBUXW NJ SOYBVRH MNGGF AZARX`

Reading and references

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; pages 119-121.

Practical Cryptography, practicalcryptography.com/ciphers/porta-cipher

Paolo Bonavoglia, "Trithemius, Bellaso, Vigenère: Origins of the Polyalphabetic Ciphers," Proceedings of the 3rd International Conference on Historical Cryptology, 2020, ep.liu.se/ecp/171/007/ecp2020_171_007.pdf, DOI: 10.3384/ecp2020171007

Augusto Buonafalce, "Bellaso's Reciprocal Ciphers," *Cryptologia* 30:1 (2006) 39-51, DOI: 10.1080/01611190500383581

Giambattista della Porta [Giovanni Battista della Porta] [Ioan. Baptista Porta], *De Furtivis Literarum Notis*, Naples [Neapoli]: Ioa. Maria Scotus, 1563, HDL: 2027/gri.ark:/13960/t37142x6g, book 2 chapter XVI.

Programming tasks

1. Implement the Porta cipher. Allow for the possibility of choosing the version of the tableau. You can hard-code the tableau into your code, or you can find a way to generate key alphabets algorithmically or with shifts, or you can use the magic of modular arithmetic.

2. Copy and make the appropriate changes to your brute-force attack on Fuer GOD so that you can aattack a ciphertext that was encrypted with the Porta cipher. Note that you only need to consider a subset of the alphabet when generating keywords.

3. Add a function to search your word list for a match to a keyword. For example, if your brute-force attack finds the keyword `SECQES`, then `SECRET` matches because it gives the same set of key alphabets.

4. Copy and modify one of your previous dictionary attacks so that you can attack a ciphertext encrypted with the Porta cipher. Remember that there are two versions, so for each word you will need to check two decryptions.

5. Copy and modify one of your previous hill-climbing attacks so that you can attack a ciphertext encrypted with the Porta cipher.

6. Copy and modify one of your previous attacks that uses monogram frequencies to break a polyalphabetic cipher so that you can attack the Porta cipher.

Exercises

1. Encipher this text in version one with the keyword `KEYWORD`.

   The Doors were a rock band from Los Angeles. The name was taken from the title of Huxley's book The Doors of Perception, perhaps because the band enjoyed hallucinogenic drugs.

2. Decipher this text in version two with the keyword `CIPHER`.

```
KNRVTOIJAQDBUEAEIJKETYDCHWHRDCFWZEIJCTTVDJWWLEJIPJXOLZUYNMY
GHQXMTIPXRELAKQXHPBTNRSIBXNZSGQGRHJSWUINEWQMAIEOTIEZXWJKJTH
PGCLTVJUWCRSIBXNRXOGUEARPUUKFEQQUPNJYMQCIKYQIYQBWNHKROWSPCO
DIZQYOJAJRUOQSBCIPKHBZEIJSWFUIIISBPXVTIHQXZTBMFYDTYTKRXLJYW
H
```

3. Brute-force this ciphertext and find a short English word that could be the keyword.

```
RGKYIPFKZHFKJONSGYELLKNIIPMUYEIYZAYYEHRLNUEXYUEUULFORXNKFHV
LFKWKNUJMQUKUIYLGKPZERGKYIPFKZHFKJWFTNPEUAPZYMUIPNMCHOLKEUY
JCQPYOVUTYJPKYWLCMFTWMYOFKYHVIUYVYEMWGNDRLKPENKKRTWGZZRGRLQ
PEEFAIOFTNLZYYHILGYYPOPYKFHVLJAYOWLXUKOIHFTJVNXIHFTJUEXYSFL
NMJKNJLPIYZPOZNKNGKXFHIZLGYMRHEUUPKEJHXYJAIYKHYHELRXNKAONKN
EFAAPUSXYRGJMWSUPENCHLKEYAXFHIUEXRMJUGINUIUEWNVNZFKNTWRRGPE
FAIILKYOWLNIWGNSZHFKJUULFREHAGWLJSWVZHFKJWFGKURGGUKMNKELFZJ
JLUIYFKFVUHENGUEYULWGZUIYWWUUJLRWYOFPYYWXFHITWEYHEMWPEMAHUU
INNIWGNSJLRDJTWSUHEYJHILFTNHKONKYHEZRNLKWMRHEWQHFLNIWGNSJCR
MQLKKWPPOKHIWLKMYZMFIJMFLLPKEFAIXNWFKWMRGPGNYZLWWYHIXRHEXFH
ILWKNPZYWSOHIWUHJYKLWGZYEMIEAUCLAPKOJIWWNWFGJMIURGKLKONEVUC
WFGJPJMFZWZNCUUINNIWGNSJHITLSKPGSNLVUUSGUEYULKOWMOHUXRGFGKO
NTJYUBNLWGZKNLKUPURGJMFGNLRXNHOMQYZHFKFINGRGP
```

4. Perform a dictionary attack on this ciphertext.

```
SOOKZSUIFAEPAXWXAMUIGCWMJYFAQNKMNCENRBEUDJAIFQEARHKAWTBMKKD
PXXIKAOBTEUAEYXIBNPPRNAEARVNBWLHILAVMQXYYAMBXXZTMPWZKACBAQR
FUKRZSOROAFKMTBOZVAUCNNKBTJOEXACSOOKZSUIFAEEYTKWANUUWTYEKUF
CVDZWNBAUCNNKBTJOEXPPAHZKACBIOWSCRVUREDZVIDEEZVUKAZCNNHVPJZ
```

```
OAAORAFLDPZRWPZMFOEXFWVSKKENZWNKHTAWFKDMLXEZMVBDFKRTJIIQWCA
IIVRDPOZZMVUIFAEPAXWXAMUVQKWRVYFAQNKMNCEEYXCDETBJWTSTTIEKNX
TTUPKPJWVZDCKLNURMKLJUASKBXPSSDACPOCXXVZDCKLJRZDKSWOSPAIFQO
ALXWANMAXJLORZILB
```

5. Perform the hill-climbing attack on this ciphertext and find an English word that could be the keyword.

```
TQWUPJPGIAHRFZUVNKYQHXXKFZJKEDXVLZCYCUXBIYQDJADEUYXROUQYMVN
DFQAHYBKZRPYBZLIDNYCNDYTQGDAKYACGSCSVMMMDYPXDARVVHPXWDAYGYR
CKILXAZKCSVRHVHPTCHPUYXBDAFGARIVGLIAPEQVRRSZPYXJGKCUXJIAZFV
VSKKZCVGCTZMVOKCJMTYJNZVAPRHZWTJBHGSZBVPLIVCTCUJRHBYGLVPPZW
IAYCQYXBTXBK
```

6. Attack this ciphertext with your code that treats the Porta cipher as a set of monoalphabetic substitutions. Find an English word that could be the keyword.

```
IBPTSPAHRUDMEEESWPCGVONLETNDXATPPARNHAYQVZSGHDUFAVVULMBRXGR
XMESXSWDCQOYYVECVQOGYPJNDORJRXASOEVOHVFRNBDNVRMBKOUVPOETDCR
TEERSTNHBSVPNGVXGBNNBEOECDEUOJBWYDBZSQFDMQREBAIWREOZDQUYSIB
DCSOSJGVGEUODUKWNBIZBAPSBAHF
```

Unit 57
Bellaso 1552 cipher

The modern versions of the Porta cipher are descendants of ciphers invented by Bellaso. Recently, his first cipher (from 1552) was discovered in Venice, Italy. Here was its tableau:

key letter	**plaintext alphabet** ABCDEFGHILMNOPQRSTUXYZ
A	NOPQRSTUXYZABCDEFGHILM
E	ZNOPQRSTUXYBCDEFGHILMA
I	YZNOPQRSTUXCDEFGHILMAB
O	XYZNOPQRSTUDEFGHILMABC
U	UXYZNOPQRSTEFGHILMABCD
B	TUXYZNOPQRSFGHILMABCDE
C	STUXYZNOPQRGHILMABCDEF
D	RSTUXYZNOPQHILMABCDEFG
F	QRSTUXYZNOPILMABCDEFGH
G	PQRSTUXYZNOLMABCDEFGHI
H	OPQRSTUXYZNMABCDEFGHIL
L	MLIHGFEDCBAZYXUTSRQPON
M	AMLIHGFEDCBYXUTSRQPONZ
N	BAMLIHGFEDCXUTSRQPONZY
P	CBAMLIHGFEDUTSRQPONZYX
Q	DCBAMLIHGFETSRQPONZYXU
R	EDCBAMLIHGFSRQPONZYXUT
S	FEDCBAMLIHGRQPONZYXUTS
T	GFEDCBAMLIHQPONZYXUTSR
X	HGFEDCBAMLIPONZYXUTSRQ
Y	IHGFEDCBAMLONZYXUTSRQP
Z	LIHGFEDCBAMNZYXUTSRQPO

Table 57.1: Tableau for the original Bellaso 1552 cipher. Plaintext letters appear across the top, key letters on the left, and ciphertext letters in the bulk of the table. The top half contains the same permutations as his version of the Porta cipher.

He used a 22-letter alphabet, which was all the rage in Italy at the time. Notice that all of the key alphabets are reciprocal, and so the cipher itself is also reciprocal. Notice also that there is only one key letter assigned to each alphabet, so that keywords are unambiguous, unlike the Porta cipher.

We can modernize the *Bellaso 1552 cipher* by using the 26-letter English alphabet and putting the key letters into standard order to get the tableau in Table 57.2.

key letter	**plaintext alphabet** ABCDEFGHIJKLMNOPQRSTUVWXYZ	**label for permutation**
A	AMLKJIHGFEDCBYXWVUTSRQPONZ	p_0
B	BAMLKJIHGFEDCXWVUTSRQPONZY	p_1
C	CBAMLKJIHGFEDWVUTSRQPONZYX	p_2
D	DCBAMLKJIHGFEVUTSRQPONZYXW	p_3
E	EDCBAMLKJIHGFUTSRQPONZYXWV	p_4
F	FEDCBAMLKJIHGTSRQPONZYXWVU	p_5
G	GFEDCBAMLKJIHSRQPONZYXWVUT	p_6
H	HGFEDCBAMLKJIRQPONZYXWVUTS	p_7
I	IHGFEDCBAMLKJQPONZYXWVUTSR	p_8
J	JIHGFEDCBAMLKPONZYXWVUTSRQ	p_9
K	KJIHGFEDCBAMLONZYXWVUTSRQP	p_{10}
L	LKJIHGFEDCBAMNZYXWVUTSRQPO	p_{11}
M	MLKJIHGFEDCBAZYXWVUTSRQPON	p_{12}
N	NOPQRSTUVWXYZABCDEFGHIJKLM	p_{13}
O	OPQRSTUVWXYZNMABCDEFGHIJKL	p_{14}
P	PQRSTUVWXYZNOLMABCDEFGHIJK	p_{15}
Q	QRSTUVWXYZNOPKLMABCDEFGHIJ	p_{16}
R	RSTUVWXYZNOPQJKLMABCDEFGHI	p_{17}
S	STUVWXYZNOPQRIJKLMABCDEFGH	p_{18}
T	TUVWXYZNOPQRSHIJKLMABCDEFG	p_{19}
U	UVWXYZNOPQRSTGHIJKLMABCDEF	p_{20}
V	VWXYZNOPQRSTUFGHIJKLMABCDE	p_{21}
W	WXYZNOPQRSTUVEFGHIJKLMABCD	p_{22}
X	XYZNOPQRSTUVWDEFGHIJKLMABC	p_{23}
Y	YZNOPQRSTUVWXCDEFGHIJKLMAB	p_{24}
Z	ZNOPQRSTUVWXYBCDEFGHIJKLMA	p_{25}

Table 57.2: Tableau for the modernized Bellaso 1552 cipher. Plaintext letters appear across the top, key letters on the left, and ciphertext letters in the bulk of the table. The order of the rows is chosen so that the first letter of each permutation matches its key letter.

Here is a short example of the encipherment of a message with this cipher. The keyword is `PLAGIA`.

```
plaintext:     BELLASO BEAT YOU TO IT BY ELEVEN YEARS
key letters:   PLAGIAP LAGI APL AG IA PL AGIAPL AGIAP
ciphertext:    QHCIITM KJGX NMT SR AS QP JIEQTN NCIUD
```

When we look at the tableau we see that, of course, p_{13} through p_{25} are the Porta permutations. We saw that they have a pattern of rotating the second half of the alphabet against the first half. The new permutations are rotations of the reversed first half against the forward first half and rotations of the reversed second half against the forward second half. Start with p_{12}, which is

```
ABCDEFGHIJKLM     NOPQRSTUVWXYZ
↕↕↕↕↕↕↕↕↕↕↕↕↕     ↕↕↕↕↕↕↕↕↕↕↕↕↕
MLKJIHGFEDCBA     ZYXWVUTSRQPON
```

Rotate the left half of the lower row leftward by $12-n$ steps, and the right half rightward by $12-n$ steps, to obtain p_n when n is less than 12.

Remember the z we defined for the atbash cipher (Equation 16.1)?

$$z = \texttt{ZYXWVUTSRQPONMLKJIHGFEDCBA}$$

Well, we can define another permutation that in some sense encapsulates the essence of the Porta and Bellaso ciphers.

$$x = \texttt{AYCWEUGSIQKOMZBXDVFTHRJPLN} \tag{57.1}$$

With this permutation we can magically relate the set $\{p_n\}$ of Porta/Bellaso permutations to the set $\{R_n\}$ of rotations for Caesar/Vigenère and to the set $\{b_n\}$ for Beaufort:

$$p_m = x \circ z \circ R_n \circ x^{-1} = x \circ (z \circ R_n) \circ x^{-1} = x \circ b_n \circ x^{-1} \tag{57.2}$$

We have written p_m with an m subscript on the left-hand side because the permutation that we get when we evaluate the right-hand side is not in the nth row of the tableau, and the relationship between m and n is not easy to write down.

$$m = \begin{cases} n+13 & \text{if } n \text{ is even}, <13 \\ n & \text{if } n \text{ is even}, >13 \\ 12-n & \text{if } n \text{ is odd}, <13 \\ 25-n & \text{if } n \text{ is odd}, \geq 13 \end{cases}$$

Reading and references

Paolo Bonavoglia, “Trithemius, Bellaso, Vigenère: Origins of the Polyalphabetic Ciphers,” Proceedings of the 3rd International Conference on Historical Cryptology, 2020, ep.liu.se/ecp/171/007/ecp2020_171_007.pdf, DOI: 10.3384/ecp2020171007

Paolo Bonavoglia, “Bellaso’s 1552 cipher recovered in Venice,” *Cryptologia* 43:6 (2019) 459-465, DOI: 10.1080/01611194.2019.1596181

Augusto Buonafalce, “Bellaso’s Reciprocal Ciphers,” *Cryptologia* 30:1 (2006) 39-51, DOI: 10.1080/01611190500383581

Giambattista della Porta [Giovanni Battista della Porta] [Ioan. Baptista Porta], *De Furtivis Literarum Notis*, Naples [Neapoli]: Ioa. Maria Scotus, 1563, HDL: 2027/gri.ark:/13960/t37142x6g, book 2 chapter XVI.

Programming tasks

1. Implement the modernized Bellaso 1552 cipher. Note that the tableau can be hard-coded into your script or the permutations can be generated on demand.

2. Copy a previous brute-force attack and modify it to attack the modernized Bellaso 1552 cipher.

3. Copy a previous dictionary attack and modify it to attack the modernized Bellaso 1552 cipher.

4. Copy a previous hill-climbing attack and modify it to attack the modernized Bellaso 1552 cipher.

5. Copy a previous attack on a polyalphabetic cipher that uses monogram frequencies and modify it to attack the modernized Bellaso 1552 cipher. If you were clever in your script for the Fuer GOD cipher, then this task should be as easy as swapping tableaux.

Exercises

1. Decipher this ciphertext with the modernized Bellaso 1552 cipher and keyword `PLAGIARIZE`.

```
HHCIXGVDUQDUHOEACJZUHEXCVJAUFTEHBCICVXHACRANPRACIADUXIAQVZK
ALIJIKGKKYADEJWITRKGTEEJBAUBXSQTLSIASVZZQJMASAJEEFPEZCCIYHX
TJLFIOPBRQPOWLSLYGKUUCPMJZPPAAHAEZGLJAJFTAEZBCUGVQYWUDUNXMK
PWYPVYCUAWZUALIXBJFJEGEXISRJJCBQBTIFEISZPBJDSJOSYVIHWTVFNIF
UAHKMTHMXFCUZPOPIOAJJFGEXIFGKPRSGESMFOEKZXQZTNMCDXAEUPPRFZA
YCBQJLNXEEYCYZDCZADAYRXIQPACUYAZFKKPUKRSXDJQELRGCZJUAPWMTJX
EUBEQJEKJBPUVUQGNDTGUFCDUQDUTRJJHEZQDLHRITUPOOMWGRKBVYGBTIF
EISZPBOMEFNYXJCGJLMASSDVSG
```

2. Break this ciphertext that was encrypted with the modernized Bellaso 1552 cipher. What is the keyword?

```
CTCTLYWVTOTWJVCZNPMBLIKSPNAMHMESLYISFWDKCIWRMCCYWBSXCJAHZWM
JRSIJSEIVALCWNLCNMQOETVYAVAZLIAAISESVRZIFHDPCWBSPHIXLQQFIEX
MQVXHNAFCMSQVRHMVXXRJRKWLYWJANACVVISLOLPOFREAMVHYZRFCPLSMKS
TWHYZSOESLUHLDGDPTNLPTRCYAVIPWFJEQDTXLCFKODINGXWEFOMJHEKMJQ
XYCMPGAGDCKCYWAVIWNOSFZIFQPQPXLMWTRCXJEXIKJREALIIJJCHTXLYAI
SEGHWJLRMKLUOQOLOQUTDBNPMRJVMXIBJOWUMQAXBSETKLUFDGRSKUOJVEP
WJWCMSYEFAQZTSPUAHFQPJAHQADNSJTDOLOHYTKSHHLLIWRDIIJDIANPRSL
LSELCTYPZNDKUIPUWELXJESHSVLTDSADPUOWUMJQXYQVTSVRMLENTFJTWFD
IIJFRGCGVRHMSSSHBTWRPHFJWIWOSHTHCNTSLPTRXFHQAUVGEIVHOOSXING
XWVTHPKWLMNASJSLNQPSJWQVTDUJHHMVTWLOMSLIHYSSEEPOJHRALHRSISK
WPNAMRAUTECXMIXIIGISDMGCSNCHXCJUXCZXHLKOWVXHVPHUSCHVYAIOEIL
ULHPNPCYWNSNWNUOQFTCRLXWSPCMAAIOGTWHWVMTYALOMSLICCRYFWDPHNH
MUYASDIOEIIJSSVJOLXYKWRIHXHIOSOVYXNWESMJRHPTGCSXHMEPWSAIOES
LUTWOHYYAMIOCOCTNLQMJEKZVVTLCTNVITKLUCLDJNEXMLZJQVCRVITOCYW
KWLYECFISEDVDLLTFRLFHCEFGJJLYFWPMXISAULIFVIVPEUJHLREDBJIUSL
LIJLLVTPZWRCWLPMSICKMJWHTREXHQAHQIWDIAJUSLPMSIBSESLUXLJPNQC
FNVPIIAWVQRJSUXMFWPIRSCMXICUOCMRGVPWNSCPHYWPQTOQXZVOHYYXHIW
DRHQOWUESLBTRNPWETBDAUXVYXYOLOCTMVALYRSNLQMYRSDCERWLALISNPK
STRVTECTMLRMJQSNLMPXLXHQVTZSAMRRMDSWRCFTDKSNLEFGJJLYQPJJGAI
WDPYJOQIPHGJQYZNYQCXCEETQTNVGNOIACLPTPW
```

3. Verify Equation 57.2 by checking that you get a Porta/Bellaso permutation when you compute the right-hand side, for all twenty-six values of *n*, and that you get them all.

Unit 58
Periodic affine cipher

Suppose we construct a periodic polyalphabetic substitution cipher in which the key alphabets are generated as if for an affine cipher. Then we have a *periodic affine cipher*. The key for such a cipher is a set of pairs of integers. This takes a step away from the highly constrained Vigenère, Beaufort, variant Beaufort, Porta, and Bellaso ciphers, as we now have 312 possibilities for the alphabetic permutation used to encipher each slice. Soon, however, we will see ciphers with more possibilities.

For example, suppose we want to encipher a message with a period of three, and want to use these affine keys (multipliers and shifts): 5, 8; 11, 2; and 21, 18. The key alphabets are these:

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	INSXCHMRWBGLQVAFKPUZEJOTYD
1	CNYJUFQBMXITEPALWHSDOZKVGR
2	SNIDYTOJEZUPKFAVQLGBWRMHCX

And here we encipher a short message:

```
ANY MONOALPHABETIC SUBSTITUTION CIPHER CAN BE USED PERIODICALLY
012 01201201201201 201201201201 201201 201 20 1201 201201201201
IPC QAFACPFBSNUBWY GENGZMBEDEAP IWLJCH IIP NC OGCJ VCHEAJESCPLG
```

When the multipliers of all of the affine ciphers are the same, then we have a special case in which the cipher can be factored into a single affine cipher followed by a Vigenère cipher. Decipherment is in the opposite order. There are 26 choices for the affine cipher, for the 26 choices of the shift. For each of those choices there is one Vigenère key that makes the combination of ciphers equivalent to the original periodic affine cipher.

Programming tasks

1. Implement the periodic affine cipher.

2. Devise a crib-based attack on the periodic affine cipher. You will need two plaintext-ciphertext letter pairs to give you two modular equations to solve for each multiplier and shift pair (see Unit 32). Use tetragram fitness to check your decryption for each consistent placement of the crib.

Exercises

1. Encipher this text with key 3, 4; 5, 6; 7, 8; 9, 10; 11, 12; 25, 24.

```
AN AFFINE PLANE IS A SET OF POINTS AND A SET OF LINES SUCH THAT
ANY TWO POINTS LIE ON EXACTLY ONE LINE AND GIVEN A LINE AND A
POINT NOT ON IT THERE IS EXACTLY ONE OTHER LINE THROUGH THAT
POINT THAT DOES NOT INTERSECT THE FIRST LINE
```

2. Decipher this ciphertext with key 11, 9; 9, 8; 7, 6; 5, 4; 3, 2.

```
HTEWPQSVNSRMGRWXEATJNAVYCKGVYEJFIVXBFINOOSDWQUSGAHBEQQMJVYS
PQCCWFOIXYPBYEWBAJJNCKTGQEVADBOHNFYAWCJ
```

3. Break this ciphertext with the crib `STORIES`. The period is 3.

```
NSSNKENSIKLSVXFIUHCXINSSHCSNYSHPFWPFKXNDVPNSSKYGHZSKYRIYAUY
CLPENZPNSIKJCKXINZWKPYUTRIYTUYNUKNDPEASIIBPYYXHPERPJGY
```

Unit 59
Attacking the periodic affine cipher with monogram frequencies

In Unit 52 we built an attack on the Vigenère cipher by partitioning the ciphertext into slices, each of which was encrypted with one Caesar shift cipher. We then used the attack from Unit 24 to break each of the Caesar ciphers by using monogram frequencies. Here, we will do the analogous thing and partition the ciphertext, but use the technique from Unit 33 to break each affine cipher with monogram frequencies.

Programming tasks

1. Implement the attack.

Exercises

1. Break this ciphertext:

```
EMNMGYUQNIXTEMNMGYUQNVGUKOYJUZKRKCINNCEKZLGLGXMEUJXKFEUKYJU
NJEXKRETSGEMKQNLUUPGQXCVOAXYCHDKJXHUPGQXMAKPNAXALQSUHFVWJYV
SKWDSFGEUETDQNQKEMXAUTYXSKWRYGLECDSMOCWDAMQLCRXPUMDLAMQLCLJ
PUMIDAIZTGEJLGKHENCDPVUXXGYDLAOSCGLFITKWJWCIQQMOSQNOETIMEVZ
APUCVSGUSKWKGVTYVCOSQNDQEMXTVUXFUOQDDKHLCXSLVEASYVZUCLELVEY
EWMAMPJXCMKNQXOUCGQUHFAKPALVEOETKAVCCNCDXQUXEUJGDWWVRKGXOKC
VZAUOYUJKCTQVZAWKRTCAXQLGJPCIAJJGVLLQSFSXZACXHFCGQOEUZFCTYP
RGGAVZCVUYVUCQSTRUEVETDVKYUVDZLGNUXUIVCYVQUKLJHGKM
```

2. Break this ciphertext from the 2017 British National Cipher Challenge. You will notice something about the resulting key. Can you factor this cipher (i.e., split it into two simpler ciphers)? Can you find a meaningful keyword (it won't be in English).

```
YKUFRQHUDDQRGZPPISMXOOEYZUDOGMPPRLZQAERLOPNLOVKTVFNXOYZCQDZ
OVDVECXUAIBKVGLBEJRVPSLARHDTWOBNIMLWDFGGTAUQGONBLRFLNMGMDRP
YBVIGKXVXDBOQXUGBUVOKLQLPGHROVAMGIUKRDPHDOQEOEIGIUKXIVDLYJL
ZIPEGELWVFDBMDBSGZKCGSYSOOOIOEOBOPTEMMKHCROCVNLWDJOKSIGWCOD
TJGZCVLSYOYVTOURIWMHFDLZQKUAIIAWBOOAMKHDNGEQBTOEICUGKFGYUCZ
IESQPUGAKANFMELCLZTMVINRRIRDBOBVGHJBQYHGCKMSJILPIZQJLXEELPZ
JSNOYCQIMJMDHKIPIAWBOAOBMBOLWESOGLYVOABXCENDBOQXUGBUVNJCDNV
BQUUQGVAJTCLIOXGBJVPUBAUTDSQYCTMNNTRHDAUFEKVPFBSNVYVWAXKZQJ
AAMMRIZOVJGMMUMXELJTHYBAJVCPKLWHMRBVCMWIGKQRNVLYROVAONGUKWB
OXGGDMVTGJJOKEUMCTGDYDHPAYTVGNKAUBSFNDMYGNEEBODYLSIVMWEIUHO
IYXIIXRZILMINLCRNYBODTQGKACUKPJNVLYROVGSBIGCRKFYRNLDPIAEGEE
```

ZSHVGMROREIAWGMEFFKAILLHWPUMTIPJLBEQBADDORIELFILZDJSNCTQCTM
EGRJAAVQQWEIAVTIBNFWAOFTGLYDDLEERFYBGUOVCMWTLLDQAWYZOQSQENA
PVPYRHGBKXNTOUXTXWXOXGNISDDMOIRVXRICLWKDZEIWINWCUNDYJSNQYWW
IRVXSYBOTNVHFLAUTECBGYBODTKGBSTLLYIUDELGTIAKCVTIYKNGLLQMQII
NGXKAOHURMNQSINWTLXXSGAAQBHSGIUQQCPLNATQBHNNJECMKCRIPGXIVHG
MDJOKPQJWYBXLIUODKUQMSZSDYROXAIMMSIFLXYEISEHTOACTDJAOXGKSMV
ANPIRDRYRUOBROTHJCDRDJQNNTOULKGHHSSCGKFNYZROVGSBIGCRKZUKSOE
OZGGKUMCBKUAPBLWISCKEIDPFERPSEJHJCGXPGRPUOIKRSOIDKBEBYZKDLY
NTAPXIEKAEFRGBBLIYVDDTJRICERAQOVCMWUOIRENVCQROPGAKVVVSRICNB
OHXDOBIRWEMPDBAOQVZGNKRIPJICEHQMWBOPIUCQIYJTGBUHELQGFSYSOYE
IIESQPXIELZOODBONLPAGJJOKQIGCJKONIGDCGQKXKLZOIRVDAKPAUKWXIE
AILLHWMDJOKPQJWIZOQVHPCVUCQOOUZMRTOHJCPVRFLGAQTQANTODDTACNN
NZIIAZEYETUFEANATGXUQGAUTNSBKIEFIIAZEBOQNIRDNOIZYNHDOGIEEGS
LNZLYFXOIHGTPGYHXYZIGTHWWETIRQNFFBQNPDYODRXSQGHQGYGZYIDMBVG
XPOGZRRGZCPIRLEUMCTGDMNKQDKRVFIHCSWDTUERPAXIPOPNHJQENTPXIGV
USYQZGUNTRHMIUUWYJPIPQUNKOUPUFENFWAGXUQGAUBTNWCZKDPIGDKYUQG
HEOGCTMEGXSGZGOIYOKPGMJDUEQVJOKDDPOGGXNDSJAFLEYVVDIVALDZGNA
UXECFRGBROVSGVIWPEXFPGPLTUEIWKEYWXOXGYQRIHJCTDCMULMLOKBIGPX
GUUWYJEDLZLXRJEMDCMRNZDSQUQEPETUKBNPQGZCTCTGGGFIDRVPGHKSFAW
QWTEFKOXEPROVHBAKUCRCLZUMCBPBMWJOKQIGCJMBMXASACNMPEIEAIYSUY
BDIGEGBOXGZIGWBKXKQKFLNHEBVRIIMNCUFEUHCDDQNTTOKTQMUQKMHGURL
GSKUOCLRICSGJTYSMZWRWMKPWEZQMDSMDIMXWULNEZJOVKWWJYKYRFBFOJY
VNGPIOKQOYVGIRGNQPXINKQYAEHYIAWBOEMSKRSLZSQPOVTIDDTWWJNFCQN
QAGBCULZEDNVYBALDQGKVAIISKMEZJTKGBMAEKOIVSRGMPXDLBGDLZQMPIA
QRGZIUJLLZQHCVOBUQFMPJLRFLNDTWQNWYBMAAISDPOYLOVGCJEUOLZQUAH
OIAWBONKELZNNQIGBRGZGJJOKQIGCJQRNVDVYKQHGBOEKOVTIGDCGQKXYYD
ILROVRQDEVHBVNQOMDMVZTMJOKCBZEGZRYTLYZAEMCBBUZHQNPYVYGG

Unit 60
Quagmire ciphers

Now we take another step away from the simplicity of the Vigenère cipher toward more complicated and more difficult ciphers. *Quamire ciphers* are usually presented in one of two ways. One is to slide one alphabet against another, with a periodically changing offset, which is given as a keyword whose letters in the lower alphabet are placed under one prespecified letter in the upper alphabet. Those two alphabets may or may not be permuted.

For example, we can take these alphabetic permutations

```
QUAGMIREBCDFHJKLNOPSTVWXYZ
CIPHERZYXWVUTSQONMLKJGFDBA
```

and choose the indexing letter to be A, and shift key ENCRYPT. Here, too, is a plaintext:

```
SECRET MESSAGE FOR YOUR EYES ONLY
```

We can repeat the shift key under the plaintext for convenience:

```
SECRET MESSAGE FOR YOUR EYES ONLY
ENCRYP TENCRYP TEN CRYP TENC RYPT
```

To encipher the first letter, S, we shift the alphabets along each other until the first letter of the shift key, E, lies under the indexing letter, A. The letter under S is the first ciphertext letter, G.

```
  QUAGMIREBCDFHJKLNOPSTVWXYZ
CIPHERZYXWVUTSQONMLKJGFDBACIPHERZYXWVUTSQONMLKJGFDBA
```

To encipher the second letter, E, we shift until N lies under A, and the letter under E is another G:

```
                QUAGMIREBCDFHJKLNOPSTVWXYZ
CIPHERZYXWVUTSQONMLKJGFDBACIPHERZYXWVUTSQONMLKJGFDBA
```

The full encipherment is

```
SECRET MESSAGE FOR YOUR EYES ONLY
ENCRYP TENCRYP TEN CRYP TENC RYPT
GGYWTJ QWYMRXY GKJ FJZZ MCGM JGOX
```

The second way that quagmires are usually explained is with a tableau. For the same example alphabets above, we make a tableau by shifting the second alphabet leftward one more step for each successive line in the tableau.

key letter	plaintext alphabet QUAGMIREBCDFHJKLNOPSTVWXYZ
C	CIPHERZYXWVUTSQONMLKJGFDBA
I	IPHERZYXWVUTSQONMLKJGFDBAC
P	PHERZYXWVUTSQONMLKJGFDBACI
H	HERZYXWVUTSQONMLKJGFDBACIP
E	ERZYXWVUTSQONMLKJGFDBACIPH
R	RZYXWVUTSQONMLKJGFDBACIPHE
Z	ZYXWVUTSQONMLKJGFDBACIPHER
Y	YXWVUTSQONMLKJGFDBACIPHERZ
X	XWVUTSQONMLKJGFDBACIPHERZY
W	WVUTSQONMLKJGFDBACIPHERZYX
V	VUTSQONMLKJGFDBACIPHERZYXW
U	UTSQONMLKJGFDBACIPHERZYXWV
T	TSQONMLKJGFDBACIPHERZYXWVU
S	SQONMLKJGFDBACIPHERZYXWVUT
Q	QONMLKJGFDBACIPHERZYXWVUTS
O	ONMLKJGFDBACIPHERZYXWVUTSQ
N	NMLKJGFDBACIPHERZYXWVUTSQO
M	MLKJGFDBACIPHERZYXWVUTSQON
L	LKJGFDBACIPHERZYXWVUTSQONM
K	KJGFDBACIPHERZYXWVUTSQONML
J	JGFDBACIPHERZYXWVUTSQONMLK
G	GFDBACIPHERZYXWVUTSQONMLKJ
F	FDBACIPHERZYXWVUTSQONMLKJG
D	DBACIPHERZYXWVUTSQONMLKJGF
B	BACIPHERZYXWVUTSQONMLKJGFD
A	ACIPHERZYXWVUTSQONMLKJGFDB

Table 60.1: Tableau for the example quagmire cipher. Plaintext letters appear across the top, key letters on the left, and ciphertext letters in the bulk of the table.

To encipher the first letter, S, with key letter E, we look for the letter in row E and column S; it is D. We can skip to the end and show the complete ciphertext:

```
SECRET MESSAGE FOR YOUR EYES ONLY
ENCRYP TENCRYP TEN CRYP TENC RYPT
DDWUQF NUWKYVW DGF BFXX KPDK FDMV
```

You have certainly noticed that the two methods of performing the cipher gave us two different ciphertexts. The reason for this is that the tableau method always uses the first letter of the upper alphabet as the indexing letter. If we shift each letter in our first ciphertext two steps to the right in the

lower alphabet, we get the second ciphertext. We should remember this discrepancy as we work through the next few units. The default procedure is to index on the first letter of the first alphabet, and we will do so from now on. We can convert the ciphertext indexed on A to the one indexed on Q by performing a “Caesar salad” shift, as in the cipher of Unit 25:

$$c_{\mathtt{Q}} \;=\; S(k_3)\; C(2)\; S^{-1}(k_3)\; c_{\mathtt{A}}$$

where $C(2)$ is the operator that performs a Caesar shift by two, and S is the operator for a monoalphabetic substitution cipher.

Suppose that we want to specify an encryption operator for a quagmire cipher. It has three keys: the two alphabets and the shift keyword. Our convention is to call the upper alphabet k_1, the lower alphabet k_3, and the shift key k_2. This may seem awkward, but we will see that the shift key, in a very real sense, belongs between the two alphabet keys. Now take a look back at the tableau in Table 60.1, and consider the plaintext across the top. To make this more like a Vigenère tableau, we have to substitute back to the unmixed alphabet, like this:

key letter	QUAGMIREBCDFHJKLNOPSTVWXYZ ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓ ABCDEFGHIJKLMNOPQRSTUVWXYZ
C	CIPHERZYXWVUTSQONMLKJGFDBA
I	IPHERZYXWVUTSQONMLKJGFDBAC
P	PHERZYXWVUTSQONMLKJGFDBACI
⋮	⋮

This is the opposite direction of a substitution cipher, so this tells us that the first stage in the cipher is actually an inverse monoalphabetic substitution. Now the ciphertext letters in the bulk of the tableau look like those in the Vigenère tableau, except that they have undergone a substitution in the forward direction. So the last stage in a quagmire cipher must be a monoalphabetic substitution. If we factor those two substitutions out, the middle part of the cipher is a Vigenère (see Table 60.2 on the next page). In terms of the operators the factorization is,

$$Q(k_1, k_2, k_3) \;=\; S(k_3)\; V(k_2')\; S^{-1}(k_1) \tag{60.1}$$

(remember that the action goes from right to left). Notice that we put a prime mark on the k_2 inside the Vigenère. This is to indicate the difference in the way the shifts are chosen, using a straight alphabet in the Vigenère, but the permuted alphabet in the quagmire.. To get from k_2 to k_2' we convert it with k_3:

$$k_2' \;=\; S^{-1}(k_3)\; k_2 \tag{60.2}$$

key letter	plaintext alphabet QUAGMIREBCDFHJKLNOPSTVWXYZ ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓ ABCDEFGHIJKLMNOPQRSTUVWXYZ	and then:
C → A	ABCDEFGHIJKLMNOPQRSTUVWXYZ	A → C
I → B	BCDEFGHIJKLMNOPQRSTUVWXYZA	B → I
P → C	CDEFGHIJKLMNOPQRSTUVWXYZAB	C → P
H → D	DEFGHIJKLMNOPQRSTUVWXYZABC	D → H
E → E	EFGHIJKLMNOPQRSTUVWXYZABCD	E → E
R → F	FGHIJKLMNOPQRSTUVWXYZABCDE	F → R
Z → G	GHIJKLMNOPQRSTUVWXYZABCDEF	G → Z
Y → H	HIJKLMNOPQRSTUVWXYZABCDEFG	H → Y
X → I	IJKLMNOPQRSTUVWXYZABCDEFGH	I → X
W → J	JKLMNOPQRSTUVWXYZABCDEFGHI	J → W
V → K	KLMNOPQRSTUVWXYZABCDEFGHIJ	K → V
U → L	LMNOPQRSTUVWXYZABCDEFGHIJK	L → U
T → M	MNOPQRSTUVWXYZABCDEFGHIJKL	M → T
S → N	NOPQRSTUVWXYZABCDEFGHIJKLM	N → S
Q → O	OPQRSTUVWXYZABCDEFGHIJKLMN	O → Q
O → P	PQRSTUVWXYZABCDEFGHIJKLMNO	P → O
N → Q	QRSTUVWXYZABCDEFGHIJKLMNOP	Q → N
M → R	RSTUVWXYZABCDEFGHIJKLMNOPQ	R → M
L → S	STUVWXYZABCDEFGHIJKLMNOPQR	S → L
K → T	TUVWXYZABCDEFGHIJKLMNOPQRS	T → K
J → U	UVWXYZABCDEFGHIJKLMNOPQRST	U → J
G → V	VWXYZABCDEFGHIJKLMNOPQRSTU	V → G
F → W	WXYZABCDEFGHIJKLMNOPQRSTUV	W → F
D → X	XYZABCDEFGHIJKLMNOPQRSTUVW	X → D
B → Y	YZABCDEFGHIJKLMNOPQRSTUVWX	Y → B
A → Z	ZABCDEFGHIJKLMNOPQRSTUVWXY	Z → A

Table 60.2: The example quagmire cipher is factored into a Vigenère between two monoalphabetic substitutions.

key letter	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
A	IYXWZVPUETSQHONMARLKCJGFDB
B	CZYXRWIVHUTSPQONBEMLAKJGFD
C	PXWVYUHTRSQOENMLCZKJIGFDBA
D	ARZYEXCWPVUTISQODHNMBLKJGF
E	ZTSQUOYNWMLKXJGFEVDBRACIPH
F	BERZHYAXIWVUCTSQFPONDMLKJG
G	DHERPZBYCXWVAUTSGIQOFNMLKJ
H	RUTSVQZOXNMLYKJGHWFDEBACIP
I	HWVUXTESZQONRMLKIYJGPFDBAC
J	FPHEIRDZAYXWBVUTJCSQGONMLK
K	GIPHCEFRBZYXDWVUKATSJQONML
L	JCIPAHGEDRZYFXWVLBUTKSQONM
M	KACIBPJHFERZGYXWMDVULTSQON
N	LBACDIKPGHERJZYXNFWVMUTSQO
O	MDBAFCLIJPHEKRZYOGXWNVUTSQ
P	EVUTWSRQYONMZLKJPXGFHDBACI
Q	NFDBGAMCKIPHLERZQJYXOWVUTS
R	YSQOTNXMVLKJWGFDRUBAZCIPHE
S	OGFDJBNALCIPMHERSKZYQXWVUT
T	QJGFKDOBMACINPHETLRZSYXWVU
U	SKJGLFQDNBACOIPHUMERTZYXWV
V	TLKJMGSFODBAQCIPVNHEURZYXW
W	UMLKNJTGQFDBSACIWOPHVERZYX
X	VNMLOKUJSGFDTBACXQIPWHERZY
Y	WONMQLVKTJGFUDBAYSCIXPHERZ
Z	XQONSMWLUKJGVFDBZTACYIPHER

Table 60.3: Tableau for the example quagmire cipher. Plaintext letters appear across the top, key letters on the left, and ciphertext letters in the bulk of the table. In this version of the tableau, columns have been rearranged so that the plaintext alphabet is unscrambled. Rows have been ordered so that their key letters are also in alphabetical order.

There are two properties of the tableau for a quagmire cipher that are essential to know. Since every row is a permutation of the alphabet, it contains exactly one of each letter. What is less obvious is the fact that the same is true for columns: every column contains exactly one of each letter. For lack of a better name, we will call this the "*column property*," but a mathematician might call it "unitarity."

Quagmire ciphers come in several categories, depending on the alphabets used. Notice that quagmire 4 (Q4) encompasses all of them. One cipher, the Vigenère is a member of four categories: Q1, Q2, Q3, and Q4, but not the self-reciprocal category.

- $k_3 = e$: quagmire 1 (also known as *polyalphabetic cipher type 1*)
- $k_1 = e$: quagmire 2 (also known as *polyalphabetic cipher type 2*)
- $k_1 = k_3$: quagmire 3 (also known as *polyalphabetic cipher type 3*)
- $k_1 = k_3 = e$: Vigenère
- any k_1, k_3: quagmire 4 (also known as *polyalphabetic cipher type 4*)
- $k_3 = k_1 \circ z$: self-reciprocal (involutory) quagmire 4

See Figure 60.1 for a diagram of this classification.

Reading and references

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; chapter XVIII.

American Cryptogram Association
www.cryptogram.org/downloads/aca.info/ciphers/QuagmireI.pdf
www.cryptogram.org/downloads/aca.info/ciphers/QuagmireII.pdf
www.cryptogram.org/downloads/aca.info/ciphers/QuagmireIII.pdf
www.cryptogram.org/downloads/aca.info/ciphers/QuagmireIV.pdf

Programming tasks

1. Create a module (a separate Python file) for quagmire ciphers. In it, define a class for generic quagmire ciphers. Do this is three ways, as follows. Make sure not to call them "`quagmire1`" etc., to avoid conflict with the categories listed above; use names like "`quagmire_versionA`" etc. Use the default taking the indexing letter to be the first letter of the alphabetic key, but allow for the choice to change it. Use the default method of filling an alphabet with the remaining letters in alphabetical order, but allow the ability to choose another method. Import your class for alphabetic permutation and its ability to generate an alphabet from a keyword.

   a. slide two alphabets against one another

   b. build a tableau and use it

   c. execute the factors (inverse substitution, Vigenère, substitution); remember to convert the quagmire shift key to a Vigenère key as in Equation 60.2

   Make sure they all give the same results, and check some encipherments by hand. Then decide which implementation you prefer, perhaps based on speed of execution. Below your three class definitions, add a line like this to give your favorite an alias:

```
quagmire = quagmire_versionA
```

Figure 60.1: Classification of periodic polyalphabetic substitution ciphers. The category of quagmire 4 includes quagmires 1, 2, and 3. The Fuer GOD cipher is not a quagmire. The Vigenère cipher is simultaneously a quagmire 1, 2, 3, and 4. There is an overlap of quagmire 1 and 2 that we shall discuss later. There is one cipher in the overlap of quagmires 1, 2, and involutory quagmires that we shall also discuss later.

Unit 61
Attacking quagmire ciphers with cribs: traditional method

In Unit 43 we discussed in detail the use of cribs in attacking a periodic polyalphabetic cipher. In that case, we knew the tableau and all its permutations of the alphabet. However, with a quagmire cipher, we have very little a priori knowledge about the tableau. We have the property that each letter occurs exactly once in each row, and we have the column property that says that each letter occurs exactly once in each column. The column property tells us that if we find two rows in the key table that we are building during the attack that have the same letter in the same position, then they must be the same permutation and have all letters the same. This is the only constraint we have that is not present in a general periodic polyalphabetic cipher with random permutations in its tableau. (We will see more constraints later in this book.)

The traditional method for attacking a periodic substitution with a crib is to find the correct placement of the crib against the ciphertext, fill in what information we get from that placement into a key table, and then search for recognizable words outside the crib. By making a partial decryption with the incomplete information about the keys, we hope to see parts of words. By filling in missing letters, we gain more information about the keys, can decrypt more of the ciphertext, and recognize more words. The cycle continues until (we hope) we have fully decrypted the text.

Here is the requisite example. Consider this ciphertext:

```
NWHUSUWXSJVJXASDZSHIEHOZNRHUIDREAWGODLTEKWUSXVMWJTPTTZECTOZ
NISCJZYMTCEDSUYSGZIJCZUUWJGSHFTWEXPENNHBZVTWWKGUHVVDROQFYXT
UGISKSGJTADZYILZNEAXBTKMUYUVJXLNEHXSGTUUYWRNGBQGIKBJQNHBDID
RDIRJCQDRHNQJVXTZNWHODYNTWYVDTSVDREUYIRJFPSDVUDJGESYEGBLHJW
KOZGWSOTEDILZNHNSQJVDLTETXTLEOHXWVKTWQCGAWJJWHXWCQSKBJRBLUJ
VXTLTZAWTGSSUKTWVKZUWSKPKJJTGGDTOZNISIJTNTWQFNWHWGFTCJCWZHZ
SGSWDENWHSELMTJTNWSZGDQHAEOTFUGNWHISDURJTVDEKGOKTWUVJWGGORH
BDZFOFSNTSLVWEOZJVTRQLWEJBFQSKQTGISCXGSLUCLNHATHXXPSDXXSYSP
CUSDVGJXNWOZNRQYYTRKHVCEJXPEQPHKGOTXPECSYYFRRXWCRLWWKGSKQFU
HUKFWVKZ
```

We will attempt to break it with the crib `SINKINLOATHSOMEMUCKWITHOUTASPOOR`. Sadly, this will be a long process. First, we find the period to be five:

Then we “scrub” the crib across the ciphertext until we find a placement that has no inconsistencies when we put letters into the key table. The first consistent placement (and maybe the only one) is at position 112.

```
...23401234012340123401234012340123...
   SINKINLOATHSOMEMUCKWITHOUTASPOOR
   OQFYXTUGISKSGJTADZYILZNEAXBTKMUY
```

From this placement we can begin to fill in the key table:

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	I.....Y.J.EK.....
1	....T...X.....M....SA.I...
2	.....KL...ATU...OX.....
3	B.....Q..U.....YSZD.....
4	..Z....N.....FG...T.....

With this information, we can partially decrypt the ciphertext:

```
...L...TS...T.......O.STH..O....M.O..I..P.OS..O..S.ENT..EST
HAT..CKON...TOR........O...T..S..T.....AC.E..........SINKIN
LOATHSOMEMUCKWITHOUTASPOOR...T.HO.TSO..OR....A..WHA....A.A.
.U....I....I..INTH..SU..E.R..E.....OR........OU....R...I...
.STO.TS...WITH...I...I...IN....T..PE.I..U.....T...THA...IL.
.IN.S.U..O.TO.S..HT..TH..M.N.O.ESTHAT..S.E.IN....O.E......T
..T.U....S..ON.S...TO......E.LO.........S....O..N......O...
A...S...E.....ST..E.I....AN.THIS.W....TIL.....S.IT...ITS...
```

```
.L........STH...RS.....O.T......O.ET...T.RN..T........THIN.
.O.N..HT
```

By searching our word list(s), we find that `.CKON` should be `ECKON`. This is not a word, but that sequence of letters occurs only with `E` except in names. Since `E` is a high-frequency letter, knowing now that it is enciphered to `J` in alphabet three is a valuable piece of knowledge. So we will perform another partial decryption now rather than look for more parts of words.

```
...L...TS...T.......O.STH..O....M.O..I..P.OS..O.ES.ENT..EST
HAT.ECKON...TOR....E...O...T..S..T.....AC.E..........SINKIN
LOATHSOMEMUCKWITHOUTASPOOR...T.HO.TSO..OR....A..WHA....A.A.
.U....I....I..INTH..SU..E.R..E.....OR........OU....R...I...
.STO.TS...WITH...I...I...IN....T..PE.I..U.E...T...THA...IL.
.IN.S.U..O.TO.S..HT..TH..M.N.O.ESTHAT.ES.E.IN....O.E.E....T
..T.U....S..ONES...TO......E.LO........ES....O..N......O...
A...S...E.....ST..E.I....AN.THIS.W....TIL.....S.IT...ITS...
.L....E...STH...RS.....O.T......O.ET...T.RN..T........THIN.
.O.N..HT
```

Now our word list tells us that `P.OS..O.ES.ENT` is likely to be `PHOSPHORESCENT`. Since `H` and `R` are medium- and high-value letters, we decrypt again:

```
.H.L..HTS.H.T.......O.STH..O....M.O..I..PHOSPHORESCENT..EST
HAT.ECKON...TOR....E...O...T..S..T.....ACHER.........SINKIN
LOATHSOMEMUCKWITHOUTASPOOR.H.T.HO.TSO..OR....A..WHA....A.A.
.U....I....I.HINTH..SU..ERR..E.....OR........OU....R...I...
.STO.TS...WITH...I.H.I...IN....T..PERI..URE...T...THA...IL.
HIN.S.UR.O.TO.S..HT..TH..M.N.O.ESTHAT.ES.ERIN.H..O.E.E....T
..TRU..H.S..ONES.H.TO......E.LO.H......ESH...O..N..H.R.O...
A...S...E.....ST.HE.I....AN.THIS.W..P.TIL.....S.IT...ITS...
.L....EP.HSTH...RS.....O.T......O.ET...T.RN..T....R...THIN.
.O.N..HT
```

The beginning of the text, `.H.L..HTS.H.T`, could be `THELIGHTSTHAT`. Our key table now has this content:

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	I.P...UV..Y.J.EK...N.....
1	J...T..WX.....M....SA.I...
2	....H..KL...ATU..WOX.....
3	B...J...Q..U.....YSZD.....
4	..Z....NS....FGX..TJ.....

Another decryption with these partial keys:

```
THELIGHTSTHAT.I.....O.STH..O....M.O..I..PHOSPHORESCENT..EST
HAT.ECKON...TORI...E...O.T.TE.S..T..T.EACHER....E....SINKIN
LOATHSOMEMUCKWITHOUTASPOOR.HAT.HO.TSO..OR....A..WHAT..EA.A.
.U..A.I....ITHINTH..SU.TERR..E.....OR..A..I..OUT...R...I.T.
.STO.TS...WITH...ITH.I...IN....T..PERI..URET..T...THAT..ILT
HIN.S.UR.O.TO.S..HT..TH..MAN.O.ESTHAT.ESTERINTHE.O.E.E...ET
I.TRU.THES..ONESTH.TO..E...E.LOTHE.I...ESH...O..N..HAR.O..E
A...S.ITE.....STTHE.I....AN.THIS.W..P.TIL...E.S.IT.I.ITS...
.LI...EPTHSTH...RS..E..OAT....E.O.ET...T.RN..T....R...THING
.O.N..HT
```

Looking at this text, we see `SU.TERR..E..`, which is likely to be `SUBTERRANEAN`. With this identification, we gain A and N in two more alphabets. These are high-value letters, and we decrypt again.

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	I.P...UV..Y.JDEK...N.....
1	J...T..WX.....M....SA.I...
2	S...H..KL...ATU..WOX.....
3	B...J...Q..U.V...YSZD.....
4	VYZ....NS....FGX..TJ.....

```
THELIGHTSTHAT.IN.A..O.STH..O....M.O..I..PHOSPHORESCENT..EST
HAT.ECKON..NTORI...E...O.T.TE.S..T..T.EACHER....ENAN.SINKIN
LOATHSOMEMUCKWITHOUTASPOOR.HAT.HO.TSO..OR....A..WHAT..EA.A.
.U..A.I....ITHINTH..SUBTERRANEAN...OR..A..IN.OUT..AR...I.T.
.STO.TS..NWITH..AITH.I...IN....T.APERI..URET..T...THAT..ILT
HIN.S.UR.O.TO.S..HT..TH..MAN.ONESTHAT.ESTERINTHE.O.E.E...ET
I.TRU.THES..ONESTHATON.E...E.LOTHE.IN..ESH...O..N..HAR.O..E
A...S.ITEA.A..STTHE.I....AN.THIS.WA.P.TIL...E.S.IT.INITSB..
.LIN..EPTHSTH...RS..EN.OAT....E.O.ET...T.RN..T....R...THING
.O.N..HT
```

We have `.ITHIN` that must be `WITHIN`, `THAT.ESTERINTHE` that is likely `THATFESTERIN`, `.EPTHS` that can only be `DEPTHS`, and `N..HT` that must be `NIGHT`. Our key table is starting to look good:

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	I.P...UVW.Y.JDEK...N.....
1	J...T.VWX.....M....SA.I...
2	S..GHE.KL...ATU..WOX..N...
3	B...J...Q..U.V...YSZD.....

4	VYZ....NS....FGX..TJ.....

The next decryption is

```
THELIGHTSTHAT.IN.A..O.STH..O....M.O..I..PHOSPHORESCENT..EST
HAT.ECKON..NTORI...E...O.T.TE.SI.T..T.EACHER....ENAN.SINKIN
LOATHSOMEMUCKWITHOUTASPOOR.HAT.HO.TSO..OR...DA..WHAT..EA.A.
.U..A.I...WITHINTHI.SUBTERRANEAN...OR..A..INGOUT..AR...I.TI
.STOITS..NWITH..AITH.I...IN....T.APERI..URETI.T...THAT..ILT
HIN.S.UR.O.TO.SIGHT.ITH..MAN.ONESTHAT.ESTERINTHE.O.E.E.I.ET
I.TRU.THES..ONESTHATON.E...E.LOTHE.IN..ESH.F.O..N..HAR.O..E
A...S.ITEA.AI.STTHE.I.I..AN.THIS.WA.P.TIL...E.S.IT.INITSB..
.LINGDEPTHSTH...RS..EN.OAT....E.O.ET...T.RN..T....R...THING
.O.NIGHT
```

Here, .HAT cannot be THAT, because T in that alphabet is already assigned to a different ciphertext letter in that alphabet, so it must be WHAT. Later we see that `THAT.ESTERINTHE` sounds like it could be `THATFESTERINTHE`. It looks as though `TRU.` is at the end of a word; it must then be `TRUE`. Also, `THATON.E` is likely to be `THATONCE`. Searching a word list tells us that `AI.ST`, which we know to be at the end of a word, can only be part of `AGAINST`. We see that .WA.P is at the beginning of a word; it must be SWAMP, according to our word list.

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	I.P...UVW.Y.JDEK..GN.....
1	J.Q.T.VWX....EM....SA.I...
2	S..GHI.KL...ATU..WOX..N...
3	B...J.L.Q..UCV...YSZD.....
4	VYZ.E..NS....FGX..TJ..U...

The next decryption is

```
THELIGHTSTHAT.IN.A..O.STH..O...NM.O..I.EPHOSPHORESCENTE.EST
HAT.ECKONMENTORIS.FE...O.TSTE.SINT.ET.EACHER..S.ENAN.SINKIN
LOATHSOMEMUCKWITHOUTASPOORWHATGHO.TSO..OR...DA.SWHAT..EA.A.
.U..A.I...WITHINTHI.SUBTERRANEAN..NOR..A..INGOUTSNARES.I.TI
.STOITS.ENWITH..AITH.I.E.INGE..T.APERI.SURETI.T...THAT..ILT
HINGS.UR.O.TO.SIGHTWITH..MAN.ONESTHATFESTERINTHE.O.E.E.I.ET
ISTRUETHESE.ONESTHATONCE.E.E.LOTHE.IN..ESH...O..N.WHAR.O..E
A...S.ITEAGAINSTTHE.I.IN.AN.THISSWAMPSTIL...E.S.IT.INITSB..
.LINGDEPTHSTH.C.RS..EN.OAT.E..E.O.ET.E.T.RN..T....R..STHING
.O.NIGHT
```

We have never heard of phosphorescent ewes, so `E.ES` must be `EYES`. And what do phosphorescent eyes do? They bekon. So `.ECKON` is `BECKON`. Next, `T.EACHER` turns up `TREACHEROUS` in a word search. Then `T.EACHER..S.ENAN.SINK` is `TREACHEROUSMENANDSINK`. Then, `GHO.TS` is `GHOSTS`, and `THESE.ONESTHAT` is probably not `ZONES`, so it must be `BONES`.

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	ILP...UVW.Y.JDEK..GN....C.
1	J.QRT.VWX...UEM..NHSA.I...
2	SC.GHI.KL...ATU..WOX..N...
3	BG..J.L.Q..UCVW..YSZD.....
4	VYZ.E..NS....FGX..TJK.U...

```
THELIGHTSTHAT.IN.A..OSSTH.SO...NMOO..I.EPHOSPHORESCENTEYEST
HATBECKONMENTORIS.FE..MOOTSTE.SINT.ETREACHEROUSMENANDSINKIN
LOATHSOMEMUCKWITHOUTASPOORWHATGHOSTSO.MOR..RDA.SWHAT.REA.A.
.U..ABI..SWITHINTHISSUBTERRANEAN..NOR..A..INGOUTSNARES.I.TI
.STOITS.ENWITH.RAITH.I.E.INGE.STOAPERI.SURETISTO..THAT..ILT
HINGS.UR.O.TO.SIGHTWITH.UMANBONESTHATFESTERINTHEOO.EBE.I.ET
ISTRUETHESEBONESTHATONCE.E.E.LOTHE.INM.ESH...O..NOWHARBO.DE
A...S.ITEAGAINSTTHE.I.IN.AN.THISSWAMPSTIL.BRE.S.IT.INITSB..
BLINGDEPTHSTH.C.RS..EN.OAT.E..E.O.ET.EYT.RN.DTO...ROUSTHING
SO.NIGHT
```

In this round, we identify `T.ETREACHEROUS` as `THETREACHEROUS`, `WITH.RAITH.I.E` as `WITHWRAITHLIKE`, `.UMANBONES` as `HUMANBONES`, `INTHEOO.E` as `INTHEOOZE`, and `B..BLINGDEPTHS` as `BUBBLINGDEPTHS`. Also, `NOWHARBO.DEA..` can be `HOWHARBOR` but cannot be `DEATH` because of a conflict in key alphabet four.

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	ILP...UVW.Y.JDEK.OGNS.H.CF
1	JPQRT.VWX..D.EM..NHSA.I...
2	SC.GHI.KL...ATU..WOX..N...
3	BG..J.LPQ.TUCVW..YSZD.....
4	VYZ.E..NS....FGX..TJK.U...

```
THELIGHTSTHAT.IN.A..OSSTH.SO...NMOORLIKEPHOSPHORESCENTEYESTHATBEC
KONMENTORIS.FE...OOTSTE.SINTHETREACHEROUS.ENANDSINKINLOATHSOMEMUC
KWITHOUTASPOORWHATGHOSTSO..OR..RDA.SWHAT.REA.AL.U..ABI..SWITHINTH
ISSUBTERRANEAN..NOR..A.HINGOUTSNARES.I.TI.STOITSKENWITHWRAITHLIKE
.INGERSTOAPERI.SURETISTO..THAT..ILTHINGS.URKOUTO.SIGHTWITHHUMANBO
NESTHATFESTERINTHEOOZEBE.I.ETISTRUETHESEBONESTHATONCE.ERE.LOTHE.I
```

N..ESHL..OR.NOWHARBORDEA...S.ITEAGAINSTTHE.I.IN.AN.THISSWAMPSTIL.
BRE.SWITHINITSBUBBLINGDEPTHSTH.C.RS..EN.OATHE.BE.ORETHEYT.RN.DTO.
..ROUSTHINGSO.NIGHT

In this round we identify TH.SO...NMOOR as THESODDENMOOR (the quagmire), WITHWRAITHLIKE.INGERS as WITHWRAITHLIKEFINGERS, THAT..ILTHINGS.URKOUTO.SIGHT as THATEVILTHINGSLURKOUTOFSIGHT, THATONCE.WERE as THATONCEWERE, DEA... as DEADLY, BE.ORETHEYT.RN.DTO as BEFORETHEYTURNEDTO, and STIL.BRE.S as STILLBREWS, since what else could it do in its bubbling depths.

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	ILP.RTUVW.YZJDEK.OGNS.H.CF
1	JPQRT.VWX..D.EM..NHSABI.F.
2	SC.GHI.KL...ATU..WOXY.N...
3	BG.IJKLPQ.TUCVW..YSZD.A...
4	VYZDE..NS..C.FGX..TJK.U...

THELIGHTSTHATWIN.A..OSSTHESODDENMOORLIKEPHOSPHORESCENTEYESTHATBEC
KONMENTORIS.FELL.OOTSTE.SINTHETREACHEROUS.ENANDSINKINLOATHSOMEMUC
KWITHOUTASPOORWHATGHOSTSOF.OR.ERDA.SWHAT.READAL.U.EABIDESWITHINTH
ISSUBTERRANEANDENOR.EA.HINGOUTSNARESVI.TI.STOITSKENWITHWRAITHLIKE
FINGERSTOAPERILSURETISTOL.THATEVILTHINGSLURKOUTOFSIGHTWITHHUMANBO
NESTHATFESTERINTHEOOZEBELI.ETISTRUETHESEBONESTHATONCEWERE.LOTHEDI
N..ESHL.FOR.NOWHARBORDEADLYS.ITEAGAINSTTHE.I.IN.AN.THISSWAMPSTILL
BREWSWITHINITSBUBBLINGDEPTHSTHECURSE.ENLOATHE.BEFORETHEYTURNEDTOL
E.ROUSTHINGSOFNIGHT

We are getting close to the end. Here we can identify LIGHTSTHATWIN.A..OSS as LIGHTSTHATWINKACROSS, .OOTSTE.S as FOOTSTEPS, WHAT.READAL.U.E as WHATDREADALLURE, TISTOL.THAT as TISTOLDTHAT, DEADLYS.ITEAGAINST as DEADLYSPITEAGAINST, and TURNEDTOLE.ROUSTHINGS as TURNEDTOLEPROUSTHINGS.

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	ILPQRTUVW.YZJDEK.OGNS.H.CF
1	JPQRTUVWX.ZD.EML.NHSABI.F.
2	SC.GHI.KL..RATU..WOXY.N...
3	BGHIJKLPQ.TUCVWF.YSZD.A...
4	VYZDE..NS..C.FGX.ITJK.U...

```
THELIGHTSTHATWINKACROSSTHESODDENMOORLIKEPHOSPHORESCENTEYEST
HATBECKONMENTORISKFELLFOOTSTEPSINTHETREACHEROUSFENANDSINKIN
LOATHSOMEMUCKWITHOUTASPOORWHATGHOSTSOFFOR.ERDA.SWHATDREADAL
LUREABIDESWITHINTHISSUBTERRANEANDENORREA.HINGOUTSNARESVICTI
.STOITSKENWITHWRAITHLIKEFINGERSTOAPERILSURETISTOLDTHATEVILT
HINGSLURKOUTOFSIGHTWITHHUMANBONESTHATFESTERINTHEOOZEBELIKET
ISTRUETHESEBONESTHATONCEWERE.LOTHEDINFLESHL.FOR.NOWHARBORDE
ADLYSPITEAGAINSTTHELI.IN.ANDTHISSWAMPSTILLBREWSWITHINITSBUB
BLINGDEPTHSTHECURSE.ENLOATHEDBEFORETHEYTURNEDTOLEPROUSTHING
SOFNIGHT
```

This is our last round of filling in blanks. We identify `GHOSTSOFFOR.ERDA.S` as `GHOSTSOFFORMERDAYS`, `REA.HINGOUT` as `REACHINGOUT`, `VICTI.S` as `VICTIMS`, `.LOTHEDINFLESHL.FOR.` as `CLOTHEDINFLESHLYFORM`, and `AGAINSTTHELI.IN.` as `AGAINSTTHELIVING`. Our key table is now almost full:

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	ILPQRTUVW.YZJDEK.OGNS.H.CF
1	JPQRTUVWX.ZDKEML.NHSABI.F.
2	SCFGHIJKL..RATU..WOXY.N.E.
3	BGHIJKLPQ.TUCVWF.YSZD.A...
4	VYZDE..NS..CWFGX.ITJKLU.Q.

Our next decryption is the last, because there are no more holes. The plaintext is the poem "Swamp Demons" by C. A. Butz:

The lights that wink across the sodden moor
  Like phosphorescent eyes that beckon men
  To risk fell footsteps in the treacherous fen,
And sink in loathsome muck, without a spoor—
What ghosts of former days, what dread allure,
  Abides within this subterranean den?
  Or, reaching out, snares victims to its ken,
With wraith-like fingers, to a peril sure?

'Tis told that evil things lurk out of sight
  With human bones that fester in the ooze;
  Belike 'tis true, these bones that once were clothed
In fleshly form now harbor deadly spite
  Against the living, and this swamp still brews
Within its bubbling depths the curse men loathed
  Before they turned to leprous Things of Night!

Once we have recovered the plaintext and most of the key table, our next tasks is to identify the cipher as quagmire 1, 2, 3, or 4 (or none of these), and recover the keywords. How to identify the

cipher will be handled later. How to recover the keywords depends on which quagmire we have, and will be discussed later as well.

Programming tasks

1. Remember Task 2 of Unit 37? You wrote a function to compare a crib to a segment of ciphertext in order to place letters into a key table. Now you need a similar function, but which checks the column property, i.e., which checks that any two rows that share the same letter in the same position must not have different letters in another position.

2. If you haven't already, write a script to decipher a text with partial keys. It should use a dummy character such as "`.`" to display a letter that it cannot decipher.

3. If you do not use Linux or Mac and do not have the `grep` program on your system, then write a Python script to search your word list(s) for partial words. It should be able to search `T.EACHER` and find `TREACHEROUS` and `UNTREACHEROUS`, for example.

Exercises

1. Break this ciphertext. The plaintext begins with "And after all the weather was ideal. They could not have had a more perfect day."

```
GKBDJGKOROQGNBXHEGNBQTEFOYCDQGNBPFVHUYLUAXGVCKESGJMXICKODHG
GJAPIVEGDRXHUWMRXALOCUKILNABUZSKOCGMGCFLGQURQXDZTTECZOLCFUK
VHTAYOVHJLLOCGNBTORUCASRIZUBBTMGNAFDDUXCJLKXTDMOHNRFULPFXJC
PMTKQGSINAIPZRTVBQPLUMAQGIYKOFDHOKBLAWFOKYHHNCKKUYZWDUKIRGW
LZEYJQXHICOKEPLUVULPMRTECJZNRQRSHGNBBDZALIRPZBRBUPIFCECXIGN
BBDMFEMJDUGRERGFUKKSHITKYUUPXOKC
```

2. Break this ciphertext. The plaintext starts with "In the beginning God created Adam," and the period is less than ten.

```
JVHWFCUPXHBGVLEPTXOFRHUPNEQEMHEFQXCVVHRLJEDXZNZUOFJVEJERSTQ
HQQSMCJFUMHEKUUNBBQOXZUJQBZTQJXIUSQNDUBGFBHMRKTVQUPTQVPQPHF
RKRVGANMFHRUMQMHMUZWCXUZUSFWUFWTXUTCU
```

Unit 62
Quagmire 1 cipher

The *quagmire 1 cipher* (also called *polyalphabetic type 1*) uses a mixed alphabet for the plaintext and shifted alphabets for the ciphertext. The mixed alphabet is generated from a keyword. The shifts of the ciphertext alphabets form another keyword, as they do in the Vigenère cipher.

Here is an example. The keywords are `QUAGMIRE` (alphabet) and `CIPHER` (shift). First, look at the table of key alphabets. Notice that the shift keyword appears in the column under the indexing letter, which in this example is `A`, in the plaintext alphabet.

	plaintext alphabet QUAGMIREBCDFHJKLNOPSTVWXYZ
0	ABCDEFGHIJKLMNOPQRSTUVWXYZ
1	GHIJKLMNOPQRSTUVWXYZABCDEF
2	NOPQRSTUVWXYZABCDEFGHIJKLM
3	FGHIJKLMNOPQRSTUVWXYZABCDE
4	CDEFGHIJKLMNOPQRSTUVWXYZAB
5	PQRSTUVWXYZABCDEFGHIJKLMNO

Now we encipher a short message with this key table. The center row is the repeating shift key.

```
THIS MESSAGE IS ENCRYPTED WITH A QUAGMIRE CIPHER
CIPH ERCIPHE RC IPHERCIPH ERCI P HERCIPHE RCIPHE
USSY GWTZPIJ UT NDOINSAUP YUUS P FDRDKSLJ YFYZMI
```

If we rearrange the key table so that the plaintext alphabet is unscrambled, we have:

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	CIJKHLDMFNOPEQRSAGTUBVWXYZ
1	IOPQNRJSLTUVKWXYGMZAHBCDEF
2	PVWXUYQZSABCRDEFNTGHOIJKLM
3	HNOPMQIRKSTUJVWXFLYZGABCDE
4	EKLMJNFOHPQRGSTUCIVWDXYZAB
5	RXYZWASBUCDETFGHPVIJQKLMNO

When we attack a cipher with cribs, the key table will look like this, with the plaintext alphabet in normal order.

Suppose instead that we use the first letter of the scrambled alphabet as the indexing letter, as we decided to do in Unit 60. Then the key table is this:

	plaintext alphabet QUAGMIREBCDFHJKLNOPSTVWXYZ
0	CDEFGHIJKLMNOPQRSTUVWXYZAB
1	IJKLMNOPQRSTUVWXYZABCDEFGH
2	PQRSTUVWXYZABCDEFGHIJKLMNO
3	HIJKLMNOPQRSTUVWXYZABCDEFG
4	EFGHIJKLMNOPQRSTUVWXYZABCD
5	RSTUVWXYZABCDEFGHIJKLMNOPQ

If we straighten out the plaintext alphabet, then the table is this:

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	EKLMJNFOHPQRGSTUCIVWDXYZAB
1	KQRSPTLUNVWXMYZAIOBCJDEFGH
2	RXYZWASBUCDETFGHPVIJQKLMNO
3	JPQROSKTMUVWLXYZHNABICDEFG
4	GMNOLPHQJRSTIUVWEKXYFZABCD
5	TZABYCUDWEFGVHIJRXKLSMNOPQ

Suppose we have decrypted a ciphertext, and we have the key table, but it looks like the last one above, i.e., there is no obvious relationship between the rows. How can we know whether we have a quagmire 1 cipher? There is an easy way to tell. Recall Equation 60.1:

$$Q(k_1, k_2, k_3) \;=\; S(k_3)\, V(k_2')\, S^{-1}(k_1)$$

where, for Q1, $k_3 = e$ and $k_2' = k_2$:

$$Q_1(k_1, k_2) \;=\; V(k_2)\, S^{-1}(k_1) \tag{62.1}$$

This means that the permutations in the Q1 tableau are of the form

$$q_n \;=\; R_n \circ k_1^{-1} \tag{62.2}$$

where, as you remember, the R_n are rotations that appear in the Vigenère tableau. Because the Q1 set is made by composing each member of the Vigenère on the right with a constant permutation, we call the set

$$Q_1[k_1] \;=\; \{R_n \circ k_1^{-1}\} \tag{62.3}$$

a *right coset* of $\{R_n\}$. Now suppose that we take any two different members of it and calculate $q_m \circ q_n^{-1}$:

$$\begin{aligned} q_m \circ q_n^{-1} &= (R_m \circ k_1^{-1}) \circ (R_n \circ k_1^{-1})^{-1} \\ &= R_m \circ k_1^{-1} \circ k_1 \circ R_n^{-1} \\ &= R_m \circ e \circ R_{-n} \\ &= R_m \circ R_{-n} \;=\; R_{m-n} \end{aligned} \tag{62.3}$$

where arithmetic in the subscript is done modulo 26. The result is another rotation. If we take pairs of keys and calculate this composition and always get a rotation, then that indicates we have a Q1 cipher. This works even when we have an incomplete key table. After a crib-based attack, we might have something like this:

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	CIJKHLDMF.OPEQRS.GTUBVW.Y.
1	IOPQNRJSL.UVKWXY.MZAHBC.E.
2	PVWXUYQZS.BCRDEF.TGHOIJ.L.
3	HNOPMQIR..TUJVWX.LYZGAB.D.
4	EKLMJNFOH.QRGST..IVWDXY.A.
5	RXYZWASBU.DETFGH.VIJQKL.N.

Alphabet key 0 composed with the inverse of 1 is clearly a rotation:

UVW.Y..BCDEFGHIJKLM.OPQRST

There are some special cases of the quagmire 1 cipher:

- period = 1: monoalphabetic substitution in which the keyword is used to generate the plaintext alphabetic
- mixed alphabet = regular alphabet: Vigenère cipher
- mixed alphabet generated as in affine cipher: periodic affine cipher

Reading and references

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; chapter XVIII.

American Cryptogram Association, www.cryptogram.org/downloads/aca.info/ciphers/QuagmireI.pdf

Programming tasks

1. In your quagmire module, define a class for Q1 that extends the base quagmire class. The only thing you will need to do is override `__init__()` to take only one alphabet key and one shift key. The other alphabet key that your parent class is expecting must be set to the unscrambled alphabet.

   If you don't yet know what any of this means, then implement a Q1 cipher by whatever means you like.

2. Implement a dictionary attack on the quagmire 1 cipher. Naïvely, you need to nest two loops through your word list, since the cipher needs two keywords. In a later unit we will see a faster way.

Exercises

1. Verify that the example above does indeed factor into a monoalphabetic substitution and a Vigenère, and that the keys are the ones given.

2. Encipher this text with the keywords `ULTIMATE` (alphabet) and `QUESTION` (shifts). Use indexing letter `U`.

   > O people waiting in the Shadow of Deep Thought! Honoured Descendants of Vroomfondel and Majikthise, the Greatest and Most Truly Interesting Pundits the Universe has ever known... The Time of Waiting is over! Seven and a half million years our race has waited for this Great and Hopefully Enlightening Day! The Day of the Answer! Never again, never again will we wake up in the morning and think Who am I? What is my purpose in life? Does it really, cosmically speaking, matter if I don't get up and go to work? For today we will finally learn once and for all the plain and simple answer to all these nagging little problems of Life, the Universe and Everything!
   >
   > (from *The Hitchhiker's Guide to the Galaxy* by Douglas Adams)

3. Decipher this ciphertext with the keywords `WAR` (alphabet) and `PEACE` (shifts). The indexing letter is `A`.

```
IMFUUBNTVNDLUSTVXIHDEFMGNDXPVMQJFJFUETUYFJBUXPXFUDPWAQJKJOV
DXNDKHEYMGGUEOGNDIFHIOESITQJSHJDGFITQJTKJCNEGQUTGVMUXZHSINF
VMSJOVYQBZLXXXIHERWPDUINPQTVJUDTFJCZFJWSLEPSERKIMFEFYXJUMUR
QLFUGXVMUYOLXUISVEIJSVDETGVMUXIDJUJWLWINOJUEDFDWUYBCWYEAQIE
```

```
HFCSYEZHFUEMDJPIXHKVJDVNKJMZNDGFLSWXIHXXNBGJPWTCWYEPQQMJNHF
WJECWPIJUXYSDVYDNTCKIJBCSEXIHFTJDCIUTGFTDKUUJTKJJMINOJXXJGD
TDXJHFHGFVDUJOVMUXIDJUWUSJQWTCXUWADJYSSRRUUMCHUWADGYXBCFMEO
GNDTTKJQWTKJMKMWHIYAVJPHDRFTNOJXEXIHKEFTWSUWPIDPFCWXYSHHSUF
AOXXJXITBQPBLUTHDEFMJFEBQJQJHJUDEHNAFTCUBLWUWTKJOMPOJEKTKJD
TBVMUFOSEQXPIXXJFWFEUFCSPSECWYETLHBEOGRPWSIFERQRFIYHCQITTKJ
RJBLSWWTDEYXPFJPSJCHERQDNHJSVMUENHFYHAUXXJAVQPSTLHYWMCSTWJQ
HBYELSWXIHGQNTLWXNSOJHEUUXQEMCWYEAQIIMFUTJXIHFDUPDXYTORKPKB
LHPJAUXPWJCWCEMOJQXICSIMFRXXJBUEDIZLXXEMHWHIFINDNTHDUWTHFDK
BRSINFDHERQDNHJSFMYSACSTXIHHEYOVFYJSVTIMFUTJXIRKYX
```

4. Break this ciphertext with a dictionary attack. Both keywords are common five-letter English words. The indexing letter is A.

```
VXUHEXTOLHJMJVCVCHQCQENAZOZWKOMQNHEJRGQUAOJVCYZFAGVFXAUAMDQ
CKZMQKUTYGVEXDENTGGQKFCGCNEJIVIMXDBNVROMESMYCBTQITTTULNFLHT
QMKZNCSYOJFUAMJVCVCGMVXTFAULMWSJVPYHCUOSMEKOJFHTXYCTUOSMEKO
WATLFUIHLTGXEEXBAEJNWACWOJFHTXYCGAURFSTPBAEJWNCMBZNDPBRTDUO
OJSUGARCZOUYHHTXYCGAURFSTPIDHTAXINGPXCSTDYINEHTQTTENAZJMYNP
KEZOKYETZCVLRAUGCGCHGCBQKLMFATBRGRUAMDVCVCFLEXWBHKXTORTMQNC
BTOXSNQQTLPMCOMTGNRCHBYIZCLO
```

key letter	plaintext alphabet QUAGMIREBCDFHJKLNOPSTVWXYZ
A	ABCDEFGHIJKLMNOPQRSTUVWXYZ
B	BCDEFGHIJKLMNOPQRSTUVWXYZA
C	CDEFGHIJKLMNOPQRSTUVWXYZAB
D	DEFGHIJKLMNOPQRSTUVWXYZABC
E	EFGHIJKLMNOPQRSTUVWXYZABCD
F	FGHIJKLMNOPQRSTUVWXYZABCDE
G	GHIJKLMNOPQRSTUVWXYZABCDEF
H	HIJKLMNOPQRSTUVWXYZABCDEFG
I	IJKLMNOPQRSTUVWXYZABCDEFGH
J	JKLMNOPQRSTUVWXYZABCDEFGHI
K	KLMNOPQRSTUVWXYZABCDEFGHIJ
L	LMNOPQRSTUVWXYZABCDEFGHIJK
M	MNOPQRSTUVWXYZABCDEFGHIJKL
N	NOPQRSTUVWXYZABCDEFGHIJKLM
O	OPQRSTUVWXYZABCDEFGHIJKLMN
P	PQRSTUVWXYZABCDEFGHIJKLMNO
Q	QRSTUVWXYZABCDEFGHIJKLMNOP
R	RSTUVWXYZABCDEFGHIJKLMNOPQ
S	STUVWXYZABCDEFGHIJKLMNOPQR
T	TUVWXYZABCDEFGHIJKLMNOPQRS
U	UVWXYZABCDEFGHIJKLMNOPQRST
V	VWXYZABCDEFGHIJKLMNOPQRSTU
W	WXYZABCDEFGHIJKLMNOPQRSTUV
X	XYZABCDEFGHIJKLMNOPQRSTUVW
Y	YZABCDEFGHIJKLMNOPQRSTUVWX
Z	ZABCDEFGHIJKLMNOPQRSTUVWXY

Table 62.1: Tableau for our example quagmire 1 cipher that uses alphabetic keyword `QUAGMIRE`*. The indexing letter is* `A`*.*

key letter	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
A	CIJKHLDMFNOPEQRSAGTUBVWXYZ
B	DJKLIMENGOPQFRSTBHUVCWXYZA
C	EKLMJNFOHPQRGSTUCIVWDXYZAB
D	FLMNKOGPIQRSHTUVDJWXEYZABC
E	GMNOLPHQJRSTIUVWEKXYFZABCD
F	HNOPMQIRKSTUJVWXFLYZGABCDE
G	IOPQNRJSLTUVKWXYGMZAHBCDEF
H	JPQROSKTMUVWLXYZHNABICDEFG
I	KQRSPTLUNVWXMYZAIOBCJDEFGH
J	LRSTQUMVOWXYNZABJPCDKEFGHI
K	MSTURVNWPXYZOABCKQDELFGHIJ
L	NTUVSWOXQYZAPBCDLREFMGHIJK
M	OUVWTXPYRZABQCDEMSFGNHIJKL
N	PVWXUYQZSABCRDEFNTGHOIJKLM
O	QWXYVZRATBCDSEFGOUHIPJKLMN
P	RXYZWASBUCDETFGHPVIJQKLMNO
Q	SYZAXBTCVDEFUGHIQWJKRLMNOP
R	TZABYCUDWEFGVHIJRXKLSMNOPQ
S	UABCZDVEXFGHWIJKSYLMTNOPQR
T	VBCDAEWFYGHIXJKLTZMNUOPQRS
U	WCDEBFXGZHIJYKLMUANOVPQRST
V	XDEFCGYHAIJKZLMNVBOPWQRSTU
W	YEFGDHZIBJKLAMNOWCPQXRSTUV
X	ZFGHEIAJCKLMBNOPXDQRYSTUVW
Y	AGHIFJBKDLMNCOPQYERSZTUVWX
Z	BHIJGKCLEMNODPQRZFSTAUVWXY

Table 62.2: Tableau for our example quagmire 1 cipher that uses alphabetic keyword `QUAGMIRE` *and indexing letter* `A`*. In this instance of the tableau, columns have been rearranged so that the plaintext letters are in alphabetical order. The tableau is nevertheless equivalent to the one in Table 62.1.*

Unit 63
Two-stage attack on the quagmire 1 cipher

Because the quagmire 1 can be factored into a monoalphabetic substitution followed by a Vigenère cipher, the shifts are exposed to cryptanalysis. In this attack, we find the shifts by comparing the monogram frequencies of slices of the ciphertext. What remains then is a monoalphabetic substitution, which we can break with the technique in Unit 36.

Here's how the attack works:

1. find the period *m*
2. partition the ciphertext into *m* slices; each slice contains every m^{th} letter, but each slice starts from a different one of the first *m* letter of the ciphertext
3. find the monogram frequencies for each slice
4. fix the frequency table for the first slice; for the others, shift them left (with wrap-around) until they match as closely as possible the table of the first slice
5. the shifts needed to align the frequency tables forms a Vigenère key; decipher the ciphertext with this key
6. break the resulting text as a monoalphabetic substitution

To recover the original keywords of the quagmire cipher, look at the key that you find for the monoalphabetic substitution. Take the value of its first letter, where we start at A = 0, B = 1, etc. Add that value to each of the shifts that you found in step 4. Convert the resulting shifts to letters to get the shift key of the quagmire. To find the alphabet keyword, invert the key of the monoalphabetic substitution. You may not always get a recognizable keyword, due to problems with infrequent letters.

Programming tasks

1. Implement the attack. Use the cosine of the angle between vectors to find the best match for the frequency tables. Feel free to also use your function that performs the hill-climbing attack on the monoalphabetic substitution cipher.

Exercises

1. Break this ciphertext and find the original quagmire keywords.

```
TNBOWMQSCQFMFOBZVKYAVNUBJVKSLLKESDBGLJPVFNEFLHUGNKVRDOQCVKNRKJFX
WKIUDNVWBTBHPSESLTBJIVUHUIIAGGKMQCMINEOUAMYONMATPOJVSSLXLNOFJAVF
```

DSDEXBGTUUENVNHDXILUDYXGFOJRNCHRJOTVOZLBMCKJBUIUJRPNRGOPBTTCXIET
UPBCJBWEWOVUAMVDAGEIKFZLLPSJVHPSMLQSSSLNHKMKZSMLYACCKGPUZAEWKBBL
LSXIXTPGMTCSCCXEFQEXIPPGKJFSBCLKPFSWBTDXYMRIVPPSACLVPFSVFSGLICHF
PHPSACLXLKGBFOBRVVEGDEBTACLKPRKGPXVMVPZSCAGPCMQSCEEGUPEHUUPVNJBU
ETZWZGEIUPRTWDPREEYUPVYGEWAEUSKXNHZFFOMHWPUOYKDAMFWNAKUEKSKFETZR
ZQTNJUWYBHNXNATUWHNOCPKOTFMTUMENTNJURXUGUISOPMMZHHENVSDHMLVVENVW
FCJTUUYSKCEFOMQOEXAEZXMKLVNFSEHXKMQRFTWOBSOLHBRJFOMTRXIJPWAESFIG
NHUIIETKMLHKXEGSBOOIJREYIETKCGPIASEPFJALQSESFLHPEGHYLVYRPNWGLCSX
AETKMUCSDEJSKFXTZGPKTTXBAUHTPFVGPSIGPSXEIMBCIZLTFXKOKFZBYSLXXCTB
XIVQYXDEOUKMDODIDPUZACLUNKEOUCKELHZKIOQURXROCLFRIFIKVVWGSBDOQLVA
PJFGZVVZLFYIRTKTWFHBLMVGUPXNAQEGETPPVXVVENVDVQNHHFRWRSTKMALPMTRS
UDB

Unit 64
Attacking quagmire 1 with cribs: symmetry

In Unit 61 we dealt with the placement of a crib against a ciphertext and showed how to intelligently guess at words in other parts of the text in order to fill in the key table. In this unit, we focus on how to fill in the table knowing only what we learn from the crib and the structure of the cipher. To explain the method, we work through an example. Suppose we know the period of the cipher to be four and have correctly placed a crib to have this plaintext-ciphertext pair:

```
0123 0123012 301 230123012 3012 30123012
THIS MESSAGE WAS ENCRYPTED WITH QUAGMIRE
YESJ UQHJHDN LHK NVORPSYQZ LILB ILUAGIRN
```

Here is what the crib tells us about the key table:

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	H.O.....I...U.....YL.....
1	U...Q.DE.....RKL.....
2	...ZN.ABS.....H.....P.
3	.....GV.SI.J...L...

Instead of working with the plaintext alphabet in order, we benefit greatly by ordering the letters in each row and allowing the plaintext alphabet to be mixed, as in Table 62.1. Let's begin by looking at the H in row 0 under the plaintext A. We fill in the rest of row 0 as a rotation of the alphabet:

	plaintext alphabet A.....
0	HIJKLMNOPQRSTUVWXYZABCDEFG

Above it we can place other letters in the plaintext alphabet for letters that we know in row 0: C above O, I above I, M above U, T above Y, and U above L.

	plaintext alphabet AI..U..C.....M...T.....
0	HIJKLMNOPQRSTUVWXYZABCDEFG

Now we can add a rotated alphabet to row 1, where we know that U is below plaintext A.

	plaintext alphabet AI..U..C.....M...T.....
0	HIJKLMNOPQRSTUVWXYZABCDEFG
1	UVWXYZABCDEFGHIJKLMNOPQRST

Above row 1 we can add letters to the plaintext alphabet: E above Q, etc.

	plaintext alphabet AI..U..C.GH..M..ST....ER..
0	HIJKLMNOPQRSTUVWXYZABCDEFG
1	UVWXYZABCDEFGHIJKLMNOPQRST

For row 2, we do not know what letter is in column A, but we can anchor it with N under plaintext E:

	plaintext alphabet AI..U..C.GH..M..ST....ER..
0	HIJKLMNOPQRSTUVWXYZABCDEFG
1	UVWXYZABCDEFGHIJKLMNOPQRST
2	RSTUVWXYZABCDEFGHIJKLMNOPQ

Add what we now know from row 2 to the plaintext alphabet:

	plaintext alphabet AI..U..CDGH..M..ST....ERY.
0	HIJKLMNOPQRSTUVWXYZABCDEFG
1	UVWXYZABCDEFGHIJKLMNOPQRST
2	RSTUVWXYZABCDEFGHIJKLMNOPQ

Add row 3 and what it can tell us about the plaintext alphabet:

	plaintext alphabet AIN.U..CDGH..M.QSTW...ERYP
0	HIJKLMNOPQRSTUVWXYZABCDEFG
1	UVWXYZABCDEFGHIJKLMNOPQRST
2	RSTUVWXYZABCDEFGHIJKLMNOPQ
3	TUVWXYZABCDEFGHIJKLMNOPQRS

The plaintext alphabet so far is

AIN.U..CDGH..M.QSTW...ERYP

The underlined section is likely to be part of the remaining letters after the keyword was placed, since they are in alphabetical order. The letters still missing in the plaintext alphabet are B, F, J, K, L, O, V, X, Z. So between M and Q we can place O. Between H and M the options are JK, JL, KL. After W we can safely place XZ. That lets us rotate the keyword portion to the front. Now we have

.ERYPAIN.U..CDGH..MOQSTWXZ

and the remaining letters are B, F, J, K, L, V. We know that F and V must be in the keyword, because there are no places for them in the alphabetically ordered segment. Of all six options, only V gives a meaningful word if we place it in the first spot.

VERYPAIN.U..CDGH..MOQSTWXZ

We can guess that the keyword is VERYPAINFUL, even though this process is not.

Reading and references

William F. Friedman, The Principles of Indirect Symmetry of Position in Secondary Alphabets and Their Application in the Solution of Polyalphabetic Substitution Ciphers, Washington D.C.: U.S. Government Printing Office, 1935, www.nsa.gov/Portals/70/documents/news-features/declassified-documents/friedman-documents/publications/FOLDER_226/41760669079979.pdf

Programming tasks

1. It is possible to automate the process, except for the educated guessing at the end, but we can try.

Exercises

1. Break this ciphertext. The plaintext begins with “Although this story is concerned principally with the Man ...”

```
TQAJLDZEAJBCIDPAUOILPWELHRCOMBURINMNCQYEBJQDBKZNDHOOWKRGENCJQFDN
ACSHQKOETJCBBJPSTCEKTRCFWLDDAJAJUMCKJUJWANFORJFWMDPDFWGKUWOMZLER
AJYQUQNBSLWWAKOCVIIIBYDHSGGOKFTIXLSFEWBXZSJIXLLSSOZOTWDMACTRFWFL
XDFXKSCSEIBXZIOIXLTISMCLTJNGYGQHNMWOEREWCODDBKLZURFXKYYGSALQRVAR
YBZXTXKKUWNMWLRGQVLHRVAJAJVFTQOOFDFXKJQSUSXZRVBMMCTQFIWWRDPXFWTD
AKOODNPWWKRZBXJLIXDLYPJTDMPJRVSXJNDJHKAXFVCIWHIHANPSTLAXOOCB
```

Unit 65
Attacking quagmire 1 with cribs: group theory

Recall from Equation 62.2 that the permutations of a quagmire 1 cipher have the form

$$q_n = R_n \circ k_1^{-1}$$

If we invert this, we have a rotated version of the cipher's key:

$$q_n^{-1} = (R_n \circ k_1^{-1})^{-1} = k_1^{-1} \circ R_{-n}$$

Therefore, given a partially filled key table, our strategy is to invert each row, line them up together so that they can share letters, then invert again to return to the original table.

Here is the example from the previous unit:

```
0123 0123012 301 230123012 3012 30123012
THIS MESSAGE WAS ENCRYPTED WITH QUAGMIRE
YESJ UQHJHDN LHK NVORPSYQZ LILB ILUAGIRN
```

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	H.O.....I...U.....YL.....
1	U...Q.DE.....RKL.....
2	...ZN.ABS.....H.....P.
3	.....GV.SI.J...L...

Let us first invert each row:

0 ($^{-1}$)	.....AI..U..C.....M...T.
1 ($^{-1}$)	...GH.....ST....ER..A.....
2 ($^{-1}$)	GH.....S.....E.Y..I.....D
3 ($^{-1}$)	.....M.QS.W.....P..N....

Now we line them up if possible, with some shifts:

0 ($^{-1}$)	.....AI..U..C.....M...T.
1 ($^{-1}$)	...GH.....ST....ER..A.....
2 ($^{-1}$)	GH.....S.....E.Y..I.....D
3 ($^{-1}$)	.....M.QS.W.....P..N....

Those shifts are really rotations:

0 ($^{-1}$)	.....AI..U..C.....M...T.
1 ($^{-1}$)	...ER..A.....GH.....ST.
2 ($^{-1}$)	...E.Y..I.....DGH.....S..
3 ($^{-1}$)	.....P..N.....M.QS.W

Now they can all share letters:

0 ($^{-1}$)	...ERYPAIN.U..CDGH..M.QSTW
1 ($^{-1}$)	...ERYPAIN.U..CDGH..M.QSTW
2 ($^{-1}$)	...ERYPAIN.U..CDGH..M.QSTW
3 ($^{-1}$)	...ERYPAIN.U..CDGH..M.QSTW

Undo the rotations:

0 ($^{-1}$)	...ERYPAIN.U..CDGH..M.QSTW
1 ($^{-1}$)	.CDGH..M.QSTW...ERYPAIN.U.
2 ($^{-1}$)	GH..M.QSTW...ERYPAIN.U..CD
3 ($^{-1}$)	CDGH..M.QSTW...ERYPAIN.U..

Finally, reverse the inversion by inverting again:

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	H.OPD.QRI...UJ.GWEXYL.Z.F.
1	U.BCQ.DEV...HW.TJRKLY.M.S.
2	R.YZN.ABS...ET.QGOHIV.J.P.
3	T.ABP.CDU...GV.SIQJKX.L.R.

We now have the ciphertext letters for the most common letters (except maybe L) in all slices of the text, and with them we can decrypt most of the remaining text beyond the limits of the crib. As a general rule, if any letter appears in the crib, then we will be able to decrypt that letter anywhere in the text. Then, to fill in any remaining holes in the decryption, we can recognize words and add missing letters as we did in Unit 61.

Reading and references

Thomas Kaeding, “Classical substitution ciphers and group theory,” Cryptology ePrint Archive, report 2023/669.

Programming tasks

1. Automate the process, up until the point that you must resort to the traditional technique of Unit 61.

Exercises

1. Break this ciphertext. The plaintext begins with “Here was one of the most embarrassing ...”

```
PDALMHXPJWVHZQWUOYGWUEBHSHXYRUJOIBWKTKUFQDURUSJEFIKFSUCHZKGCKXJL
MVRNOFVOMIRHXZHTICEHAVZYWGKYKVWHBHGBKBURRPOSGMHZCRUJOALNWMBJTFJP
JMVRNOVQSEQCUACIGKACORPDCIGKXZZYSBPHWJJOLUFANXWIANGCKXGWKTGFHWHL
ULWAJNUCKTKJEQDPPRPJYQTLJZ
```

Unit 66
Quagmire 1: recovering keywords

Suppose we have solved a ciphertext encrypted with Q1, and that we have completely filled the key table. We now want to recover the keywords from the table. We can read the shift key a column of the table. Unless the cipher used an indexing letter that was not the first letter of its alphabet keyword, that column tells us the first letter of the other keyword. Here is as example from Unit 62:

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	EKLMJNFOHPQRGSTUCIVWDXYZAB
1	KQRSPTLUNVWXMYZAIOBCJDEFGH
2	RXYZWASBUCDETFGHPVIJQKLMNO
3	JPQROSKTMUVWLXYZHNABICDEFG
4	GMNOLPHQJRSTIUVWEKXYFZABCD
5	TZABYCUDWEFGVHIJRXKLSMNOPQ

Remember that each row of the key table has the form

$$q_n = R_n \circ k_1^{-1}$$

Its inverse is $k_1 \circ R_{-n}$, which is a rotation of k_1. Therefore, to find the alphabet keyword, we need to invert one of the rows of the key table. Take row 2, for instance; its inverse is

FHJKLNOPSTVWXYZQUAGMIREBCD

We cam rotate it to find the scrambled alphabet used by the quagmire:

k_1 = QUAGMIREBCDFHJKLNOPSTVWXYZ

Programming tasks

1. Write a function that can take an alphabetic permutation and rotate it until it looks like a keyword followed by the remaining letters, where the remaining letters have been added in alphabetical order. There will be some ambiguity if the keyword starts with Z.

Unit 67
Quagmire 2 cipher

The *quagmire 2 cipher* (also called *polyalphabetic type 2*) uses a mixed and shifted alphabet for the ciphertext. The mixing is generated from a keyword. The shifts form another keyword, as they do in the Vigenère cipher.

Here is an example. The keywords are `QUAGMIRE` and `CIPHER`. First, look at the table of key alphabets. Notice that the shift keyword appears under the `A` in the plaintext alphabet, but that we could have chosen a different indexing letter.

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	CDFHJKLNOPSTVWXYZQUAGMIREB
1	IREBCDFHJKLNOPSTVWXYZQUAGM
2	PSTVWXYZQUAGMIREBCDFHJKLNO
3	HJKLNOPSTVWXYZQUAGMIREBCDF
4	EBCDFHJKLNOPSTVWXYZQUAGMIR
5	REBCDFHJKLNOPSTVWXYZQUAGMI

Now we encipher a short message with this key table. The center row indexes the key alphabet.

```
THIS MESSAGE IS ENCRYPTED WITH A QUAGMIRE CIPHER
CIPH ERCIPHE RC IPHERCIPH ERCI P HERCIPHE RCIPHE
AHQM SDUXPPF KU CIKYMYYWL GKAH P AURLOQGF BOTZNY
```

There are some special cases of the quagmire 2 cipher:

- period = 1: monoalphabetic substitution (keyword cipher)
- mixed alphabet = regular alphabet: Vigenère cipher
- mixed alphabet generated as in affine cipher: periodic affine cipher

The quagmire 2 cipher can be factored into a Vigenère cipher followed by a monoalphabetic substitution cipher. From Equation 60.1 with $k_1 = e$,

$$Q_2(k_2, k_3) = S(k_3)\, V(k_2') \tag{67.1}$$

The key for the monoalphabetic substitution is the same as the mixed alphabet generated by the alphabet keyword of the quagmire, but the Vigenère key is the shift keyword of the quagmire after it is

encrypted by the inverse of the monoalphabetic substitution, as in Equation 60.2. Because the second factor of the quagmire 2 is a substitution cipher, it is resistant to the two-stage attack on the quagmire 1.

From Equation 67.1, we find that the permutations in the tableau of a Q2 have the form

$$q_n = k_3 \circ R_n \tag{67.2}$$

where the R_n are rotations that appear in the Vigenère tableau. Because the elements of the set

$$Q_2[k_3] = \{k_3 \circ R_n\} \tag{67.3}$$

are the elements of the Vigenère set composed on the left by a constant, we call it a *left coset* of $\mathcal{V}$. Now suppose that we take any two different members of the set and calculate $q_m^{-1} \circ q_n$:

$$\begin{aligned} q_m^{-1} \circ q_n &= (k_3 \circ R_m)^{-1} \circ (k_3 \circ R_n) \\ &= R_m^{-1} \circ k_3^{-1} \circ k_3 \circ R_n \\ &= R_{-m} \circ e \circ R_n \\ &= R_{-m} \circ R_n = R_{n-m} \end{aligned} \tag{67.4}$$

where arithmetic in the subscript is done modulo 26. The result is another rotation. If we take pairs of keys and calculate this composition and always get a rotation, then that indicates we have a Q2 cipher. This works even when we have an incomplete key table. After a crib-based attack in which we have not identified the cipher yet, we might have a key table like this:

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	CDFHJKLNO.STVWXY.QUAGMI.E.
1	IREBCDFHJ.LNOPST.WXYZQU.G.
2	PSTVWXYZQ.AGMIRE.CDFHJK.N.
3	HJKLNOPST.WXYZQ..GMIREB.D.
4	EBCDFHJK..OPSTVW.YZQUAG.I.
5	REBCDFHJK.NOPSTV.XYZQUA.M.

The inverse of alphabet key 0 composed with key 1 is a rotation, albeit with some holes:

W.Y.ABCDE.GHI.KL.NOP.RS.U.

By taking combinations as in Equation 67.4 we can decide whether we are dealing with a quagmire 2 if they all look like rotations.

Reading and references

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; chapter XVIII.

American Cryptogram Association, www.cryptogram.org/downloads/aca.info/ciphers/QuagmireII.pdf

Fletcher Pratt, *Secret and Urgent: The Story of Codes and Ciphers*, New York: Bobbs-Merrill, 1939, chapter XI, section II.

Programming tasks

1. Create a Python class for quagmire 2 ciphers that inherits the properties of your class of generic quagmires. Override the `__init__()` function to set $k_1 = e$.

2. Implement a naïve dictionary attack on the quagmire 2 cipher that nests two loops over a word list.

Exercises

1. Factor the quagmire 2 cipher used in the example above, and find the keys of the two factor ciphers.

2. Encipher this text with the keywords `CHARLES` (alphabet) and `DICKENS` (shifts).

   > It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness, it was the spring of hope, it was the winter of despair...
   >
   > (from *A Tale of Two Cities* by Charles Dickens)

3. Decipher this ciphertext with the keywords `PLANET` (alphabet) and `EARTH` (shifts).

```
FAFSTXGTYRDWTCRESHFKTAUMCEVWWNRCHIMYMXTNHGBRHTJWFYCOXYRDYWXEDSTX
PISROHVKBGEHBYTWEZOOSDURXXEJNZYVBYTACBVNMMJBLNYMBWIIVPRQXFPXHXAV
JNXATCMRCFSTGFNNYIMWYFXYBQSMJPSYOGNFNIURRTXVWWWAGTXSGMPGSBATYWIV
HOMTJIFQVBWRPMATFEPFBXMEQWDMWEWRKDTNJODCBWXWAFFNRAETGIMYOFSSPQSX
GJFEHAHYRDPGYSMJHISQIVJQIVROCEVUIMWAFFHSSWYEAMWTEITWT
```

4. Break this ciphertext with a dictionary attack. Both keywords are common five-letter English words.

```
CTXMSTARXSYRSWGOUKWBRPEHCNHCRNZDXJRDAAPSCUWGQTAPUJOLGXDGPEJACUWJ
DZRIZPYOKHBCNSWBGDEPBCZTNCHDOMUKFRUOYODESYOIEWZIGGBZHIICVIBGSJMT
SSSLOHQGDTMSDAVFRZMMNNJNTNBTZPMIGDTMXCQHGBNAZGSWTOOBHDEJZCNSMSWO
BOBWAOBBMOGZSULMJTNHCKAVBXOTVSRUECSDESQATKBTABPBGHQWZSZOOVVMTJAG
OCKSCKRJAKPBJEKSGUBGRMNGKDTMSCUWJGKVIBSWUUJAHRGCLVVDENGKTGEONVRI
YRVGRGLTDNUOUDLDHBOVGDOOQOBUGAZNEOTYOROVKTDWZSHIIILRIJKNHCEIYTTN
```

NYAPMSJITOCKSUJADACFQKTDENGUWGLCSTRNOSKFCNUWGWYBTGQONVOVKZOMSGRH
IVGUWGWYMOHNMAOHQGODENGAIZAOFTMBJAOMZADXMSJIHOWNHCXAVUWGAMRXHRVU
WFRMVIZSYDXDORUVMIEUWRIYUWGJNHQEVOTOHWTRGNIUDXNQRNVEWSLCWAKRCDDH
DPJDUFDJVOMONWDACFQKTDEWZNEOVONVOVCUWGBVPGGRGHUPZSZVFHKTRGCNACJG
KUWFRMUWWCURDPAKCOQSTAWFBHOHHCKNOHPKTTGGZUEONQAUMITMTESSADNGKBTG
QEHDEVOTWFRJTPWAZMTOVKPGWPKOUWOYODESYRIZNZMHYVCTXHVOMGFSZMXHSZMM
ISUUXCWGZKFCNMMGQREOOWVNDEWZHCFCCFODCNRDIAOAJGBTAOHQGDTNEOSXOILM
MRNGDTMJNHQEWGDXHYOTKFCNHIDSDAVFRZUEDDGHIUNMRXHBGUWFBZABQWGVSGWE
HGCRVLEHUKSTHQRSTFCGDEPUNZTOWQNESRVXXNSYARGQADESCVRJJWJHD

key letter	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
A	QUAGMIREBCDFHJKLNOPSTVWXYZ
B	UAGMIREBCDFHJKLNOPSTVWXYZQ
C	AGMIREBCDFHJKLNOPSTVWXYZQU
D	GMIREBCDFHJKLNOPSTVWXYZQUA
E	MIREBCDFHJKLNOPSTVWXYZQUAG
F	IREBCDFHJKLNOPSTVWXYZQUAGM
G	REBCDFHJKLNOPSTVWXYZQUAGMI
H	EBCDFHJKLNOPSTVWXYZQUAGMIR
I	BCDFHJKLNOPSTVWXYZQUAGMIRE
J	CDFHJKLNOPSTVWXYZQUAGMIREB
K	DFHJKLNOPSTVWXYZQUAGMIREBC
L	FHJKLNOPSTVWXYZQUAGMIREBCD
M	HJKLNOPSTVWXYZQUAGMIREBCDF
N	JKLNOPSTVWXYZQUAGMIREBCDFH
O	KLNOPSTVWXYZQUAGMIREBCDFHJ
P	LNOPSTVWXYZQUAGMIREBCDFHJK
Q	NOPSTVWXYZQUAGMIREBCDFHJKL
R	OPSTVWXYZQUAGMIREBCDFHJKLN
S	PSTVWXYZQUAGMIREBCDFHJKLNO
T	STVWXYZQUAGMIREBCDFHJKLNOP
U	TVWXYZQUAGMIREBCDFHJKLNOPS
V	VWXYZQUAGMIREBCDFHJKLNOPST
W	WXYZQUAGMIREBCDFHJKLNOPSTV
X	XYZQUAGMIREBCDFHJKLNOPSTVW
Y	YZQUAGMIREBCDFHJKLNOPSTVWX
Z	ZQUAGMIREBCDFHJKLNOPSTVWXY

Table 67.1: Tableau for our example quagmire 2 cipher that uses alphabetic keyword `QUAGMIRE`.

Unit 68
Better dictionary attack on quagmire 2

In Task 2 of Unit 67 we asked you to implement a dictionary attack on Q2. That attack would try all possible pairs of keywords. If your dictionary has N words in it, then this attack takes time and effort that is proportional to N^2. In this unit we will show you how to attack the Q2 by passing only once through your word list, so that the time and effort is only proportional to N.

Recall that the quagmire 2 can be factored into a Vigenère followed by a monoalphabetic substitution; as we wrote in Equation 67.1:

$$Q_2(k_2, k_3) \;=\; S(k_3)\; V(k_2')$$

where

$$k_2' \;=\; S^{-1}(k_3)\; k_2$$

And remember that in Unit 52 we saw a neat way to attack the Vigenère by rotating the monogram frequencies of each slice until we get a match to English frequencies. That attack on the Vigenère took very little effort. We can combine these ideas to build a better dictionary attack on the Q2. The idea is this: for each word in our list, we decipher the ciphertext with a keyword substitution cipher, and then evaluate the result's fitness as a Vigenère (its Vigenère-ness). The keyword used in the substitution that gives the best V-ness. Note that we could just as well complete the decryption for each substitution keyword, but then evaluating the plaintext with tetragram fitness is more computationally expensive.

We need an operational definition of V-ness. One way is to find the monogram frequencies of each slice and rotate them to align with English frequencies, as we did in Unit 52, like this:

```
function Vness (text, period):
    cut the text into slices s[0], ..., s[period−1]
    for each s[i]:
        find its monogram frequencies f[i][j] where j runs over the alphabet
    set sum to 0
    for each i in 0, ..., period:
        set best to 0
        for j in 0, ..., 25:
            set temp equal to f[i] rotated by j steps
            set d equal to the cosine of the angle between f_English and temp
            if d > best, then set best equal to d
        add best to sum
    return sum / period
```

To find the cosine of the angle between two vectors, use Equation 9.1:

$$\cos\theta = \frac{\mathbf{U}\cdot\mathbf{V}}{\sqrt{(\mathbf{U}\cdot\mathbf{U})(\mathbf{V}\cdot\mathbf{V})}}$$

Let us work look at an example. Suppose we have this ciphertext:

```
GUUISXKNQIIBKMGLRKCHNPSDFUDRPBJPESVIRKHVOGSDSERTDGKOKTHFAUH
HWGSNYJWNQIFXZDRDAJDFWADOFKGSQIIBKNHIYSQIWMEPCSWFORLFBORLXX
YXKYSCPZKUKXDGRPLIUFKJXKKPIIRDRXCGXPHWXDRYJGDDHNZFTCZWKJRWU
NVSKGSFSLGSDRDDRPHJOSPNWWFTCEWUOCTFDFSEEIDDIVRIYMWDRXUYRMZN
BDOXEFIHWJYECIWLYOFKYSCPZKUKXDGRVSKQIINNVJOWNMZOKIMDOXEWTOR
LCFYDOWRHWAPJHGFMZNSBPIPXEEPYGORKCHNPSDFUGSPXPYSQXIMGYZLCGY
KIDFIGLYXPSBYXYVWMEYVLGUHSJCTDGCGXPHNVZSXDGSOCTFDOXEWTORLCF
YDOW
```

The first step is to find the period, which we do with Sinkov's method:

Next we run over all words in our dictionary, decipher as a monoalphabetic substitution, and find the V-ness of the result. Below is an histogram of those values for a dictionary with over 500,000 words. The vertical axis is logarithmic, so that the one good datum is visible. The best value is for the keyword `DICTIONARY`. From this histogram we see that near the top of the distribution there is only a narrow gap to put a cutoff. A solution to this issue is to keep track of the highest V-ness and whenever a new high is found, decrypt completely and evaluate the plaintext with tetragram fitness in order to decide if the search for the keyword is finished. This does not add much time to the attack, because we only need to check tetragram fitness about a dozen times.

After the ciphertext is deciphered with the keyword `DICTIONARY`, we have this intermediate text:

```
MVVBUYPFTBBJPRMQHPCNFSUALVAHSJOSKUWBHPNWEMUAUKHDAMPEPDNLGVN
NXMUFIOXFTBLYZAHAGOALXGAELPMUTBBJPFNBIUTBXRKSCUXLEHQLJEHQYY
IYPIUCSZPVPYAMHSQBVLPOYPPSBBHAHYCMYSNXYAHIOMAANFZLDCZXPOHXV
FWUPMULUQMUAHAAHSNOEUSFXXLDCKXVECDLALUKKBAABWHBIRXAHYVIHRZF
JAEYKLBNXOIKCBXQIELPIUCSZPVPYAMHWUPTBBFFWOEXFRZEPBRAEYKXDEH
QCLIAEXHNXGSONMLRZFUJSBSYKKSIMEHPCNFSUALVMUSYSIUTYBRMIZQCMI
PBALBMQIYSUJIYIWXRKIWQMVNUOCDAMCMYSNFWZUYAMUECDLAEYKXDEHQCL
IAEX
```

To this we apply the attack of Unit 52 on the Vigenère cipher and obtain the key k_2' = `XEHAU`. Inverting equation 60.2 gives us

$$k_2 \;=\; S(k_3)\, k_2' \;=\; \texttt{WORDS}$$

We now have both keywords for the Q2 cipher, and the plaintext is from *1000 Things Worth Knowing* by Nathaniel C. Fowler, Jr.:

> Probably the first dictionary was produced in China, and was said to contain 40,000 [forty thousand] characters. In the 15[fifteen]th and 16[sixteen]th centuries, encyclopedias were published. The first authoritative dictionary was in Latin, and was translated into eight languages about the year 1500 [fifteen hundred]. Chamber's Encyclopedia was published in 1728 [seventeen twenty-eight], and Johnson's famous English dictionary in 1755 [seventeen fifty-five]. Webster's American dictionary was first issued in 1828 [eighteen twenty-eight].

Programming tasks

1. Implement the attack. Keep track of the highest V-ness seen so far. Each time you find a new highest value, decrypt completely and use tetragram fitness to decide if the search is over.

Exercises

1. Use the attack to break this ciphertext. Recover both keywords.

```
WUQKFVHCMDGXPLJGYYFRIQDWGNYFRGJYWGYYXXXTPKSDMPSHBNYITKXXESF
UVFIBYYRDYOKMIEGDIMDRWIVGVEHHTJDBCBWXHPIHIAUSPCNTBYFQXLOBLP
QJTDXXIIGRDSVVHPGMBXPIJHIYWXQWDHHTJDBCVOSDRYNNAZIEESLDAJVOJ
HTIGRGSVTDPIPSMDHWGYIXEXUXHXXMBYFQXLWDVDIYUCDSCZBBCBWIHTKHD
HMPYZRLVRGVXNHBPEYYAHWYUIHRGUOXXXMAIVNHVDEEIDQHCRLDGCNFVXUI
BIGIJEGKOQMSBCVALSPVNDHUOWDDZBYHQQGPQVABUMOJGRHPITXIHITVOQE
SVJDQTYPSVWHHBYDPXQWRSYWLUQTKYELGOKEAXUDRWDLXKXHCXQAXMMYVEL
GKRHDIJWFJARGQMRWBQOWQKEWIFTDBCVWVITOEYSRHDONLLBELDGDWGSQXI
IHITGYBCMSLXKGOQQXPGKINSRYVSYOPMOIHJDGDUBILEKFJGRVORWSARXRK
VFHUNDOWCDIDHDDBPVMBUWIJDQIIDHVWROQQOAJBSLOWXHTWQ
```

Unit 69
Attacking quagmire 2 with cribs: symmetry

Much like we did in Unit 64, we are going to focus on what we can learn only from a pairing of a crib with its segment of ciphertext. But for Q2 we keep the plaintext alphabet in order and the rows are rotated versions of each other. This is our example:

```
0123 0123012 301 230123012 3012 30123012
THIS MESSAGE WAS ENCRYPTED WITH QUAGMIRE
FJLG EDQGHHI MHE IUKVVBFDA MWRU CUUFFWVI
```

Here is what the crib tells us about the key table:

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	H.K.....W...E.....FU.....
1	U...D.HJ.........VER.....
2	...AI.FUL.........Q.....V.
3	............FU.BC.G...M...

We may as well start with A, which appears in row 2. There we have the distances to the other six letters:

AI = 1
AF = 3
AU = 4
AL = 5
AQ = 15
AV = 21

There is no A in row 3, but we can see that

FB = 3
FC = 4
FG = 6
FM = 10

Since AB = AF + FB, etc. (all addition must be done modulo 26), we get

AB = AF + FB = 3 + 3 = 6

$$
\begin{aligned}
\mathtt{AC} &= \mathtt{AF} + \mathtt{AC} = 3 + 4 = 7 \\
\mathtt{AG} &= \mathtt{AF} + \mathtt{FG} = 3 + 6 = 9 \\
\mathtt{AM} &= \mathtt{AF} + \mathtt{FM} = 3 + 10 = 13
\end{aligned}
$$

We don't have an A in row 1, but we do have an U that allows us to write

$$
\begin{aligned}
\mathtt{UD} &= 4 \\
\mathtt{UH} &= 6 \\
\mathtt{UJ} &= 7 \\
\mathtt{UE} &= 18 \\
\mathtt{UR} &= 19
\end{aligned}
$$

So

$$
\begin{aligned}
\mathtt{AD} &= \mathtt{AU} + \mathtt{UD} = 4 + 4 = 8 \\
\mathtt{AH} &= \mathtt{AU} + \mathtt{UH} = 4 + 6 = 10 \\
\mathtt{AJ} &= \mathtt{AU} + \mathtt{UJ} = 4 + 7 = 11 \\
\mathtt{AE} &= \mathtt{AU} + \mathtt{UE} = 4 + 18 = 22 \\
\mathtt{AR} &= \mathtt{AU} + \mathtt{UR} = 4 + 19 = 23
\end{aligned}
$$

Row 0 also lacks an A but has an U. From it we have

$$
\begin{aligned}
\mathtt{UK} &= 8 \\
\mathtt{UW} &= 14
\end{aligned}
$$

and so

$$
\begin{aligned}
\mathtt{AK} &= \mathtt{AU} + \mathtt{UK} = 4 + 8 = 12 \\
\mathtt{AW} &= \mathtt{AU} + \mathtt{AW} = 4 + 14 = 18
\end{aligned}
$$

If we start with A and place as many letters as we can, we have

AI.FUL<u>BCDGHJKM.Q..W</u>..VER..

The underlined segment appears to be part of the portion of the alphabet where letters are placed after the keyword, since they are in alphabetical order. The missing letters are N, O, P, S, T, X, Y, Z. The only thing that can go between Q and W is ST, and if we guess that the very infrequent letters X and Z should go after W, we have

AI.FUL<u>BCDGHJKM.QSTWXZ</u>VER..

The missing letter list is now N, O, P, Y. A search of a word list for AI.FUL at the end of a word turns up these:

AIDFUL	PREPAINFUL
AIMFUL	RAINFUL
AVAILFUL	STAINFUL

```
    DAINFUL                TRAINFUL
 DESPAIRFUL             UNAVAILFUL
 DISDAINFUL              UNGAINFUL
    GAINFUL              UNPAINFUL
OVERPAINFUL                VAINFUL
    PAILFUL                WAILFUL
    PAINFUL                WAINFUL
   PLAINFUL
```

Since some letters appear in the alphabetically ordered segment or after `.AI.`, we know they cannot be in the positions before or after `AI`. The possibilities are ...`PAINFUL` and `VAINFUL` (the `V` appears *before* `.AI.`, so could be a dropped duplicate letter). Let's take `PAINFUL`, so that we have

`VER.PAINFUL`<u>`BCDGHJKM.QSTWXZ`</u>

The only remaining letters are `O` and `Y`, so it appears that the keyword is `VERYPAINFUL`.

Reading and references

William F. Friedman, The Principles of Indirect Symmetry of Position in Secondary Alphabets and Their Application in the Solution of Polyalphabetic Substitution Ciphers, Washington D.C.: U.S. Government Printing Office, 1935, www.nsa.gov/Portals/70/documents/news-features/declassified-documents/friedman-documents/publications/FOLDER_226/41760669079979.pdf

Programming tasks

1. Automate the process and do your best to get your program to fill in missing letters based on alphabetical ordering or dictionary searches.

Exercises

1. Break this ciphertext. The plaintext begins with "His mission was to deliver one of the Terran Empire's elite, ..."

```
CBWKDUQFNKYHWWMTIKEXANQMAMTXDAVIVUSKIMQDRIWLGDRBNKAJCWCAAVL
AKUNELMNMLTRHNCARQWPLAESWMVLBVWNNBKASGLBDRQXVNAHJLRMGENJAWQ
UGTHKWCMUDDVCHXVMWGLLTUBMQGAINNWBLFWYMUKAVSDBIMVLCERAKQZARH
NAVRBAGARY
```

Unit 70
Attacking quagmire 2 with cribs: group theory

From Equation 67.2 we know that the rows of a Q2 key table are rotated versions of the cipher key:

$$q_n = k_3 \circ R_n$$

Here again is the example from the previous unit:

```
0123 0123012 301 230123012 3012 30123012
THIS MESSAGE WAS ENCRYPTED WITH QUAGMIRE
FJLG EDQGHHI MHE IUKVVBFDA MWRU CUUFFWVI
```

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	H.K.....W...E.....FU.....
1	U...D.HJ.....VER.....
2	...AI.FUL.....Q.....V.
3	.....FU.BC.G...M...

Let's rotate the four rows to line them up:

0	H.K.....W...E.....FU.....
1′	HJ.....VER.....U...D.
2′	.....Q.....V....AI.FUL....
3′	...M.....FU.BC.G

Merge them together to get:

<u>HJKM.Q..W</u>..VER..AI.FUL<u>BCDG</u>

The underlined segment appears to be part of the portion of the key that is in alphabetcal order. The missing letters are N, O, P, S, T, X, Y, Z. Filling them into the key can be done exactly as in the previous unit.

Alternatively, we can unrotate and put the rows back into the key table. From there, we can continue decrypting the text beyond the crib with traditional techniques.

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	HJKM.Q..W..VER..AI.FULBCDG
1	ULBCDGHJKM.Q..W..VER..AI.F
2	R..AI.FULBCDGHJKM.Q..W..VE
3	.W..VER..AI.FULBCDGHJKM.Q.

Reading and references

Thomas Kaeding, “Classical substitution ciphers and group theory,” Cryptology ePrint Archive, report 2023/669.

Programming tasks

1. Automate the process, at least up to the point where merging has taken place.

Exercises

1. Repeat Exercise 1 from Unit 69.

Unit 71
Quagmire 2: recovering keywords

Suppose that we have solved a ciphertext encrypted with a Q2 cipher, and that we have a completely filled key table. To recover the keywords, we simply read them from the table. This works because each of them has the form

$$q_n = k_3 \circ R_n$$

which is a rotation of the key used to construct the quagmire.

Here is the example from Unit 67:

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	CDFHJKLNOPSTVWXYZQUAGMIREB
1	IREBCDFHJKLNOPSTVWXYZQUAGM
2	PSTVWXYZQUAGMIREBCDFHJKLNO
3	HJKLNOPSTVWXYZQUAGMIREBCDF
4	EBCDFHJKLNOPSTVWXYZQUAGMIR
5	REBCDFHJKLNOPSTVWXYZQUAGMI

The shift key is found in a column of the table. That column is under A in the plaintext alphabet unless we made some choice of index letter other than the first letter of the alphabet keyword. The alphabet keyword appears in each row of the table, and rows are rotations of each other and rotations of the scrambled alphabet used by the Q2 cipher. In this example, that alphabet is

$$k_3 = \texttt{QUAGMIREBCDFHJKLNOPSTVWXYZ}$$

Unit 72
Quagmire 3 cipher

The *quagmire 3 cipher* (also called *polyalphabetic type 3*) uses a mixed alphabet for both the plaintext and the ciphertext. The mixing is generated from a keyword. The ciphertext alphabets are shifted versions of the mixed alphabet, and the shifts form another keyword, as they do in the Vigenère cipher.

Here is an example. The keywords are `QUAGMIRE` and `CIPHER`. First, look at the table of key alphabets. Notice that the shift keyword appears under the first letter of the alphabet key, unless we chose a different indexing letter and used sliding alphabets to define the cipher.

	plaintext alphabet QUAGMIREBCDFHJKLNOPSTVWXYZ
0	CDFHJKLNOPSTVWXYZQUAGMIREB
1	IREBCDFHJKLNOPSTVWXYZQUAGM
2	PSTVWXYZQUAGMIREBCDFHJKLNO
3	HJKLNOPSTVWXYZQUAGMIREBCDF
4	EBCDFHJKLNOPSTVWXYZQUAGMIR
5	REBCDFHJKLNOPSTVWXYZQUAGMI

It is customary to rearrange the columns so that the plaintext letters are in alphabetical order:

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	FOPSNTHVKWXYJZQUCLAGDMIREB
1	EJKLHNBODPSTCVWXIFYZRQUAGM
2	TQUAZGVMXIREWBCDPYFHSJKLNO
3	KTVWSXLYOZQUNAGMHPIRJEBCDF
4	CLNOKPDSHTVWFXYZEJQUBAGMIR
5	BKLNJOCPFSTVDWXYRHZQEUAGMI

Now we encipher a short message with this key table. The center row indexes the key alphabet.

```
THIS MESSAGE IS ENCRYPTED WITH A QUAGMIRE CIPHER
CIPH ERCIPHE RC IPHERCIPH ERCI P HERCIPHE RCIPHE
GOXI FJAYTLK FA HBVJMUZZW GFGO T HBBHCXPK LKXMSJ
```

There are some special cases of the quagmire 3 cipher:

- period = 1: monoalphabetic substitution (the "Caesar salad" cipher of Unit 25)
- period = 1 and shift keyword = first letter of alphabet keyword: no encryption
- mixed alphabet = regular alphabet: Vigenère cipher
- mixed alphabet generated as in affine cipher: Vigenère cipher (whose key is a Caesar shift of the quagmire's shift keyword)

We saw earlier that a quagmire can be factored into a Vigenère sandwiched between two monoalphabetic substitution ciphers. With the Q3, the two substitutions are inverses of each other:

$$Q_3(k_1, k_2) \;=\; S(k_1)\, V(k_2')\, S^{-1}(k_1) \tag{72.1}$$

where

$$k_2' \;=\; S^{-1}(k_1)\, k_2$$

We already have seen that the set of alphabetic permutations of the Vigenère cipher form a group that is isomorphic to $(\mathbb{Z}_{26}, +_{26})$ (see Equation 47.2 and the discussion preceding it). The Q3 permutations are isomorphic to the Vigenère. Define the function by

$$\varphi\colon\; R_n \;\rightarrow\; k_1 \circ R_n \circ k_1^{-1}$$

Then we see that it is an isomorphism:

$$\begin{aligned}
\varphi(R_m) \circ \varphi(R_n) \;&=\; (k_1 \circ R_m \circ k_1^{-1}) \circ (k_1 \circ R_n \circ k_1^{-1}) \\
&=\; k_1 \circ R_m \circ (k_1^{-1} \circ k_1) \circ R_n \circ k_1^{-1} \\
&=\; k_1 \circ R_m \circ e \circ R_n \circ k_1^{-1} \\
&=\; k_1 \circ (R_m \circ R_n) \circ k_1^{-1} \;=\; \varphi(R_m \circ R_n)
\end{aligned}$$

Since isomorphism is transitive, this means that the set of Q3 permutations is isomorphic to $(\mathbb{Z}_{26}, +_{26})$:

$$(Q_3[k_1], \circ) \;\cong\; (\mathbb{Z}_{26}, +_{26}) \tag{72.2}$$

where

$$Q_3[k_1] \;=\; \{k_1 \circ R_n \circ k_1^{-1}\} \tag{72.3}$$

One consequence of this isomorphism is that the orders of the permutations in the set are 1, 2, 13, and 26, where twelve have order 13 and twelve have order 26. The permutation with order 1 is the identity: e. That's right, the unmixed alphabet appears in the tableau of every quagmire 3. One permutation has order 2, i.e., it is self-reciprocal (involutory). Another useful consequence is that any one of the order-26 permutations can be used to generate the entire tableau, in the same way that 1 or 3 or 5 or ... can be used to generate $\mathbb{Z}_{26}$. Such an element is called a *generator*. (often written g). The way it works is to take powers of the generator; in the case of alphabetic permutations those powers are compositions: $g \circ$

$g \circ g \circ g \circ g$ .... And if we don't have an order-26 permutation handy, we can make one by composing the order-2 permutation with any order-13 permutation, in the same way that 13 + 8 = 21, for example, is a generator of $\mathbb{Z}_{26}$. If we have forgotten to mention anything else useful, just keep in mind that whatever you can do with integers and addition modulo 26 can be done with the permutations of a quagmire 3 cipher and the composition operation. Commutivity holds in $Q_3[k]$ as well.

Suppose now that we have solved a ciphertext, but do not know which polyalphabetic cipher was used to encrypt it. We might, for example, have the following key table after finding all the plaintext letters in the text. How can we determine whether or not the cipher is a Q3, assuming that the cipher is a quagmire or other periodic polyalphabetic cipher that we have studied?

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	FOPSNTHVK.XYJZQU.LAGDMI.E.
1	EJKLHNBO..STCVWX.FYZRQU.G.
2	TQUAZGVMX.REWBCD.YFHSJK.N.
3	KTVWSXLYO.QUNAGM...RJEB.D.
4	CLNOKPDSH.VWFXYZ.JQUBAG...
5	BKLNJOCP..TVDWXY.HZQEUA.M.

Because of the isomorphism to $\mathbb{Z}_{26}$, we know that the composition of any two permutations must be another permutation in the set in Equation 72.3, and that the inverse of any permutation must also be in the set. We only have six partial permutations in our example above, so we may have to try various combinations until we find matches. Taking the composition of rows 1 and 4 gives

KTVWSXLYO.QUN.G....RJEB...

This is an obvious match to row 3. The composition of row 0 with itself is

TQUAZGVMX..E...D.YFHSJK.N.

which is a match to row 2. The inverses of the six that we have are not in the key table. And there are too many holes to calculate the order of any of them (the number of holes increases with each composition). Later we will see how to fill in those holes. Nevertheless, the matches that we have found so far indicate strongly that we are dealing with a quagmire 3.

Reading and references

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; chapter XVIII.

American Cryptogram Association, www.cryptogram.org/downloads/aca.info/ciphers/QuagmireIII.pdf

Thomas Kaeding, "Classical substitution ciphers and group theory," Cryptology ePrint Archive, report 2023/669.

Programming tasks

1. Write a function that takes the keywords for a quagmire 3 cipher and generates the key alphabets for the periodic polyalphabetic substitution cipher.

2. Implement the quagmire 3 cipher and given keywords. You may use the function from Task 1, but there are other ways to accomplish this.

3. Write a function or script to perform a dictionary attack on the quagmire 3 cipher.

4. Write a function that takes the two quagmire 3 keywords and outputs the keys of the substitution ciphers and Vigenère cipher into which the quagmire can be factored.

Exercises

1. Factor the quagmire 3 cipher in the example above, and find the keys of the factor ciphers.

2. Encipher this text with the keywords `NURSERY` (alphabet) and `RHYME` (shifts).

   The cat and her kittens they put on their mittens, to eat a Christmas pie.
   The poor little kittens they lost their mittens, and then they began to cry.

3. Decipher this ciphertext with the keywords `DOLPHINS` (alphabet) and `FISHBOWL` (shifts).

```
UXDAPTKWHERHJUJPNESLUUHKUCGFKOKNQFDBZFKNXSFQGUAUGVDAUOMBIPB
BLUYGVMSEJOSHUVCJQVBNICDFLBWSYQRWUSKCRRBKQSWKJCDXUTSLGEJIZF
ASRMKGOBPSGEJIKUDGYCJARLRHTFOWVBNGMMEKEOOVNAKHXVPVQEUXGFHAZ
SRXCUDGJCUALBLLTZSHUSYDTGPJEFHUJPDAUGOAXSIKKJJCSMSEGBZGTNTD
CTWBTBKHCXSLGEDAUJYCTFDAUZPCQIJIELLKUESXGBLQT
```

4. Break this ciphertext with a dictionary attack. Both keywords are common five-letter English words.

```
OHSRHJIURBYNPSWGTIMBOHFCMYNHECDEBJVUBPAAUNEJIOHFOCJLIMTVEFK
JORPKWYROCJGNAACELEFXFECRHRLPKWYR
```

key letter	**plaintext alphabet** QUAGMIREBCDFHJKLNOPSTVWXYZ
Q	QUAGMIREBCDFHJKLNOPSTVWXYZ
U	UAGMIREBCDFHJKLNOPSTVWXYZQ
A	AGMIREBCDFHJKLNOPSTVWXYZQU
G	GMIREBCDFHJKLNOPSTVWXYZQUA
M	MIREBCDFHJKLNOPSTVWXYZQUAG
I	IREBCDFHJKLNOPSTVWXYZQUAGM
R	REBCDFHJKLNOPSTVWXYZQUAGMI
E	EBCDFHJKLNOPSTVWXYZQUAGMIR
B	BCDFHJKLNOPSTVWXYZQUAGMIRE
C	CDFHJKLNOPSTVWXYZQUAGMIREB
D	DFHJKLNOPSTVWXYZQUAGMIREBC
F	FHJKLNOPSTVWXYZQUAGMIREBCD
H	HJKLNOPSTVWXYZQUAGMIREBCDF
J	JKLNOPSTVWXYZQUAGMIREBCDFH
K	KLNOPSTVWXYZQUAGMIREBCDFHJ
L	LNOPSTVWXYZQUAGMIREBCDFHJK
N	NOPSTVWXYZQUAGMIREBCDFHJKL
O	OPSTVWXYZQUAGMIREBCDFHJKLN
P	PSTVWXYZQUAGMIREBCDFHJKLNO
S	STVWXYZQUAGMIREBCDFHJKLNOP
T	TVWXYZQUAGMIREBCDFHJKLNOPS
V	VWXYZQUAGMIREBCDFHJKLNOPST
W	WXYZQUAGMIREBCDFHJKLNOPSTV
X	XYZQUAGMIREBCDFHJKLNOPSTVW
Y	YZQUAGMIREBCDFHJKLNOPSTVWX
Z	ZQUAGMIREBCDFHJKLNOPSTVWXY

Table 72.1: Tableau for our example quagmire 3 cipher that uses alphabetic keyword `QUAGMIRE`.

key letter	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
A	MDFHCJIKELNORPSTABVWGXYZQU
B	DNOPLSFTJVWXHYZQBKUACGMIRE
C	FOPSNTHVKWXYJZQUCLAGDMIREB
D	HPSTOVJWLXYZKQUADNGMFIREBC
E	CLNOKPDSHTVWFXYZEJQUBAGMIR
F	JSTVPWKXNYZQLUAGFOMIHREBCD
G	IFHJDKRLBNOPESTVGCWXMYZQUA
H	KTVWSXLYOZQUNAGMHPIRJEBCDF
I	EJKLHNBODPSTCVWXIFYZRQUAGM
J	LVWXTYNZPQUAOGMIJSREKBCDFH
K	NWXYVZOQSUAGPMIRKTEBLCDFHJ
L	OXYZWQPUTAGMSIRELVBCNDFHJK
M	RHJKFLENCOPSBTVWMDXYIZQUAG
N	PYZQXUSAVGMITREBNWCDOFHJKL
O	SZQUYATGWMIRVEBCOXDFPHJKLN
P	TQUAZGVMXIREWBCDPYFHSJKLNO
Q	ABCDEFGHIJKLMNOPQRSTUVWXYZ
R	BKLNJOCPFSTVDWXYRHZQEUAGMI
S	VUAGQMWIYREBXCDFSZHJTKLNOP
T	WAGMUIXRZEBCYDFHTQJKVLNOPS
U	GCDFBHMJRKLNIOPSUETVAWXYZQ
V	XGMIARYEQBCDZFHJVUKLWNOPST
W	YMIRGEZBUCDFQHJKWALNXOPSTV
X	ZIREMBQCADFHUJKLXGNOYPSTVW
Y	QREBICUDGFHJAKLNYMOPZSTVWX
Z	UEBCRDAFMHJKGLNOZIPSQTVWXY

Table 72.2: Tableau for our example quagmire 3 cipher that uses alphabetic keyword `QUAGMIRE`*. In this instance of the tableau, rows and columns have been rearranged so that the plaintext letters and key letters are in alphabetical order. The tableau is nevertheless equivalent to the one in Table 72.1.*

Unit 73
Better dictionary attack on quagmire 3

This attack is similar to the one in Unit 68 on the Q2 cipher. When we look at the factorization (see Equation 72.1) of the quagmire 3, we see a Vigenère between substitutions:

$$Q_3(k_1, k_2) \;=\; S(k_1)\, V(k_2')\, S^{-1}(k_1)$$

where (from Equation 60.2)

$$k_2' \;=\; S^{-1}(k_1)\, k_2$$

When we remove the last substitution (the one to the left of the Vigenère), what remains is a Q1 (Equation 62.1):

$$Q_1(k_1, k_2') \;=\; V(k_2')\, S^{-1}(k_1)$$

(The prime mark reminds us that the k_2 of the Q1 is different from the k_2 of the Q3.) This complicates things, compared to Unit 68. Instead of scoring a word by the Vigenère-ness of the remainder, we have now to define a Q1-ness, like this:

```
function Q1ness (text, period):
    cut the text into slices s[0], ..., s[period−1]
    for each s[i]:
        find its monogram frequencies f[i][j] where j runs over the alphabet
    set sum to 0
    for each i > 0:
        set best to 0
        for j in 0, ..., 25:
            set temp equal to f[i] rotated by j steps
            set d equal to the cosine of the angle between f[0] and temp
            if d > best, then set best equal to d
        add best to sum
    return sum / (period − 1)
```

The difference between this function and V-ness is that, because of the substitution that remains in the Q1, we can only compare slices to one another, and not to English.

When we find a word with a high Q1-ness, we can then check whether we have found the solution by continuing with the attack on Q1 from Unit 63, except that *we already know the substitution key*, because it is the same one based on the word in question. There is still an overall shift that has not been determined, so really we need to check twenty-six possibilities.

The same considerations from Unit 68 about the inability to set a clear cutoff apply here: When we find a word with high Q1-ness, it is safest to complete the decryption and evaluate the text with tetragram fitness. Only then can we know if the search is finished.

Programming tasks

1. Implement the Q1-ness function and put it somewhere so that it can be used in this attack and again in a later attack.

2. Implement the attack.

Exercises

1. Break this ciphertext from the British National Cipher Challenge 2018. Each letter of the shift key will be off by one because the creators of the challenge were thinking about the Hutton cipher (see Unit 154).

```
      TSTJAN FP JON PVZDYPRFXTFEX, NQWKEJLOPJKHQ WYJ VZDEBRZB FG
JON FXCOVE GADBJLI; KTNB TSGJYIKPUWATSUH. HCH CSST SHNF SUMN.
      XFTV YFFCHD HCH ZCXYCWHDCYPZ IZ JON QSYFDKY FOJTSXGAT ZW
IIZKMZ WE XFP VZTKGAOMKH TRI IDZJLLATOW WH XFP LSDQL-RZZKQSRIPJZ
JEXNXGNISRMP YMFEGB, DXK HRWIKH CTZORWO, BFIIGQNKT IIPPMYFU XJ
IV ZAS WHNXNXS. XFP MKQVVA NSUUTLYF MKEVQXPJEDJLJZV LWI WMDCFPQ
GJZDKTELYR NBF TVMGCHF, TD KPGR DQ W EOITDIH JQ GKHQCYF GKDIRNJ
WYWC WYIGDHDD JQ CZH TGATQNUDIE. TN CMQC MKJZDDRE I AYJSJ
EOITDIH JQ NBF ZYHAOUU TDEPNW WH XFP YYERLOP YUO IIMKITFEBGDCFPQ
WE DS ZAS YEP SY ODDZKVCM VJ AGSWF TDEPNW.
      URI EGYEWX WHNXNXS LE IJ NPUIZPJEFEX RZNKMBLQPJJF TKNZDD
HSX DS AD URI ZACHDJI EPDYFJWHC NF ASV XIEKHJH SOOKP LLMDSYJT LY
DXK MTXN PCMAJINZNB ESRAOMD. JJ EGF OWUTFVTVBFU XJ NSUOKGA
FPYXSLVXYUES WYJ NF ETVHC SSU ZCOPMZ ODTVKICUDG GNNNWED IPVNIEII
DS LVIXFPM ZAS LYDOYFQXE KW BFI QGUOWUN'D QKQKTEQNZN, GADPN
SMFXDBRZB CP SPNYOUU VJ ZXYSHDFVP ZKPDWLTGNUN LY DXK FLIYD NBDJ
LAF YJUDZRDCKH XIHP ZNHZCOPMKO. QGGZN EWZYXPJZDJLJZ VZJBP NBCWUQ
JHKI ZADD EDYMF VJ AGO KHJWLXCWACIYD SL URI GADBJLI, AGSSIR
KRWOU URI IPGNETXN ZYZVII JQ CZH VTNAYZJVRE, TN TDN FN DXCU LIHC
WKB QYMG ZFEKQNZNW BSVN PQKT XVRDNKP. JON QOG URWA JS YFCWRZ YYF
YH YINSTS THTQCUS GJYISPDGGDCFPQ WYJ KKTQCYIG YFZCHJV.
```

Unit 74
Attacking quagmire 3 with cribs: symmetry

The approach is similar to that in Units 64 and 69, but now the plaintext alphabet and the rows in the key table are mixed *all in the same way*. Take this example:

```
1234 1234123 412 341234123 4123 41234123 41234
THIS MESSAGE WAS ENCRYPTED WITH QUAGMIRE THREE
BXFK NLWKQWY OQP YYEBAVBLH OSAK JXGJGSBY MPBYW
```

Below is what the crib tells us about the key table. We have used “0” for the plaintext row and numbered the ciphertext rows from 1, for convenience, since we can use all five of them interchangeably.

0	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
1	Q.E....PS...N......BX.....
2	G...L.WX.........BPA......
3	...HY.JKF.........W.....A.
4	....W.......GY.VJ.KM..O...

The trick is to look for rectangles in the key table. Rows 0 and 1 have a rectangle ACQE, and it tells us that the distance from A to C in the Q3 key is the same as from Q to E. We are trying to reconstruct a key table similar to Table 72.1, where plaintext and ciphertext alphabets are mixed in the same way, merely rotated relative to each other. In that table, we can see that the Q3 key also appears (rotated) in each column. So from the rectangle ACQE we can also deduce that the distances A to Q and C to E are the same. (Actually, we are only trying to reconstruct the Q3 key; the table follows from it.) From that one rectangle we have these two relations:

```
AC = QE
AQ = CE
```

The strategy is to collect as many of these relations as we can, and then use them to sort out the contents of the mixed alphabet that is the key of the Q3 cipher.

Here are the relations that we can gleam from the partial key table:

source rows	relations
0, 1	AQ = CE = HP = IS = MN = TB = UX AC = QE AH = QP AI = QS AM = QN AT = QB AU = QX CH = EP CI = ES CM = EN CT = EB CU = EX HI = PS HM = PN HT = PB HU = PX IM = SN IT = SB IU = SX MT = NB MU = NX TU = BX
0, 2	AG = EL = GW = HX = RB = SP = TA AE = GL AG = GW AH = GX AR = GB AS = GP AT = GA EG = LW EH = LX ER = LB ES = LP ET = LA GH = WX GR = WB GS = WP GT = WA HR = XB HS = XP HT = XA

	RS = BP RT = BA ST = PA
0, 3	DH = EY = GJ = HK = IF = SW = YA DE = HY DG = HJ DH = HK DI = HF DS = HW DY = HA EG = YJ EH = YK EI = YF ES = YW EY = YA GH = JK GI = JF GS = JW GY = JA HI = KF HS = KW HY = KA IS = FW IY = FA
0, 4	EW = MG = NY = PV = QJ = SK = WO EM = WG EN = WY EP = WV EQ = WJ ES = WK EW = WO MN = GY MP = GV MQ = GJ MS = GK MW = GO NP = YV NQ = YJ NS = YK NW = YO PQ = VJ PS = VK PW = VO QS = JK

	QW = JO SW = KO
1, 2	QG = PX = BA QP = GX QB = GA PB = XA
1, 3	PK = SF PS = KF
1, 4	NG = BM NB = GM
2, 3	LY = WJ = XK = PW LW = YJ LX = YK LP = YW WX = JK WP = JW XP = KW
2, 4	LW = PK = AM LP = WK LA = WM PA = KM
3, 4	YW = WK

The longest relational chains are the ones for vertical distances between row 0 and any other row; each relates seven pairs. Take the second one, because it has shared letters in some pairs:

AG = EL = GW = HX = RB = SP = TA

We do not know the numerical value of this distance, but we know that if we choose wrong, then we can take every n^{th} letter of our results, where n is 1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25 (the elements of $\mathbb{Z}_{26}^{*}$), and look for on in which we can see the keyword clearly. So we may as well take this distance to be one step, provisionally.

AG = EL = GW = HX = RB = SP = TA = 1

Now, if AG = 1, then GA = 26 − 1 = 25, etc. We can write down some fragments of the key by chaining together letters that are one step apart:

TAGW EL HX RB SP

From this we can also see that TG = AW = 2 and TW = 3, and GA = 26 − AG = 25, etc. From rows 1 and 2 we have QB = GA, so we can place the Q after RG:

```
TAGW EL HX RBQ SP
```

From rows 0 and 4 we have EM = WG, so we can place the M:

```
TAGW MEL HX RBQ SP
```

From rows 0 and 1 we know HI = PS, so we can place the I before HX:

```
TAGW MEL IHX RBQ SP
```

From rows 0 and 4 we have PS = VK, and from rows 1 and 3 we have PS = KF, so we have another fragment.

```
TAGW MEL IHX RBQ SP FKV
```

From rows 2 and 4, LW = AM. But from the first two fragments, AM = WL = −LW. So AM = −AM, and AM can only be 13 (modulo 26). Also from rows 2 and 4, PK = AM.

```
TAGW..........MEL.........
          IHX RBQ
SP...........FKV..........
```

From rows 0 and 4 the vertical relations are EW = MG = NY = PV = QJ = SK = WO, and from the last fragment we have SK = 14, so these are all equal to 14. These allow us to fill in some more letters.

```
TAGW..........MELO........
            IHX
RBQ.............J.........
SP...........FKV..........
```

From rows 1 and 4 we have NG = BM and NB = GM, which allow us to place N.

```
TAGW..........MELO........
            IHX
RBQ............NJ.........
SP...........FKV..........
```

From rows 0 and 4, NQ = YJ, which allows us to place Y.

```
TAGW..........MELO........
            IHX
RBQY...........NJ.........
SP...........FKV..........
```

From rows 0 and 3, GY = JA. And from the third fragment, YJ = 13, and we already know that GA = 25. Since GA = GY + YJ + JA (mod 26), GY + JA = 12. Combining that with GY = JA, we have GY = JA = 6 or 19. We have these two possibilities for combining the first and third fragments:

```
TAGW.RBQY.....MELO..NJ....
TAGW...NJ.....MELORBQY....
```

From rows 3 and 4, `YW` = `WK`. This allows us to combine the last fragment into the two options. They become

```
TAGW.RBQY.SP..MELO..NJ.FKV
TAGW...NJ.FKV.MELORBQY.SP.
```

From rows 1 and 2, `QG` = `PX`. This causes a conflict we when try to merge `IHX` into the first of these two possibilities. So we must reject it and accept the second. Merge `IHX` into it, and we have only this remaining:

```
TAGWIHXNJ.FKV.MELORBQY.SP.
```

From rows 0 and 1 we have `AC` = `QE`, which allows us to place `C`.

```
TAGWIHXNJ.FKV.MELORBQYCSP.
```

Also from rows 0 and 1, `HU` = `PX`, and so we can place `U`.

```
TAGWIHXNJ.FKVUMELORBQYCSP.
```

From rows 0 and 3, `DE` = `HY`, so we can place `D`.

```
TAGWIHXNJ.FKVUMELORBQYCSPD
```

The only letter left is `Z`.

```
TAGWIHXNJZFKVUMELORBQYCSPD
```

Taking every third letter reveals the keyword:

```
TWXZVERYPAINFULBCDGHJKMOQS
```

And this time it’s true.

Reading and references

William F. Friedman, The Principles of Indirect Symmetry of Position in Secondary Alphabets and Their Application in the Solution of Polyalphabetic Substitution Ciphers, Washington D.C.: U.S. Government Printing Office, 1935, www.nsa.gov/Portals/70/documents/news-features/declassified-documents/friedman-documents/publications/FOLDER_226/41760669079979.pdf

Programming tasks

1. Implement the procedure. Good luck.

Exercises

1. Find the final word. Do not look up the plaintext from *The Secret Agent: A Simple Tale* by Joseph Conrad until you have solved the problem.

```
ciphertext:  MHIVIIHCBZBJZNYDICNTKRMEUGJBPONZHVUJMTOGPUOZYXICQFHUOBA
             KLTEVZGEOBFHANVDUCUMBGBAVVPEUFMHOVFNTCNFMHOSAGSPRLNGMDU
             LXCVZBLOXKWKNVDFUCXDHKIODNYQHPAUWOPR
plaintext:   HEHADBEENVERYSUCCESSFULINTRACKINGANDBREAKINGUPCERTAINNE
             FARIOUSSECRETSOCIETIESAMONGSTTHENATIVESTHENHETOOKHISLON
             GLEAVEANDGOTMARRIEDRATHER___________
```

2. Break this ciphertext. The plaintext begins with "... a certain document came into being. It was the draft of a secret agreement ..."

```
VHGATVLWROWXFPNFHKNEOROLBQLWXIFSKKTSYIAAXWCWALYLAEFGPAEQDGBTFVGU
TUHKELOWBZAFFCFLOPGSTIGARRVSWFPCYKRYXBSKIZRKCUCYHTTSYEUROBMKRQTS
PSQROUTOZGKADCIAAIRMDIDGFPROHKUTFKKCTODGUNQXOAABHCFNFVJSTIGARILT
KCCSYICOQRPEADCHTALTGVIVQFPSLYWXECEGEEWWGRFNVOZAFTMAPNEGUYNXWXFQ
QVLWWGVEETCKBVQVASTIGAZOYYICHVWOZEIKCEEVINUICKKRBQYWMEYWALSQHSPT
FKKCNNWQSNZSCFLTKKGEBYKMETBMCTSGOMIDCCWHNTGSSGEMULBFISSVTXLIDWGR
SNDGQOTFKEWVFACABPA
```

Unit 75
Attacking quagmire 3 with cribs: group theory

The overall strategy is the same: use the crib and ciphertext to partially fill a key table, then expand our knowledge of that key table until we can find the cipher's key; usually this means we will (or nearly will) construct its entire tableau. The details are different, however. We will use what we know about the Q3 and its group properties to do that expansion.

Before we begin, we need the ability to compose and invert permutations that are only partially known. Our key table has holes in it, and we need to modify the techniques we learned in Units 15 and 18 to handle holes in the permutations. Consider this partially known permutation; we will find its (partially known) inverse.

$$k = \texttt{Q.E....PS...N.....BX.....}$$

Remember that we think of the permutation as taking the alphabet and replacing letters as in a monoalphabetic substitution. So

```
plaintext letters:     ABCDEFGHIJKLMNOPQRSTUVWXYZ
                       ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
ciphertext letters:    Q.E....PS...N.....BX.....
```

Now we reverse the substitution:

```
plaintext letters:     ABCDEFGHIJKLMNOPQRSTUVWXYZ
                       ↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑
ciphertext letters:    Q.E....PS...N.....BX.....
```

Reorder the columns so that the lower row has the known letters in their alphabetical positions:

```
plaintext letters:     BTDECFGJKLNOPMQHARISVWXUYZ
                       ↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑
ciphertext letters:    .B..E........N.PQ.S....X..
```

The partially known inverse is obtained by taking the top row and replacing any letter that is above an unknown letter with a dot. So:

$$k^{-1} = \texttt{.T..C........M.HA.I....U..}$$

When we find the composition of two partially known permutations, typically some letters are lost and more holes appear. Consider these two:

k_1 = Q.E....PS...N.....BX.....
k_2 = G...L.WX.....BPA.....

The action of k_1 is this:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
Q.E....PS...N.....BX.....
```

and of k_2 is this:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
G...L.WX.....BPA.....
```

If k_1 acts first, then their composition is

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
Q.E....PS...N.....BX.....
↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
..L.....P.................
```

It doesn’t leave a lot. But as we fill holes in the tableau, compositions will give more lucrative results.

We also need to know how to merge rows of a table. Suppose we want to merge

a = ABC.....

and

b = ...XYZ

To merge them, we fill holes in one of them with letters in the same position of the other. Their merger (and we are borrowing a symbol for it) is

$a \boxplus b$ = ABC.....XYZ

When we can and cannot merge rows of the table will be explained shortly.

Now let us take inventory of what we know about the Q3 that can help us to go from a partially filled key table to a complete or nearly complete tableau.

1. The Q3 is a quagmire cipher. This alone has some important consequences regarding the merging of rows.

a. Each row of the key table or tableau is a permutation of the alphabet. For us, this means that each letter of the alphabet can occur exactly once in each row. If two rows have the same letter but in different positions, they cannot be merged. For example,

```
ABCQ......................
```

and

```
.......................QXYZ
```

cannot be merged.

b. Quagmire ciphers have the "column property" that says each column in its tableau also contains each letter exactly once. So if two rows have even only one letter in common, and that letter is in the same position of each, then those rows can and should be merged. For example,

```
ABC.............Q.........
```

and

```
................Q......XYZ
```

can and should be merged into

```
ABC.............Q......XYZ
```

2. Q3 ciphers are quagmire 3 ciphers, and not quagmire 1 or quagmire 2 ciphers. This means that we can find the key from one of the rows of the tableau. To understand how, take a look at the top of the tableau in Table 72.1:

key letter	**plaintext alphabet** QUAGMIREBCDFHJKLNOPSTVWXYZ
Q	QUAGMIREBCDFHJKLNOPSTVWXYZ
U	UAGMIREBCDFHJKLNOPSTVWXYZQ

If we encipher Q with row U we get U. If we encipher U we get A, etc. This forms a cycle:

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

This may seem obvious and maybe even silly, but consider this portion of the tableau in Table 72.2, where the plaintext has been put in order:

key letter	**plaintext alphabet** ABCDEFGHIJKLMNOPQRSTUVWXYZ

Q	ABCDEFGHIJKLMNOPQRSTUVWXYZ
U	GCDFBHMJRKLNIOPSUETVAWXYZQ

Again, if we use row U and form a cycle, we get the same one, and that cycle is

(GMIREBCDFHJKLNOPSTVWXYZQUA)

The rotation is irrelevant, since cycles can be written to start at any point in it. And the key for the Q3 is immune to rotations. Suppose we have a Q3 with key k, and that k' is a version of it that has been rotated left by n steps. Then (check it if you don't believe it)

$$k' = k \circ R_n$$

But the rows of the tableau are exactly the same:

$$\begin{aligned} q_m' &= k' \circ R_m \circ k'^{-1} \\ &= (k \circ R_n) \circ R_m \circ (k \circ R_n)^{-1} \\ &= k \circ R_n \circ R_m \circ R_{-n} \circ k^{-1} \\ &= k \circ R_{m+n-n} \circ k^{-1} \\ &= k \circ R_m \circ k^{-1} = q_m \end{aligned}$$

Once we have constructed the tableau, we try each row that is a 26-cycle until we find one that gives us the cipher's key.

3. The set of Q3 permutations, which we write as $Q_3[k]$, is a group. The binary operation in Q_3 is the composition of permutations, $\circ$. Therefore:

   a. The identity permutation is in every Q3 tableau, and it has order 1. In every isomorphism it is mapped to 0, the identity element of $\mathbb{Z}_{26}$.

   b. The composition of any two permutations in the tableau is another member of the tableau.

   c. The inverse of any permutation in the tableau is another member of the tableau.

4. $(Q_3[k], \circ)$ is isomorphic to $(\mathbb{Z}_{26}, +_{26})$. This tells us some things:

   a. In every isomorphism the identity permutation is mapped to 0, the identity element of $\mathbb{Z}_{26}$.

   b. There is exactly one permutation in the tableau that is its own inverse. In every isomorphism it is mapped to 13 in $\mathbb{Z}_{26}$. If we find a permutation that behaves as its own inverse, then we know that it is this one.

   c. There are twelve permutations that have order 13, and they are mapped to the non-zero even numbers in $\mathbb{Z}_{26}$. We have to be careful in deciding to which number each is mapped, because it may not be clear how to choose. But because there are twelve isomorphisms from $Q_3[k]$ to $\mathbb{Z}_{26}$, we can always choose to map one of them to 2. How the others are mapped will require consistency with this decision.

d. There are twelve permutations with order 26, and they are mapped to the odd numbers, except 13. How we decide this mapping will depend on consistency with our choice(s) in regard to the rule in 4c above, and with the rule in 4e below.

e. Whenever we compose two rows of the tableau, the result corresponds to a number in $\mathbb{Z}_{26}$ that is the sum (modulo 26) of the numbers corresponding to the two rows that are composed. So if q_m is mapped to 5 and q_n is mapped to 12, then $q_m \circ q_n$ is mapped to 17.

f. $(\mathbb{Z}_{26}, +_{26})$ is a commutative group, and therefore so is $(Q_3[k], \circ)$. For us this means that whenever we take the composition of two permutations, we can take the factors in either order, and merge the results. This helps us to fill in holes in the result, since some letters will survive the composition of two partially filled permutations in one way of choosing which is first, and other letters will survive in the other way. In a sense, we can redefine composition to be a symmetrized version:

$$a \circ b \;=\; (a \circ b) \boxplus (b \circ a)$$

(Remember that in general, this does not work outside the context of Q3s and Vigenères.) *For the remainder of this unit, all compositions will be symmetrized.*

g. Any order-26 element can be used to generate the set of numbers in $\mathbb{Z}_{26}$. So any permutation with order 26 can be used to generate the entire tableau by taking compositions with itself.

h. Since 13 + an even number is odd, if we have the order-2 permutation and one order-13 permutation, we can construct an order-26 permutation by composing them.

i. The order of a permutation is the same as the order of its inverse, since $\mathbb{Z}_{26}$ has an automorphism in which $n \to -n$ modulo 26.

5. $(Q_3[k], \circ)$ is isomorphic to the Vigenère group $(\mathcal{V}, \circ)$. In exercises 4 and 5 of Unit 20 you looked at the permutations in $\mathcal{V}$ and tabulated their orders and decomposed them into cycles. From them we know:

a. The permutation that has order 2 is factored into 13 2-cycles. So if we ever find a 2-cycle in a permutation, then we know that it has order 2 and only has 2-cycles.

b. The permutations that have order 13 are factored into two 13-cycles.

c. The permutations that have order 26 are 26-cycles.

It is time to consider an example and work it through to the end. Doing so by hand is a long and tiring process, but we can automate it (or most of it, depending on whether we can translate all of the rules above into program code). Here is the example from the previous unit:

```
1234 1234123 412 341234123 4123 41234123 41234
THIS MESSAGE WAS ENCRYPTED WITH QUAGMIRE THREE
BXFK NLWKQWY OQP YYEBAVBLH OSAK JXGJGSBY MPBYW
```

$q_0 = e$	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
q_1	Q.E....PS...N.....BX.....
q_2	G...L.WX.....BPA.....
q_3	...HY.JKF.....W.....A.
q_4	....W.....GY.VJ.KM..O...

Let's start a new table, and add some columns to help us keep track of the relationships between permutations in the Q3 to elements of $\mathbb{Z}_{26}$. Row numbers will be arbitrary and only for convenience, and we will drop some when their rows are merged with others, and add some when new permutations are found. The first one that we can add is the identity permutation, since we know that it is always present in a Q3 tableau. A temporary label, q_n, is numbered arbitrarily and does not refer to the permutation constructed from R_n; again, this notation is for convenience. This, then, is our starting point:

row #	**temp. label**	**permutation**	**notes**	**order**	**element of $\mathbb{Z}_{26}$**
0	$q_0 = e$	ABCDEFGHIJKLMNOPQRSTUVWXYZ	always present	1	0
1	q_1	Q.E....PS...N.....BX.....			
2	q_2	G...L.WX.....BPA.....			
3	q_3	...HY.JKF.....W.....A.			
4	q_4	....W.....GY.VJ.KM..O...			

The next we might add are the inverses of the rows already present.

5		.T..C.....M.HA.I....U..	q_1^{-1}		
6		TR....A....E...S.....GH..	q_2^{-1}		
7		Y....I.D.GH.....S.E.	q_3^{-1}		
8		.....M..QS.T.W.....PE.N.	q_4^{-1}		

More that we can add are compositions of the original four. Notice that the first one has no surviving letters when the composition was taken. We include it because we will find letters for it later. Note that compositions are symmetrized in this unit of the book. Whenever we compose a permutation with itself, we know that the result is mapped under the isomorphism to an even number in $\mathbb{Z}_{26}$, so we know that its order is 13. We can arbitrarily assign one of them to be mapped to 2, so long as we do not make

any more assignments unless they are derived from this choice. It may seem dumb to assign 2 to the one with no letters known, but that's what we did. If $q_1 \circ q_1$ is mapped to 2, then q_1 is mapped either to 1 or to 14, since doubling either of these gives 2 modulo 26.

1	q_1	`Q.E....PS...N......BX.....`			1 or 14
9		`.....`	$q_1 \circ q_1$	13	2
10		`..L.....P.....Q.....`	$q_1 \circ q_2$		
11		`..YP....W.....Q.`	$q_1 \circ q_3$		
12		`J.W....VK...Y......N.....`	$q_1 \circ q_4$		
13		`W.....G.....`	$q_2 \circ q_2$	13	
14		`J..X.....G.`	$q_2 \circ q_3$		
15		`.....O.....W.....V.....`	$q_2 \circ q_4$		
16		`...KA.....`	$q_3 \circ q_3$	13	
17		`.....JA....O.....`	$q_3 \circ q_4$		
18		`....O.....G.....`	$q_4 \circ q_4$	13	

Because they share a `G` in the same position, rows 13 and 18 can be merged. Row 18 will be dropped. Likewise, since rows 12 and 14 share a `J` in the first position, they can be merged. Row 14 will be dropped.

12		`J.WX...VK...Y......N....G.`	$q_1 \circ q_4 = q_2 \circ q_3$		
13		`W...O.....G.....`	$q_2 \circ q_2 = q_4 \circ q_4$	13	

Now we might try adding compositions of inverses.

19		`.A.....U.....XGTH.....`	$q_1^{-1} \circ q_2$		
20		`.....K..F.....`	$q_1^{-1} \circ q_3$		
21		`.M.....G.....`	$q_1^{-1} \circ q_4$		
22		`B.....QS.....RH..P..`	$q_2^{-1} \circ q_1$		
23		`.....Y...W..G...JKT.`	$q_2^{-1} \circ q_3$		
24		`M...G.....WA..K.....`	$q_2^{-1} \circ q_4$		
25		`.....S....P.....`	$q_3^{-1} \circ q_1$		
26		`.....S..WX.....Y..P.L.`	$q_3^{-1} \circ q_2$		
27		`....S.....E..G.H...K.W.`	$q_3^{-1} \circ q_4$		
28		`.....N.....B.....`	$q_4^{-1} \circ q_1$		
29		`M.....E...P.A.....L...`	$q_4^{-1} \circ q_2$		
30		`....N.QS..W.....E...Y...`	$q_4^{-1} \circ q_3$		

Notice immediately that rows 20, 24, 25, and 29 can be merged.

20		`M...GSE...PWA..K..F.....`	$q_2^{-1}q_4 = q_3^{-1}q_1 = q_4^{-1}q_2$		

Row 24 has a 2-cycle: A→M→A, so this is the self-reciprocal element. It has order 2 and only has 2-cycles. It is mapped to 13 in $\mathbb{Z}_{26}$. By completing the other 2-cycles that are partially present, we can add one letter (L):

20		M...GSE...PWA..K..F...L...	$2^{-1}4$, $3^{-1}1$, $4^{-1}2$, $1^{-1}3$	2	13

Rows 19 and 27 can be merged, as can rows 22 and 30. They are inverses of each other.

19		.A..S..U.....E.XGTH...K.W.	$q_1^{-1} \circ q_2 = q_3^{-1} \circ q_4$		
22		B...N.QS..W.....ERH.YP..	$q_2^{-1} \circ q_1 = q_4^{-1} \circ q_3$		

Let's pause for a moment and see where we are:

row #	temp. label	permutation	notes	order	element of $\mathbb{Z}_{26}$
0	$q_0 = e$	ABCDEFGHIJKLMNOPQRSTUVWXYZ	always present	1	0
1	q_1	Q.E....PS...N.....BX.....			1 or 14
2	q_2	G...L.WX.....BPA.....			
3	q_3	...HY.JKF.....W.....A.			
4	q_4	....W.....GY.VJ.KM..O...			
5	q_5	.T..C.....M.HA.I....U..	q_1^{-1}		
6	q_6	TR....A....E...S.....GH..	q_2^{-1}		
7	q_7	Y....I.D.GH.....S.E.	q_3^{-1}		
8	q_8	.....M..QS.T.W.....PE.N.	q_4^{-1}		
9		.....	$q_1 \circ q_1$	13	2
10		..L.....P.....Q.....	$q_1 \circ q_2$		
11		..YP....W.....Q.	$q_1 \circ q_3$		
12		J.WX...VK...Y.....N....G.	$q_1 \circ q_4 = q_2 \circ q_3$		
13		W...O.....G.....	$q_2 \circ q_2 = q_4 \circ q_4$	13	
15		.....O.....W.....V.....	$q_2 \circ q_4$		
16		...KA.....	$q_3 \circ q_3$	13	
17		.....JA....O.....	$q_3 \circ q_4$		
19		.A..S..U.....E.XGTH...K.W.	$1^{-1}2$, $3^{-1}4$, 22^{-1}		
20		M...GSE...PWA..K..F...L...	$1^{-1}3$, $2^{-1}4$, $3^{-1}1$, $4^{-1}2$	2	13
21		.M.....G.....	$q_1^{-1} \circ q_4$		
22		B...N.QS..W.....ERH.YP..	$2^{-1}1$, $4^{-1}3$, 19^{-1}		
23		.....Y...W..G...JKT.	$q_2^{-1} \circ q_3$		
26		.....S..WX.....Y..P.L.	$q_3^{-1} \circ q_2$		
28		.....N.....B.....	$q_4^{-1} \circ q_1$		

We cannot merge row 5 with row 1, even though they do not have any letters in conflict. The reason is that if we were to merge them, then they would become an order-2 permutation. But row 20 is already an order-2 permutation, and we can only have one (rows 5 and 20 are incompatible and cannot be

merged either). So we have given row 5 a label. Rows 6, 7, and 8 are incompatible with any rows above them, due to same letters in different positions or different letters in same positions; they get labels, too.

Now we are going to try to fill in some holes in the permutations that we already have. For that purpose, row 20 is a lucky find. It has a relationship with each of rows 1, 2, 3, and 4. By composing row 20 with row 1, we get

`N.G.YLJKF...Q.....W.....A.`

This is merged into row 3, giving us four new letters. By inverting the new row 3 we add four new letters to row 7 as well. By composing row 20 with row 2, we get

`E...WPL.....G.....KM.....`

This gets merged into row 4 to add three more letters; the inverse adds three more to row 8. Composing row 20 with the new row 3 adds five letters to rows 1 and 5 similarly, and composing rows 20 and 4 adds four letters to rows 2 and 6.

1	q_1	`Q.E.JWYPS...N.....LBX...M.`			1 or 14
2	q_2	`G...LKWX..VOE....BPA.....`			
3	q_3	`N.GHYLJKF...Q.....W.....A.`			
4	q_4	`E...WPL.....GY.VJ.KM..O...`			
5	q_5	`.T..C....E.SYM.HA.I...FUG.`	q_1^{-1}		
6	q_6	`TR..M.A...FE..LS.....KGH..`	q_2^{-1}		
7	q_7	`Y....ICD.GHF.A..M.....S.E.`	q_3^{-1}		
8	q_8	`....A.M..QSGT.WF.....PE.N.`	q_4^{-1}		

Row 20 cannot be merged with any of rows 0 through 8, so we give it a label.

20	q_9	`M...GSE...PWA..K..F...L...`	$1^{-1}3$, $2^{-1}4$, $3^{-1}1$, $4^{-1}2$	2	13

If we square the new row 1 to finally add some letters to row 9, we get

`..J...M.L...............N.`

This shares an `M` with row 8, so we merge them and row 8 becomes the one that maps to 2 in $\mathbb{Z}_{26}$. Also, row 16 shares an `A` with row 8, so we merge it in also. Inverting the new row 8 lets us add some letters to row 4. Knowing that row 4 is the inverse of the one that maps to 2 in $\mathbb{Z}_{26}$ means that it must map to 24. Since q_3 squared maps to 2, q_3 must map to 1 or 14.

3	q_3	`N.GHYLJKF...Q.....W.....A.`			1 or 14
4	q_4	`E...WPL..CDIGY.VJ.KM..O...`		13	24
8	q_8	`..JKA.M.LQSGT.WF.....PE.N.`	$q_1 \circ q_1$, $q_3 \circ q_3$, q_4^{-1}	13	2

Rows 7 and 17 share A in the same position, so we can merge them. It is the inverse of row 3, so it must map to 25 or 12 in $\mathbb{Z}_{26}$. Inverting the new row 7 adds two letters to row 3.

3	q_3	N.GHYLJKFM..Q.S...W.....A.			1 or 14
7	q_7	Y....ICD.GHFJA..M.O...S.E.	$q_3^{-1} = q_3 \circ q_4$		25 or 12

Merge row 12 into row 5 to add six letters to row 5. Use the newer rows 1 through 4 to add more letters to row 5. Invert it to add letters to row 1.

1	q_1	Q.E.JWYPSAI.NTFO..LBXHCDM.			1 or 14
5	q_5	JTWXCO.VKE.SYMPHA.IN..FUG.	q_1^{-1}, $q_1 \circ q_4$, $q_2 \circ q_3$		25 or 12

Recalculate $q_8 = q_1 \circ q_1 = q_3 \circ q_3$ and merge in some more letters. Invert it to add letters to row 4.

4	q_4	ENFUWPL..CDIGYHVJ.KM..O...		13	24
8	q_8	..JKACMOLQSGTBWF....DPE.N.	$q_1 \circ q_1$, $q_3 \circ q_3$, q_4^{-1}	13	2

Recalculate $q_2 \circ q_2$ and $q_4 \circ q_4$ to add letters to row 13:

13		WYP.OVI..FU.L...C.DG..H...	$q_2 \circ q_2 = q_4 \circ q_4$	13	

Time for another pause to take stock of our progress.

row #	**temp. label**	**permutation**	**notes**	**order**	**element of $\mathbb{Z}_{26}$**
0	$q_0 = e$	ABCDEFGHIJKLMNOPQRSTUVWXYZ	always present	1	0
1	q_1	Q.E.JWYPSAI.NTFO..LBXHCDM.			1 or 14
2	q_2	G...LKWX..VOE....BPA.....			
3	q_3	N.GHYLJKFM..Q.S...W.....A.			1 or 14
4	q_4	ENFUWPL..CDIGYHVJ.KM..O...		13	24
5	q_5	JTWXCO.VKE.SYMPHA.IN..FUG.	q_1^{-1}, $q_1 \circ q_4$, $q_2 \circ q_3$		25 or 12
6	q_6	TR..M.A...FE..LS.....KGH..	q_2^{-1}		
7	q_7	Y....ICD.GHFJA..M.O...S.E.	$q_3^{-1} = q_3 \circ q_4$		25 or 12
8	q_8	..JKACMOLQSGTBWF....DPE.N.	$q_1 \circ q_1$, $q_3 \circ q_3$, q_4^{-1}	13	2
10		..L.....P.....Q.....	$q_1 \circ q_2$		
11		..YP....W.....Q.	$q_1 \circ q_3$		
13		WYP.OVI..FU.L...C.DG..H...	$q_2 \circ q_2 = q_4 \circ q_4$	13	
15		.....O.....W.....V.....	$q_2 \circ q_4$		
19		.A..S..U.....E.XGTH...K.W.	$1^{-1}2$, $3^{-1}4$, 22^{-1}		
20	q_9	M...GSE...PWA..K..F...L...	$1^{-1}3$, $2^{-1}4$, $3^{-1}1$, $4^{-1}2$	2	13
21		.M.....G.....	$q_1^{-1} \circ q_4$		

22		B...N.QS..W.....ERH.YP..	$2^{-1}1, 4^{-1}3, 19^{-1}$		
23		.....Y...W..G...JKT.	$q_2^{-1} \circ q_3$		
26		.....S..WX.....Y..P.L.	$q_3^{-1} \circ q_2$		
28		.....N.....B.....	$q_4^{-1} \circ q_1$		

There are still a lot of holes in rows 10, 11, 15, 19, 21, 22, 23, 26, and 28, but they are all related to rows 1 through 8, which have become more populated. By now you know how it's done, so I can gloss over the details.

10		Y.L..ICDPGHFJAK...OQ.X..E.	$q_1 \circ q_2$		
11		T.YPM.AIWNF...LS..C..KGHQ.	$q_1 \circ q_3$		
15		L.K.IDO....HW.X..NVE.....	$q_2 \circ q_4$		
19		.AI.S.FUVL.PCEDXGTHJ..K.W.	$1^{-1}2, 3^{-1}4, 22^{-1}$		
21		CMO.FHS.DWXK.GV.E..Y..P.L.	$q_1^{-1} \circ q_4$		
22		B.MONGQSCTWJ...L..ERHIYP..	$2^{-1}1, 4^{-1}3, 19^{-1}$		
23		..A.QENF..LY...W..G...JKT.	$q_2^{-1} \circ q_3$		
26		C...FHS..WXK.G..E..Y..P.L.	$q_3^{-1} \circ q_2$		
28		..AIQENF..LYB.CW..G..OJKT.	$q_4^{-1} \circ q_1$		

Right away we can see that row 10 should merge into row 7, 11 into 6, 26 into 21, and 28into 23.

row #	temp. label	permutation	notes	order	element of $\mathbb{Z}_{26}$
0	$q_0 = e$	ABCDEFGHIJKLMNOPQRSTUVWXYZ	always present	1	0
1	q_1	Q.E.JWYPSAI.NTFO..LBXHCDM.			1 or 14
2	q_2	G...LKWX..VOE....BPA.....			
3	q_3	N.GHYLJKFM..Q.S...W.....A.			1 or 14
4	q_4	ENFUWPL..CDIGYHVJ.KM..O...		13	24
5	q_5	JTWXCO.VKE.SYMPHA.IN..FUG.	$q_1^{-1}, q_1 \circ q_4, q_2 \circ q_3$		25 or 12
6	q_6	TRYPM.AIWNFE..LS..C..KGHQ.	$q_2^{-1}, q_1 \circ q_3$		
7	q_7	Y.L..ICDPGHFJAK.M.OQ.XS.E.	$q_3^{-1}, q_3 \circ q_4, q_1 \circ q_2$		25 or 12
8	q_8	..JKACMOLQSGTBWF....DPE.N.	$q_1 \circ q_1, q_3 \circ q_3, q_4^{-1}$	13	2
13		WYP.OVI..FU.L...C.DG..H...	$q_2 \circ q_2 = q_4 \circ q_4$	13	
15		L.K.IDO....HW.X..NVE.....	$q_2 \circ q_4$		
19		.AI.S.FUVL.PCEDXGTHJ..K.W.	$1^{-1}2, 3^{-1}4, 22^{-1}$		
20	q_9	M...GSE...PWA..K..F...L...	$1^{-1}3, 2^{-1}4, 3^{-1}1, 4^{-1}2$	2	13
21		CMO.FHS.DWXK.GV.E..Y..P.L.	$q_1^{-1} \circ q_4, q_3^{-1} \circ q_2$		
22		B.MONGQSCTWJ...L..ERHIYP..	$2^{-1}1, 4^{-1}3, 19^{-1}$		
23		..AIQENF..LYB.CW..G..OJKT.	$q_2^{-1} \circ q_3, q_4^{-1} \circ q_1$		

We have a lot of letters, but only sixteen rows. And we would like to determine which permutation is mapped to 1 in $\mathbb{Z}_{26}$. We do have the permutations that map to 2 and 13, so we can use them to construct other permutations whose mappings we know. Start with 13 + 2 = 15 (rows 20 and 8). Their composition is

`T..PM.A.W.FE..LS..C..KG...`

But this is most of row 6, so what we learn is that row 6 maps to 15:

6	q_6	`TRYPM.AIWNFE..LS..C..KGHQ.`	q_2^{-1}, $q_1 \circ q_3$	26	15

Compose again with row 8, which is our 2 in $\mathbb{Z}_{26}$, to get 17:

31		`..NFTY.LEBCA.RG...J.PSMO..`		26	17

Continue as far as we can up the ladder of odd numbers:

32		`..BC.N.GA.J...MY..Q.F.TWR.`		26	19
33		`...J.B.M..Q...TN....CY.E..`		26	21
34		`...Q...T.......B....JN.A..`		26	23
35		`.....QB....`		26	25

Row 25 conflicts with row 7 because they have `Q` in different positions. So row 5 must be the one that maps to 25. Merge row 35 into it, then invert it to add letters to row 1. Row 1 must be the one mapped to 1, in order to be the inverse of 25 mod 26.

1	q_1	`QVE.JWYPSAI.NTFOU.LBXHCDM.`	q_1	26	1
3	q_3	`N.GHYLJKFM..Q.S...W.....A.`	q_3	13	14
5	q_5	`JTWXCO.VKE.SYMPHA.INQBFUG.`	q_1^{-1}, $q_1 \circ q_4$, $q_2 \circ q_3$	26	25
7	q_7	`Y.L..ICDPGHFJAK.M.OQ.XS.E.`	q_3^{-1}, $q_3 \circ q_4$, $q_1 \circ q_2$	13	12

Having the permutation that corresponds to 1 in $\mathbb{Z}_{26}$ we can try to generate the tableau. Start by making a new table:

row #	**permutation**	**element of $\mathbb{Z}_{26}$**
0	`ABCDEFGHIJKLMNOPQRSTUVWXYZ`	0
1	`QVE.JWYPSAI.NTFOU.LBXHCDM.`	1
8	`..JKACMOLQSGTBWF....DPE.N.`	2
7	`Y.L..ICDPGHFJAK.M.OQ.XS.E.`	12
20	`M...GSE...PWA..K..F...L...`	13
3	`N.GHYLJKFM..Q.S...W.....A.`	14
6	`TRYPM.AIWNFE..LS..C..KGHQ.`	15
31	`..NFTY.LEBCA.RG...J.PSMO..`	17
32	`..BC.N.GA.J...MY..Q.F.TWR.`	19

33	...J.B.M..Q...TN....CY.E..	21
34	...Q...T.......B....JN.A..	23
5	JTWXCO.VKE.SYMPHA.INQBFUG.	25
2	G...LKWX..VOE....BPA.....	
13	WYP.OVI..FU.L...C.DG..H...	
15	L.K.IDO....HW.X..NVE.....	
19	.AI.S.FUVL.PCEDXGTHJ..K.W.	
21	CMO.FHS.DWXK.GV.E..Y..P.L.	
22	B.MONGQSCTWJ...L..ERHIYP..	
23	..AIQENF..LYB.CW..G..OJKT.	

Start with 1 + 1 = 2. Merge the result into row 8.

8	UHJKACMOLQSGTBWFX..VDPE.N.	2

Next, 2 + 1 = 3. It matches row 23.

23	XPAIQENF.ULYBVCWD.GH.OJKT.	3

3 + 1 = 4 is a new row.

36	DOQSUJTWGX.MVHEC..YPKFAIB.	4

Let's just compute the entire trajectory of row 1 and merge when necessary. Row 34 and 21 will merge as well.

row #	**permutation**	**element of $\mathbb{Z}_{26}$**
0	ABCDEFGHIJKLMNOPQRSTUVWXYZ	0
1	QVE.JWYPSAI.NTFOU.LBXHCDM.	1
8	UHJKACMOLQSGTBWFX..VDPE.N.	2
23	XPAIQENF.ULYBVCWD.GH.OJKT.	3
36	DOQSUJTWGX.MVHEC..YPKFAIB.	4
37	.FULXABCYDGNHPJEK.MOIWQSV.	5
38	KWX.DQVEM.YTPOAJI.NFSCULH.	6
39	ICD..UHJNKMBOFQAS.TWLEX.P.	7
40	SE..KXPATINVFWUQL.BC.JD.O.	8
15	LJK.IDOQBSTHWCXU.NVE.A..F.	9
19	.AI.S.FUVLBPCEDXGTHJ.QK.W.	10
2	GQS.LKWXH.VOEJ.DYBPA.UI.C.	11
7	YUL..ICDPGHFJAK.MVOQ.XS.E.	12
20	MX..GSE.OYPWAQIKNHFU.DL.J.	13

3	NDGHYLJKFMOCQUSITPWX....A.	14
6	TRYPM.AIWNFEUXLSBOCD.KGHQ.	15
22	BKMONGQSCTWJXD.LVFERHIYPU.	16
31	VINFTYULEBCADRG.HWJKPSMOX.	17
41	HSTWBMX.JVEQRKYGPCAIOLNFD.	18
32	PLBCVNDGAHJUKIMYOEQSF.TWR.	19
42	O.VEHTRYQPAXISNMFJULWGBCK.	20
33	FGHJPBKMUOQDSLTNWAX.CYVEI.	21
13	WYPAOVINXFU.L.BTCQDGEMHJS.	22
21	CMOQFHSTDWXK.GVBEU.YJNPAL.	23
43	ENFUWPLB.CDIGYHVJXKMATOQ..	24
5	JTWXCO.VKE.SYMPHADINQBFUG.	25

Look at row 31. The permutation there maps to 17 and has order 26. It only has two holes in it. If we try to write down its 26-cycle, we have this segment:

...QHLAVSJBIETKCNRWMDFYXOGUP...

The only missing letter is Z, so the cycle is

(AVSJBIETKCNRWMDFYXOGUPZQHL)

Then the permutation is

VINFTYULEBCADRGZHWJKPSMOXQ

With a complete permutation of order 26, we can generate the entire tableau without any remaining holes, by taking powers of the permutation. We do just that, and match them to our table, and fill in all the gaps:

permutation	**element of $\mathbb{Z}_{26}$**
ABCDEFGHIJKLMNOPQRSTUVWXYZ	0
QVERJWYPSAIZNTFOUKLBXHCDMG	1
UHJKACMOLQSGTBWFXIZVDPERNY	2
XPAIQENFZULYBVCWDSGHROJKTM	3
DOQSUJTWGXZMVHECRLYPKFAIBN	4
RFULXABCYDGNHPJEKZMOIWQSVT	5
KWXZDQVEMRYTPOAJIGNFSCULHB	6
ICDGRUHJNKMBOFQASYTWLEXZPV	7
SERYKXPATINVFWUQLMBCZJDGOH	8
LJKMIDOQBSTHWCXUZNVEGARYFP	9
ZAINSRFUVLBPCEDXGTHJYQKMWO	10
GQSTLKWXHZVOEJRDYBPAMUINCF	11
YULBZICDPGHFJAKRMVOQNXSTEW	12
MXZVGSEROYPWAQIKNHFUTDLBJC	13

NDGHYLJKFMOCQUSITPWXBRZVAE	14
TRYPMZAIWNFEUXLSBOCDVKGHQJ	15
BKMONGQSCTWJXDZLVFERHIYPUA	16
VINFTYULEBCADRGZHWJKPSMOXQ	17
HSTWBMXZJVEQRKYGPCAIOLNFDU	18
PLBCVNDGAHJUKIMYOEQSFZTWRX	19
OZVEHTRYQPAXISNMFJULWGBCKD	20
FGHJPBKMUOQDSLTNWAXZCYVEIR	21
WYPAOVINXFURLZBTCQDGEMHJSK	22
CMOQFHSTDWXKZGVBEURYJNPALI	23
ENFUWPLBRCDIGYHVJXKMATOQZS	24
JTWXCOZVKERSYMPHADINQBFUGL	25

Now we take all the permutations with order 26 and write them as cycles to reveal the keyword.

(AQUXDRKISLZGYMNTBVHPOFWCEJ)	1
(AXKLYTHFEQDIZMBPWJURSGNVOC)	3
(ARZTOJDLNPEXSMHCUIYVWQKGBF)	5
(AINFULBCDGHJKMOQSTWXZVERYP)	7
(ALHQZPUGOXYFDMWRNCKTEIBJSV)	9
(AGWIHXNJZFKVUMELORBQYCSPDT)	11
(ATDPSCYQBROLEMUVKFZJNXHIWG)	15
(AVSJBIETKCNRWMDFYXOGUPZQHL)	17
(APYREVZXWTSQOMKJHGDCBLUFNI)	19
(AFBGKQWVYIUCHMSXEPNLDJOTZR)	21
(ACOVNGSRUJWPBMZIDQEFHTYLKX)	23
(AJECWFOPHVBTNMYGZLSIKRDXUQ)	25

And we have found it. The key for the cipher is

VERYPAINFULBCDGHJKMOQSTWXZ

Reading and references

Thomas Kaeding, “Classical substitution ciphers and group theory,” Cryptology ePrint Archive, report 2023/669.

Programming tasks

1. Extend your routines for alphabetic permutations to handle the possibility that there are missing letters. Also symmetrize the composition, but remember only to use it when working with quagmire 3 ciphers.

2. Automate the process of building the complete tableau from partial keys. It's not as hard as it seems from doing the example in this unit.

Exercises

1. Find both keywords:

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
1	S...Y.HW...U.....JZK.N...
2	..G.D...M...Q.S....F...A..
3	I..S...M..YJ..L..H.....DF.
4	..OQM...U..GX.Z..L.N.....
5	....I...Q..C.V.D.YZJA.....

2. Find both keywords. The period of the cipher is five.

```
plaintext:    SOMERELEVANTWORDSTOQUAGMIREARESWAMPMIASMABOG
ciphertext:   MYQEORSVVCWDFOOHBLOFPJUMADHXRXMEXMZAZXSWBNJG
```

3. Find the secret word. The period of the cipher is four.

```
plaintext:   YOUCANNOTGUESSTHEWORDBECAUSEITISUNUSUALTHESECRETWORDIS__________
ciphertext:  SMFKUQZLOHFWKVGPMXAXEDHKUTOWDEVDCQFDCBYVBROWWAHVNMICDVDYWITCMAB
```

Unit 76
Quagmire 3: recovering keywords

Did we mention that the set of Q3 permutations is isomorphic to ($\mathbb{Z}_{26}$, +) and that the automorphism of Q3 are the permutations in Table 29.1 for the multiplication cipher, which correspond to elements of $\mathbb{Z}_{26}^*$? What this means is that we can find the keywords for a quagmire 3 from its key table by manipulating the permutations in it. First of all, the shift key appears in the column under the first letter of the alphabet key, unless we choose some other indexing letter. How we find the alphabet key will depend on the order of the permutation we use. The easiest way is to use one of order 26. Consider this example:

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ	order
0	OWQXTRKZHFLBYJINGVMSPAUCDE	26
1	ZNLGYOWSTAUPCVERBIXDFHJKQM	13
2	JQMTOUDVRWXCZBFLYPHIKNGSEA	13
3	RKDMHNCOAPQGTUVWXJEZBFLYSI	13
4	QZVFWDOLKYIHNMGSAXPUTCERJB	2

Row 0 has order 26. We begin by expressing it as a cycle. Since A→O and O→I, etc, we have

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

So that permutation can be written as this cycle:

$$k' = (\text{AOIHZETSMYDXCQGKLBWUPNJFRV})$$

Now we try taking every 1st letter, every 3rd letter, every 5th, 7th, 9th, 11th, 15th, 17th, 19th, 21st, 23rd, and 25th (these are the numbers in $\mathbb{Z}_{26}^*$):

1	AOIHZETSMYDXCQGKLBWUPNJFRV
3	AHTYCKWNROZSDQLUJVIEMXGBPF
5	AEDKPVZYGURHMQWFISCBJOTXLN
7	ASGNIYLFZXWVTQPOMKJHDBRECU
9	AYWODUIXPHCNZQJEGFTKRSLVMB
11	AXJSWHGVDNTBIQRYPELOCFMUZK
15	AKZUMFCOLEPYRQIBTNDVGHWSJX
17	ABMVLSRKTFGEJQZNCHPXIUDOWY
19	AUCERBDHJKMOPQTVWXZFLYINGS

21	ANLXTOJBCSIFWQMHRUGYZVPKDE
23	AFPBGXMEIVJULQDSZORNWKCYTH
25	AVRFJNPUWBLKGQCXDYMSTEZHIO

These can also be generated by taking k' and composing it with the permutations of the multiplication cipher in Table 29.1. We pick out the one that looks most like a keyword followed by letters in alphabetical order:

AUCERBDHJKMOPQTVWXZFLYINGS

A rotation gives us the original key used by the cipher:

FLYINGSAUCERBDHJKMOPQTVWXZ

Suppose that we do not have a permutation with order 26. If we have orders 13 and 2, then we can construct one with order 26 by taking the composition of them. To understand why this works, remember that Q3 is isomorphic to $(\mathbb{Z}_{26}, +)$. The elements of $\mathbb{Z}_{26}$ with order 13 are the even numbers (except 0), and the element with order 2 is 13. Well, 13 plus any non-zero even number is an odd number, and those (except 13) have order 26 in $\mathbb{Z}_{26}$. In our example, take rows 3 and 4 and find their composition:

XIFNLMVGQSAOUTCERYWBZDHJPK

This has the correct order, so we can just repeat the above process to find the keyword.

Next, suppose we have only permutations with order 13. Take row 1 from our example and call it k':

k' = ZNLGYOWSTAUPCVERBIXDFHJKQM

If we try to write it as a cycle, we find that we come back to A on the thirteenth step:

A→Z→M→C→L→P→R→I→T→D→G→W→J→A

So we have a 13-cycle. If we start with any letter that is not in this cycle, we will get another 13-cycle. So we can only rewrite this permutation as the product of two cycles:

k' = (AZMCLPRITDGWJ) (BNVHSXKUFOEYQ)

The traditional way to recover the keyword is to interleave these two cycles, then take every n^{th} letter as we did for the order-26 permutation. There are thirteen ways to interleave them, and twelve ways to take each nth letter, for a total of 156 possibilities to try. To make it clear how the interleaving is done, here are the thirteen for our example (we only label the first one, for use later):

$$k'' = \begin{array}{l} A_B Z_N M_V C_H L_S P_X R_K I_U T_F D_O G_E W_Y J_Q \\ A_N Z_V M_H C_S L_X P_K R_U I_F T_O D_E G_Y W_Q J_B \\ A_V Z_H M_S C_X L_K P_U R_F I_O T_E D_Y G_Q W_B J_N \end{array}$$

$$A_H Z_S M_X C_K L_U P_F R_O I_E T_Y D_Q G_B W_N J_V$$
$$A_S Z_X M_K C_U L_F P_O R_E I_Y T_Q D_B G_N W_V J_H$$
$$A_X Z_K M_U C_F L_O P_E R_Y I_Q T_B D_N G_V W_H J_S$$
$$A_K Z_U M_F C_O L_E P_Y R_Q I_B T_N D_V G_H W_S J_X$$
$$A_U Z_F M_O C_E L_Y P_Q R_B I_N T_V D_H G_S W_X J_K$$
$$A_F Z_O M_E C_Y L_Q P_B R_N I_V T_H D_S G_X W_K J_U$$
$$A_O Z_E M_Y C_Q L_B P_N R_V I_H T_S D_X G_K W_U J_F$$
$$A_E Z_Y M_Q C_B L_N P_V R_H I_S T_X D_K G_U W_F J_O$$
$$A_Y Z_Q M_B C_N L_V P_H R_S I_X T_K D_U G_F W_O J_E$$
$$A_Q Z_B M_N C_V L_H P_S R_X I_K T_U D_F G_O W_E J_Y$$

If we take the seventh of these and take every n^{th} letter, for n = 1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25, we have the following twelve. The keyword is evident when n = 3.

1	AKZUMFCOLEPYRQIBTNDVGHWSJX
3	AUCERBDHJKMOPQTVWXZFLYINGS
5	AFPBGXMEIVJULQDSZORNWKCYTH
7	AOIHZETSMYDXCQGKLBWUPNJFRV
9	AEDKPVZYGURHMQWFISCBJOTXLN
11	AYWODUIXPHCNZQJEGFTKRSLVMB
15	ABMVLSRKTFGEJQZNCHPXIUDOWY
17	ANLXTOJBCSIFWQMHRUGYZVPKDE
19	AVRFJNPUWBLKGQCXDYMSTEZHIO
21	AHTYCKWNROZSDQLUJVIEMXGBPF
23	ASGNIYLFZXWVTQPOMKJHDBRECU
25	AXJSWHGVDNTBIQRYPELOCFMUZK

It is possible to automate this process as presented above, but let's see if we can't find a way to list all of the 156 possibilities by using alphabetic permutations. Let's start by considering this permutation:

$$D_1 = \text{ADCFEHGJILKNMPORQTSVUXWZYB}$$

It has every other letter in its usual position, but the other thirteen letters have been shifted to the left by two steps, so that this permutation is a prototype for the shifting we did in listing the thirteen interleaved permutations earlier. If we look at the trajectory of D_1 we see all of the thirteen ways of interleaving the two slices of the alphabet:

$$\begin{aligned}
D_1 & = \text{ADCFEHGJILKNMPORQTSVUXWZYB}\\
D_2 &= D_1^2 = \text{AFCHEJGLINKPMROTQVSXUZWBYD}\\
D_3 &= D_1^3 = \text{AHCJELGNIPKRMTOVQXSZUBWDYF}\\
D_4 &= D_1^4 = \text{AJCLENGPIRKTMVOXQZSBUDWFYH}\\
D_5 &= D_1^5 = \text{ALCNEPGRITKVMXOZQBSDUFWHYJ}\\
D_6 &= D_1^6 = \text{ANCPERGTIVKXMZOBQDSFUHWJYL}
\end{aligned}$$

$$
\begin{aligned}
D_7 &= D_1^{7} = \texttt{APCRETGVIXKZMBODQFSHUJWLYN} \\
D_8 &= D_1^{8} = \texttt{ARCTEVGXIZKBMDOFQHSJULWNYP} \\
D_9 &= D_1^{9} = \texttt{ATCVEXGZIBKDMFOHQJSLUNWPYR} \\
D_{10} &= D_1^{10} = \texttt{AVCXEZGBIDKFMHOJQLSNUPWRYT} \\
D_{11} &= D_1^{11} = \texttt{AXCZEBGDIFKHMJOLQNSPURWTYV} \\
D_{12} &= D_1^{12} = \texttt{AZCBEDGFIHKJMLONQPSRUTWVYX} \\
D_0 &= D_1^{13} = \texttt{ABCDEFGHIJKLMNOPQRSTUVWXYZ} = e
\end{aligned}
$$

(These form a cyclic group isomorphic to $\mathbb{Z}_{13}$, since $D_m \circ D_n = D_{m+n}$.) The strategy is to find one interleaving of the two cycles that make up k' and generate the 156 possibilities by shifting the interleaving with a D_m and then taking every n^{th} letter with an M_n (from Table 29.1):

$$k''' = k'' \circ D_{\mathrm{m}} \circ M_n$$

One of the k''' will be a rotated version of k_1, the key used in the quagmire 3. We can recognize it if we look for a permutation with a large segment that is in alphabetical order (even if some letters are missing from that segment). For our example, that one is

$$k''' = k'' \circ D_6 \circ M_3 = \texttt{AUCER}\underline{\texttt{BDHJKMOPQTVWXZ}}\texttt{FLYINGS}$$

The ordered segment is underlined.

Reading and references

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; chapter XVIII.

Thomas Kaeding, “Classical substitution ciphers and group theory,” Cryptology ePrint Archive, report 2023/669.

Programming tasks

1. Create a function that rewrites an alphabetic permutation as a product of disjoint (not overlapping) cycles. Add it as a method to your class of alphabetic permutations.

2. Automate the process of recovering the keyword for a quagmire 3 key table. The function you wrote in Task 1 of Unit 66 may be useful.

Exercises

1. These are exercises from page 183 of Gaines's book.

   a. `UVDWSXKYHZCFRJQLINGPTOMEAB`

   b. `HJGKFPEQORSTDMBUVWXAYZCLIN`

   c. `GXYZMHAFTRLKEVQUOJWIPNSBCD`
      `E.GJIK.LB..UTCVW.QDXS.....`

Unit 77
Quagmire 4 cipher

The generic quagmire cipher that we saw in Unit 60 is what we call a *quagmire 4 cipher* (also called *polyalphabetic type 4*). It uses mixed alphabets for both the plaintext and the ciphertext, and the mixing is generated from two keywords. The ciphertext alphabets are shifted versions of the mixed alphabet, and the shifts form another keyword, as they do in the Vigenère cipher. We won't work out an example here; look back at the example in Unit 60.

Remember that the permutations of a Q4 are formed by sandwiching a rotation from the Vigenère set between two (different) monoalphabetic substitutions:

$$Q_4[k_1, k_3] = \{k_3 \circ R_n \circ k_1^{-1}\} \tag{77.1}$$

where the k_1 and k_3 are seen in Equation 60.1. If we rewrite a generic element of Q_4 by inserting an identity permutation in this way:

$$\begin{aligned} q_n &= k_3 \circ R_n \circ k_1^{-1} \\ &= k_3 \circ R_n \circ e \circ k_1^{-1} \\ &= k_3 \circ R_n \circ (k_3^{-1} \circ k_3) \circ k_1^{-1} \\ &= (k_3 \circ R_n \circ k_3^{-1}) \circ (k_3 \circ k_1^{-1}) \end{aligned}$$

we see that it is an element of a Q3 composed with some permutation on the right. That permutation, $k_3 \circ k_1^{-1}$, is the same for all members of $Q_4[k_1, k_3]$, and so the set is a right coset of the Vigenère. But if we rewrite q_n in this way:

$$\begin{aligned} q_n &= k_3 \circ R_n \circ k_1^{-1} \\ &= k_3 \circ e \circ R_n \circ k_1^{-1} \\ &= k_3 \circ (k_1^{-1} \circ k_1) \circ R_n \circ k_1^{-1} \\ &= (k_3 \circ k_1^{-1}) \circ (k_1 \circ R_n \circ k_1^{-1}) \end{aligned}$$

we see that $Q_4[k_1, k_3]$ is also a left coset of $\mathcal{V}$, and that the constant permutation composed on the left is the same as before.

Suppose now that we have a key table that we may have found by performing a crib-based attack, but we do not know the identity of the cipher used. In earlier units, we have seen how to check for Q1, Q2, and Q3 ciphers. To check for a Q4, build two new key tables. In one, put all possible compositions of inverses of rows of the original table with (not inverted) rows. In the other, reverse the order of composition. If the two new tables pass the Q3 tests, then the original table was from a Q4. Here is an example:

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	AKLMJNBOF.STDUVW.GXYRZC.P.
1	QXYZWCSI..HETRAB.VDFOGJ.L.
2	PDFGBJHKR.MNEOQS.ATUIVW...
3	BLMNKODQG.TUFVWX.JYZACI.H.
4	CABDRFIGH.KLPMNO..QSZTU.W.
5	EGJKFLRM..OQASTU.DVWHXY.C.
6	VIPHCEWRY.BDXFGJ.ZKLUMN.Q.

The first table we construct is by taking the inverses of each row and composing with non-inverted rows, i.e., $q_m^{-1} \circ q_n$. Here we have sorted them and merged any duplicates.

plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHI.KLMNOP.RSTUVW.Y.
BKLN.OCPF.TVDW.Y.H..EUA.M.
CLNOKPDSH.VWF.Y....UBAG.I.
E.KLHNBOD.STCVW..FY.R.U.G.
FOPSNTHVK..Y...U.LAGDMI.E.
GCDFBHM.R.LNIOPS.ETVAW....
HPSTOV.W..Y.K.UA.NGMFIR.B.
.FH.DKRLB.OPESTV.CW.MY..U.
.STVPWK.N...LUAG.OMIHRE.C.
KTVWS.LYO..UNAGM.PIR.EB.D.
LVW.TYN...UAOGM..SREKBC.F.
MDFHC.IKE.NORPST.BVWG.Y...
NW.YV.O.S.AGPMIR.TEBLCD.H.
O.Y.W.PUT.GMSIRE.VBCNDF...
PY...USAV.M.TREB.WCDOFH.K.
.REBICUDG.H.AKLN.MOP.ST.W.
RH.KFLENC.PSBTVW.D.YI...A.
S..UYATGW.IRVEBC..DFPH..L.
UEBCRDAFM..KGLNO.IPS.TV...
VUAG.MWIY.EB.CDF..H.TKL.O.
WAGMUI.R..BCYDFH...KVLN.P.
.GMIARYE..CD.FH..UKLWNO.S.
YMIRGE.BU.DF.H.K.ALN.OP.T.
.IREMB.C..FHU.KL.GNOYPS.V.

We see immediately that the identity permutation is on the top row. If we compose the second and third rows, we get

```
LVW.TYN.P.UAO.M....EKBC.F.
```

which matches a row near the middle, etc. All signs point to this new table being a Q3. We leave the construction of the table for $q_m \circ q_n^{-1}$ and checking its properties as an exercise for the reader.

The last thing to mention in this unit is that there is a special class of Q4 ciphers that are self-reciprocal (involutory). We construct them by taking k_1 and k_3 to be reversals of each other, i.e.,

$$\begin{aligned} k_3 &= k_1 \circ z \\ k_1 &= k_3 \circ z \end{aligned}$$

where z is the reversed alphabet (see Equation 16.1). Because k_1 and k_3 are related, the cipher depends only on one of them, and we can write

$$Q_4^{(l)}[k_1] = \{k_1 \circ z \circ R_n \circ k_1^{-1}\} \tag{77.2}$$

We will see this again in the next unit.

Reading and references

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; chapter XVIII.

American Cryptogram Association, www.cryptogram.org/downloads/aca.info/ciphers/QuagmireIV.pdf

Thomas Kaeding, “Classical substitution ciphers and group theory,” Cryptology ePrint Archive, report 2023/669.

Programming tasks

1. You already have an implementation of the quagmire 4 cipher from Unit 60, but it was called simply “`quagmire`.” Make a link to it so that you also have a definition for `quagmire4`.

2. Write a function or script to perform a dictionary attack on the quagmire 4 cipher.

3. Write a function that takes the three quagmire 4 keywords and outputs the keys of the substitution and Vigenère ciphers into which the quagmire can be factored.

Exercises

1. Finish the example concerning the identification of a Q4 key table.

2. Encipher this text with keywords `FOUR` (plaintext alphabet), `LETTER` (ciphertext alphabet), and `WORD` (shifts).

   Seems like only yesterday
   I left my mind behind
   Down in the Gypsy Café
   With a friend of a friend of mine
   She sat with a baby heavy on her knee
   Yet spoke of life most free from slavery
   With eyes that showed no trace of misery
   A phrase in connection first with she I heard
   That love is just a four-letter word

   (from “Love Is Just a Four-Letter Word” by Bob Dylan)

3. Decipher this text with keywords `FOUR` (plaintext alphabet), `PIGMENT` (ciphertext alphabet), and `COLOR` (shifts).

```
MYCSWLPRQAJWJIFMPXASGCXCFUQNQRCQQSMZXXAPZXXTHLQMQVHYVYCMQRC
MZDZYPAYPTHQDRUYSSONWMIVQVRUNIFYCCPPLQMQVGASWPUCLQUHXPNPZCP
QBJEPTGFNSTFMPXZPZYLQVMPXSPOAPZAQTCIFDTCPPTWZSSYQBQAQCONPNQ
EYBSUDYHNQWIRCYRYBMFXSFUQNCSCDZUHRYNYWZXCPPHYLQVQPSFPCQQSVJ
DZYVMPSTHSSXYHAAXCSCWOESHSXWPPDOSFDTXCULPFQAJTVQAPESIBQXCPW
MCZYYDRMXBQFXSRUTWWTDDZYFMPSDAJDVP
```

4. Break this ciphertext with a dictionary attack. All keywords are common five-letter English words.

```
XZPRVLJINCQXKYWVKXJJQXPRVCARJTCQSJDQCRECDGGIXTKKYWDUWJXBKTZ
CAKJAHRKMZFNMNZLBLRXVAIRZGQXGIXTKPTFNCQJUCIPCDQCIZSLJINCQXS
JDQCRECDGRHKWMPAANXKBTCQHRQLUMHCAOVCWXZOAANIHJVOSNWCVTPLJNC
QJTQYIOHBMVWHRIIPMNAGLGOUXLGJUXJJDUKWJNQIPVAOIOJQYIAANJYHKF
XKAVCQOJGNAKWJNXKACGUPFCLCQLKQOMJDNAKACCZRHCALRHKXZOCCLTKMD
NQSZFVAKECDG
```

5. Which quagmire cipher was used to create each of these key tables?

   a.

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	ASBCEFHIJ.MNPQRT.WXYZDO.G.
1	DOUGLASBC.FHIJK..PQRTVW.Y.
2	ASBCEFHIJ.MNPQRT.WXYZDO.G.
3	MNPQRTVWX.ZDOUGL.SBCEFH.J.
4	SBCEFHIJK.NPQRTV.XYZDOU.L.

b.

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	Y.BEWRATU.NSCILD.XGZ.JK.P.
1	N.GHTJKMR.QESFD..OWCX.Z.I.
2	JFQSG.VWD.YCZPM..HBKERA.O.
3	CTHJOKMPA.SRUGFV.NXDYZ..B.
4	NAGHTJKMR.QESFDU.OWCX.Z.I.
5	J.QSGUVWD.YCZPML.HBKERA.O.
6	ELONICDFZ.HYJTAK.BPRQSU.W.

c.

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	DIJKLMNGO.QRSTUV.EXFHYZ.B.
1	E.KLMNOHP.RSTUVW.FYGI.A.C.
2	NSTUVWXQY.ABCDEF.OHPRIJ.L.
3	TYZABCDWE.GHIJKL.UNVXOP.R.

d.

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	P.HUJVWZT..KERF..YOMN.A.C.
1	XA.BS..FH.KLMOE..UNYV.Z.I.
2	C.YFNHJKM.PXQUL..ZBATIG...
3	Y.GXEA.CD..RJKIM.PLSQUV.Z.
4	F.XJAKMOP.UBVWNZ.IDCGER.L.
5	X.RBS.DFH.KLMOE..UNYV...I.

e.

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	C...INZGW.BDEXF..UKLMO..V.
1	M..PQS.TJ..WXKZ..FYING..H.
2	J.ALMO.PE..TUFVW.BZC..I.D.
3	B.REFHAJN.LMOGP..YTUV...I.
4	Z.SRYI.NV.ABDWEF.TJKL.O.U.

Unit 78
Better dictionary attack on quagmire 4

This attack is almost the same at that in Unit 73 on the Q3. The difference is that Q4 uses two different substitution keys, so the last step in the attack is to use the hill-climbing attack on the substitution from Unit 36.

Programming tasks

1. Implement the attack. Use your Q1-ness function from Unit 73, Task 1.

Exercises

1. Break this ciphertext:

```
KTTLZGQWMHZQNJXEDIXEUDAYAVBBISQKKNRHPIXOHHBNEKQOTILBIGFWKOL
EMKBVWVUQRGKTULSODSGWHUIWXEBBXMKPEYUINKGHZQOOAZTTLZSCCJHLTB
QHLPPGEOQMJEMUHJESCKWLEBBISQJOWPGGBGEUBDJOKPORVCQGVPTBWJIVB
BGEUBDJLDDOOLXNMYNWWKIBKOQCSLBOYEUDVRVCZKGISTBRVWGCJNEBEGPN
BEYUFDOOWTQUIZWNVJZSCITNKKSGZSDHJCWOBEGKITGSQFBVSSCBGWTPBGA
VBPIWTVONWGUERWMUOJZLDOOUXPIPHLTMEUHUAYNMUEYVSJOCHKPIGEHUPW
HWYAJVSDOOILKKQIVGSNHVTBRVSKBNELQGLLSJPXEUKPCOUQUGSKHKGFWPC
JNEBEGPHPITIVQUTHAXKMHZQNJIODCJWTXTLEOBBIHAQIXOSCKWIGZFJEXJ
MTWTBBNFXFALVATTJGWNBGPHPSBFKFTTAVKPCVAJANWFBSJXXKBGHEZIXEH
FPXZZJPCZXTAJVSJPXNHKBTZWUSWHGTBZXVQVLSHKCJQWFARWGTTRZKUAHH
LTBQHHHSZPVJAYGZQOTLKQUTVWUSMASJARZGQAGFWHIXIEUALOWDSBHUDVJ
ZWBORVHVCJKXOQVHXUALOWJSZWLUIMSWXTJVXFARWGKUXZLXIPEEZAYPSUI
XEWPFJMWPFMASXIGFEJAGSWPKJMHHEYVLFIYALUUZHVWITPHPAHPLOKNFXP
IFHARUIXHUKPAEDEGPHPQIHLBOGZZQOOAZTIXPSUAHPVGSGICBAHISQGTHZ
QOOVXTUIHHKOJOXTIKHZQOTLKQUTVWUSLZSCITEUJCOPLNKLZSVOOHVUAYW
AJSYNSBONWGHUTHAXIGFSCKPPVUALVAZKGPLJOIHXNIGBSCKNWFRMJEWBXZ
GWUSRWGQGGFWOPIIEJFJIEDOOWTTTLEVBEYVZJMRISJPXWTPBGAVBBXMLRI
IPSQGBFKFTBHXKKRVLBBILUJWRSEGKKWSBBPHORBQHLTPRELBXZSXPBGPHP
```

Unit 79
Attacking quagmire 4 with cribs: symmetry

The approach is similar to that in Unit 74, but now the plaintext alphabet is mixed in a different way from the rows in the key table. Take this example:

```
12341234123412341234123412341234123412341234123412341234123412341234123412341234123
THISMESSAGEWASENCRYPTEDWITHQUAGMIREFOURTHECRIBNEEDSTOBESOMEWHATLONGERTHANBEFORE
HPZYLXLYJTIIJFIVSWSXHXEIMGARGSXFMWIOCQCZVXHJMYGKOCLZCYIYCUIIVSMUCAXKNGABZYIOCWI
```

Below is what the crib tells us about the key table. We have used “0” for the plaintext row and numbered the ciphertext rows from 1, for convenience, but unlike the Q3 case, we cannot treat row 0 on equal footing with the others.

0	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
1	J.S.O..VM...LZC..N.HG.....
2	SY.CX.TP....UA...WFGQ.....
3	..HEI.XAZ....G...CLM....S.
4	B...KO.....UFV.XRJYZ..I...

We can only use rows 1, 2, 3, and 4 to find relationships between distances. What we have are:

source rows	relations
1, 2	JS = OX = VP = LU = ZA = NW = HG = GQ JO = SX JV = SP JL = SU JZ = SA JN = SW JH = SG JG = SQ OV = XP OL = XU OZ = XA ON = XW OH = XG OG = XQ VL = PU

	VZ = PA VN = PW VH = PG VG = PQ LZ = UA LN = UW LH = UG LG = UQ ZN = AW ZH = AG ZG = AQ NH = WG NG = WQ
1, 3	SH = OI = VA = MZ = ZG = NC = HM SO = HI SV = HA SM = HZ SZ = HG SN = HC OV = IA OM = IZ OZ = IG ON = IC OH = IM VM = AZ VZ = AG VN = AC VH = AM MN = ZC MH = ZM ZN = GC ZH = GM NH = CM
1, 4	JB = OK = LF = ZV = NJ = HZ JO = BK JL = BF JZ = BV JN = BJ JH = BZ OL = KF OZ = KV ON = KJ OH = KZ LZ = FV

	LN = FJ LH = FZ ZN = VJ ZH = VZ NH = JZ
2, 3	CE = XI = TX = PA = AG = WC = FL = GM CX = EI CT = EX CP = EA CA = EG CF = EL CG = EM XP = IA XA = IG XW = IC XF = IL XG = IM TP = XA TA = XG TW = XC TF = XL TG = XM PW = AC PF = AL PG = AM AW = GC AF = GL WF = CL WG = CM FG = LM
2, 4	SB = XK = UF = AV = WJ = FY = GZ SX = BK SU = BF SA = BV SW = BJ SF = BY SG = BZ XU = KF XA = KV XW = KJ XF = KY XG = KZ UA = FV UW = FJ

	UG = FZ AW = VJ AF = VY AG = VZ WF = JY WG = JZ FG = YZ
3, 4	IK = GV = CJ = LY = MZ IG = KV IC = KJ IL = KY IM = KZ GC = VJ GL = VY GM = VZ CL = JY CM = JZ LM = YZ

Nicely, we have MZ in the vertical relationships between rows 1 and 3 and between 3 and 4, so we can combine them into

SH = OI = VA = MZ = ZG = NC = HM = IK = GV = CJ = LY

If we provisionally assign this distance a value of one, we have these key fragments:

SHMZGVA OIK NCJ LY

From rows 1 and 2, JH = SG, so we can combine two of the fragments.

NCJ..SHMZGVA OIK LY

From rows 2 and 4, SG = BZ, so we can add B.

NCJ.BSHMZGVA OIK LY

From rows 1 and 4, JB = LF, so we can add F.

NCJ.BSHMZGVA OIK LYF

From rows 2 and 4, SB = XK = UF = WJ, so we can add W, X, and U.

NCJWBSHMZGVA OIKX LYFU

From rows 1 and 2, JS = VP = GQ, so we can add Q and P.

```
NCJWBSHMZGVAQP OIKX LYFU
```

From rows 2 and 3, CE = XI = TX, we can place E and T.

```
ENCJWBSHMZGVAQP OIKX.T LYFU
```

The missing letters are D and R. One of them must go between X and T. The other will be placed between fragments, and those fragments have two orderings. The twelve possibilities are

```
ENCJWBSHMZGVAQPROIKXDTLYFU
ENCJWBSHMZGVAQPOIKXDTRLYFU
ENCJWBSHMZGVAQPOIKXDTLYFUR
ENCJWBSHMZGVAQPDOIKXRTLYFU
ENCJWBSHMZGVAQPOIKXRTDLYFU
ENCJWBSHMZGVAQPOIKXRTLYFUD
ENCJWBSHMZGVAQPRLYFUOIKXDT
ENCJWBSHMZGVAQPLYFUROIKXDT
ENCJWBSHMZGVAQPLYFUOIKXDTR
ENCJWBSHMZGVAQPDLYFUOIKXRT
ENCJWBSHMZGVAQPLYFUDOIKXRT
ENCJWBSHMZGVAQPLYFUOIKXRTD
```

Since J and K are infrequent letters, they are not likely to be in the keyword (although we can't know for certain just yet). Similarly for X, Y, and Z. In only two of these twelve possibilities do we have JK = XY = YZ, and there it has the value 19.

```
ENCJWBSHMZGVAQPLYFUROIKXDT
ENCJWBSHMZGVAQPLYFUDOIKXRT
```

If we take every nineteenth letter (wrapping around) for these two, we have

```
ERABDFGJKLMNOQSTUVWXYZCIPH
EDABRFGJKLMNOQSTUVWXYZCIPH
```

The underlined portion is in alphabetical order in the first of these two, but R is out of place in the second. So it appears that we have found one of our keys:

```
CIPHERABDFGJKLMNOQSTUVWXYZ
```

To find the other key, we can put this one into a key table, and line up the columns as in the original. For example, in the original we had NWCJ under plaintext R.

	plaintext alphabet
0	.....R.....
1	CIPHERABDFGJKLMNOQSTUVWXYZ
2	BDFGJKLMNOQSTUVWXYZCIPHERA
3	JKLMNOQSTUVWXYZCIPHERABDFG

4	WXYZCIPHERABDFGJKLMNOQSTUV

If we fill in the letters that we can by comparing to the original key table, we get most of the key in row 0:

	plaintext alphabet
0	OPST.W.Y.QUAGMIREBCDFH..LN
1	CIPHERABDFGJKLMNOQSTUVWXYZ
2	BDFGJKLMNOQSTUVWXYZCIPHERA
3	JKLMNOQSTUVWXYZCIPHERABDFG
4	WXYZCIPHERABDFGJKLMNOQSTUV

Since I is already used, between H and L likely goes JK. And T.W.Y. looks to be TVWXYZ, since U has also been used elsewhere. We have this:

	plaintext alphabet
0	OPSTVWXYZQUAGMIREBCDFHJKLN
1	CIPHERABDFGJKLMNOQSTUVWXYZ
2	BDFGJKLMNOQSTUVWXYZCIPHERA
3	JKLMNOQSTUVWXYZCIPHERABDFG
4	WXYZCIPHERABDFGJKLMNOQSTUV

We might want to rearrange them and straighten out the plaintext alphabet, to make decrypting more of the text (before and after the crib) easier.

	plaintext alphabet
0	ABCDEFGHIJKLMNOPQRSTUVWXYZ
1	JQSTOUKVMWXYLZCIFNPHGERABD
2	SYZCXITPVHERUABDOWFGQJKLMN
3	WPHEIRXAZBDFYGJKUCLMVNOQST
4	BLMNKODQGSTUFVWXRJYZACIPHE

Reading and references

William F. Friedman, The Principles of Indirect Symmetry of Position in Secondary Alphabets and Their Application in the Solution of Polyalphabetic Substitution Ciphers, Washington D.C.: U.S. Government Printing Office, 1935, www.nsa.gov/Portals/70/documents/news-features/declassified-documents/friedman-documents/publications/FOLDER_226/41760669079979.pdf

Programming tasks

1. Borrow your code for Q3 with some alteration to implement (at least most of) the procedure for Q4.

Exercises

1. Break this ciphertext. The plaintext begins with “Three times a day for many months I had witnessed this performance, but custom had not reconciled ...”

```
CLYAECKFALXGMLJUEFIRJPHUVPOCSFZXCXRQOVAICLCETQEADSEDGFEYVKFWNUHT
MXGGDVRMODRVKZAIEZFQRZUHQVUQKSEVAGXSXETNSUPSIHCASIHLLMKGQBHTEEAY
ANRECXFYSXIVCLXENOLKIRZPTFBSOOQESBXEYQSZAISKBSVTZQQVPKGTEXUKSECL
HYKPUKSFCKWIITDOGEZUWACUVYIKQUHVSUUXEVXJMQRXQSIKXKGQMXGYAKLOKASQ
GMZBGUWIVLOWDWTGSAALWXBQJOHYADRHUVPMVYGMMOXGDUKSERMQILCLMXNSUWAC
UAZUBSBWIAXQKINRAATHVAFVFSKHUYPKOSQXGXSNEUWAAXOKTFSXCXMGLHTBSMQD
WTGOISQUHXFWMMUHCLCUKXRIBBXBWQROUHIALFXBVJLCEKRMMXTUXXBLPKVTFNUX
BAJPMURQEMUICLXAZAMYQESBXHMLBWQMXGWIIUIWQLEDGLEIUYIBRGCUFRZJYFTK
KQENDZCQXGXTEZKANNZMGXFSGEICWXMBILQOBBNOCUKPKF
```

Unit 80
Attacking quagmire 4 with cribs: group theory

Recall from Equation 77.1 that the permutations in a quagmire 4 tableau have this form:

$$q_n = k_3 \circ R_n \circ k_1^{-1}$$

Suppose we take this combination:

$$\begin{aligned} q_m^{-1} \circ q_n &= (k_3 \circ R_m \circ k_1^{-1})^{-1} \circ (k_3 \circ R_n \circ k_1^{-1}) \\ &= k_1 \circ R_{-m} \circ k_3^{-1} \circ k_3 \circ R_n \circ k_1^{-1} \\ &= k_1 \circ R_{n-m} \circ k_1^{-1} = q_{n-m}^{(3a)} \end{aligned} \tag{80.1}$$

The result is a quagmire 3 permutation whose key is the Q4's key on the plaintext side. On the other hand, suppose we take this combination:

$$\begin{aligned} q_m \circ q_n^{-1} &= (k_3 \circ R_m \circ k_1^{-1}) \circ (k_3 \circ R_n \circ k_1^{-1})^{-1} \\ &= k_3 \circ R_m \circ k_1^{-1} \circ k_1 \circ R_{-n} \circ k_3^{-1} \\ &= k_3 \circ R_{m-n} \circ k_3^{-1} = q_{m-n}^{(3b)} \end{aligned} \tag{80.2}$$

Now we have a Q3 permutation built from the key on the ciphertext side of the Q4. Our strategy to expand our knowledge of a Q4 key table is to take such combinations and work on the two Q3 tables with the technique of Unit 75. We can return to the Q4 by composing rows from the Q3 tables with rows from the Q4 table. If we have been working with the Q3 on the right (the plaintext side), we return by composing with a Q4 permutation on the left:

$$\begin{aligned} q_m^{(4)} \circ q_n^{(3a)} &= (k_3 \circ R_m \circ k_1^{-1}) \circ (k_1 \circ R_n \circ k_1^{-1}) \\ &= k_3 \circ R_m \circ k_1^{-1} \circ k_1 \circ R_n \circ k_1^{-1} \\ &= k_3 \circ R_{m+n} \circ k_1^{-1} \end{aligned} \tag{80.3}$$

Similarly, if we have been working on the Q3 on the left (the ciphertext side), we get back to the Q4 by composing on the right with a Q4 permutation:

$$\begin{aligned} q_n^{(3b)} \circ q_m^{(4)} &= (k_3 \circ R_n \circ k_3^{-1}) \circ (k_3 \circ R_m \circ k_1^{-1}) \\ &= k_3 \circ R_n \circ k_3^{-1} \circ k_{13} \circ R_m \circ k_1^{-1} \\ &= k_3 \circ R_{m+n} \circ k_1^{-1} \end{aligned} \tag{80.4}$$

Here is the example from the previous unit:

```
123412341234123412341234123412341234123412341234123412341234123412341234123412341234123
THISMESSAGEWASENCRYPTEDWITHQUAGMIREFOURTHECRIBNEEDSTOBESOMEWHATLONGERTHANBEFORE
HPZYLXLYJTIIJFIVSWSXHXEIMGARGSXFMWIOCQCZVXHJMYGKOCLZCYIYCUIIVSMUCAXKNGABZYIOCWI
```

0	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
1	J.S.O..VM...LZC..N.HG.....
2	SY.CX.TP....UA...WFGQ.....
3	..HEI.XAZ....G...CLM....S.
4	B...KO.....UFV.XRJYZ..I...

Take all possible combinations of the form $q_m^{-1} \circ q_n$ from these four. We can ignore those for which $m = n$, since those are the identity permutation which we assume is in all Q3 tableaux.

plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
C..O.....U.....
..T.....N....U...OMI....C.
.....E.....H...A.N.....
..A.....D.....T.....
.....EN.....T...D.....A.
.....MS..E..B.....
..Y.....T...SIR....CN.....
Y..RG.....H.....N.....
.....G...I..E...
R...F..N.....T.....
.S..P.....L.....M.....
....W.P.T.....

We use the technique of Unit 75 to expand our knowledge of this Q3 to the following:

plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
CLNO.P..H.....Y....U..G.I.
D.O...FT.....Y.....AC...R.
H..T.....U..N.....
I.H.....T..C.....U.
N..Y.....I..T.....H.
O.Y.W.PUT..MSIRE..BCN.F...
R...F.ENC..SBT.W.D.YI...A.
T.UA.G.....C..Y.H....N.

```
U..C.........N..I.......
YMIRGE..U....H...ALN..P.T.
.ST.PW..N...LUAG.OMIH.E.C.
..A...WIY..B.CDF....T...O.
..D.....R....O......A.....
..R....CA..........OY.....
...I.........H..U.......
...N............H.......
.......A.....R.....DO.....
.......OD...........R.....
.......R.....D............
.......YO....A.....R....D.
```

We can return to the Q4 by composing each row of the Q3 table with a row from the Q4 table on the left, as in Equation 80.3. When we do, and after we have merged any rows that can be merged, we find that we have added twenty-five letters to the four rows of the original key table. Unfortunately, if we take these results into the Q3 on the left and work on it, the result is that we cannot add any more letters to the Q4 table.

0	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
1	J.S.O.KVM..YLZCI.N.HG...B.
2	SYZCXITPV...UAB..WFGQ.K.M.
3	W.HEI.XAZ..FYGJK.CLMV.O.S.
4	BLMNKO.QG..UFVWXRJYZA.I.H.
	A..M.........V..G......P.
	C.B...IGH..L.MNO.E.SZ...W.
	E.J....MB....S.....WH...C.
	G.QS.T........Z..M.P....A.
	HFGJ.K..A....QST.BUVP.X.Z.
	MUVWTX..Q....PH..S.A....G.
	N.W...OZS....HE....BM...J.
	P..G.........Q..A........
	Q..Z.........A..V........
	V.PH.........G..Z......Q.
	Z.AB....P.....M..H.Q..T.V.
	..C....HW....B.....JS...N.
	..E....BC....J.....NW.....
	..N....SJ....W.....CB...E.
	...A............P........
	.......C.....E......N.....
	.......JE....N......C.....
	.......WN....C.....EJ.....
	..............P..Q........

Here is a second example. The period of the cipher is four.

```
THEQUICKBROWNFOXJUMPSOVERTHELAZYDOGTHEQUICKBROWNFOXJUMPSOVERTHELAZYDOG
HPIRGVHTQWJIZIJPWQYXPBNKNGAKYSTHTBXZVXUAMZDLNBOVUBQSGUKYCJIJHPIUJNSNCT
```

We have this information about the key table:

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	JQ.T.U.VMW.Y.ZC..NPHG.....
1	S.Z.XITPV...U.B..W.GQJ...N
2	..H.I.XA..D.Y.JKU....NOQST
3	.L.NK....STU.V.XRJYZA.IPH.

Take all possible combinations of the form $q_m^{-1} \circ q_n$ from these four. We can ignore those for which $m = n$, since those are the identity permutation which we assume is in all Q3 tableaux.

plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
..N...DSH...F....J.UBA...R
..T.........L.A.F....R.B.D
...R......DF.H...ALN...ST.
VU.G.M.I.R...C...ZH.T.....
....F.E.......V.M....Z.UAG
...Z.....AGM.I.E.V.C..FH..
OX.Z.Q.....M.....V.C......
Y...GEZ.....Q.......XO...V
...VP....YZQ...G.OM.H.E.C.
R..K.L.N...S.T...DXY......
J.T.PWKXN...L........R...D
..Y.W.PU....S.REL....D..JK

Apply the technique of Unit 75 to get

permutation	**element of $\mathbb{Z}_{26}$**
ABCDEFGHIJKLMNOPQRSTUVWXYZ	0
JSTVPWKXNYZQLUAGFOMIHREBCD	1
YMIRGEZBUCDFQHJKWALNXOPSTV	2
CLNOKPDSHTVWFXYZEJQUBAGMIR	3
TQUAZGVMXIREWBCDPYFHSJKLNO	4
IFHJDKRLBNOPESTVGCWXMYZQUA	5
NWXYVZOQSUAGPMIRKTEBLCDFHJ	6
UEBCRDAFMHJKGLNOZIPSQTVWXY	7

HPSTOVJWLXYZKQUADNGMFIREBC	8
XGMIARYEQBCDZFHJVUKLWNOPST	9
BKLNJOCPFSTVDWXYRHZQEUAGMI	10
SZQUYATGWMIRVEBCOXDFPHJKLN	11
MDFHCJIKELNORPSTABVWGXYZQU	12
LVWXTYNZPQUAOGMIJSREKBCDFH	13
QREBICUDGFHJAKLNYMOPZSTVWX	14
FOPSNTHVKWXYJZQUCLAGDMIREB	15
WAGMUIXRZEBCYDFHTQJKVLNOPS	16
EJKLHNBODPSTCVWXIFYZRQUAGM	17
PYZQXUSAVGMITREBNWCDOFHJKL	18
GCDFBHMJRKLNIOPSUETVAWXYZQ	19
KTVWSXLYOZQUNAGMHPIRJEBCDF	20
ZIREMBQCADFHUJKLXGNOYPSTVW	21
DNOPLSFTJVWXHYZQBKUACGMIRE	22
VUAGQMWIYREBXCDFSZHJTKLNOP	23
RHJKFLENCOPSBTVWMDXYIZQUAG	24
OXYZWQPUTAGMSIRELVBCNDFHJK	25

Writing the permutation corresponding to element 19 of $\mathbb{Z}_{26}$ as a cycle gives us the first keyword:

(AGMIREBCDFHJKLNOPSTVWXYZQU)

Return to the parital Q4 key table and this time take all possible combinations of the form $q_m \circ q_n^{-1}$ from the four rows. Again we can ignore those for which $m = n$.

plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
.C....H.U.....VG.J..MN...
V.....C.....
G.....N.Q.T....W.YZ..PH
..B...QG.S..VW.....IP....
P.....ZXB...J.....N...TU.
Q.....WX.....G
..J.....A....
.J.....N...T.A...XY..I.H
.....SO.I....Q.U.D...K..
.....AZ.....J.YL..N..S.UV
.....Z.....A.....JK..
...T....K.X...I.P.H.R.....

Apply the technique of Unit 75 to get

permutation	element of $\mathbb{Z}_{26}$
ABCDEFGHIJKLMNOPQRSTUVWXYZ	0
VWNX..ZSOCIPH.RQAUBD.GJKLM	1
GJ.K..MBRNOQSTUAV.WXYZCIPH	2
ZCTI..HWU.RABD.VGYJKLMNOQS	3
MNDO..SJ.TUVWXYGZLCIPH.RAB	4
H.XR..BCYD.GJKLZMPNOQSTUVW	5
STKU..WNLXYZCIPMHQ.RABD.GJ	6
BDI...J.PKLMNOQHSATUVWXYZC	7
WXOY..CTQIPH.RASBVD.GJKLMN	8
JKRL..NDAOQSTUVBWGXYZCIPH.	9
CIUP...XVRABD.GWJZKLMNOQST	10
NO.Q..TKGUVWXYZJCMIPH.RABD	11
.RYA..DIZ.GJKLMCNHOQSTUVWX	12
TULV..XOMYZCIPHN.SRABD.GJK	13
D.PG..KRHLMNOQS.TBUVWXYZCI	14
XYQZ..IUSPH.RABTDW.GJKLMNO	15
KLAM..O.BQSTUVWDXJYZCIPH.R	16
IPVH..RYWABD.GJXKCLMNOQSTU	17
OQGS..ULJVWXYZCKINPH.RABD.	18
RAZB...PCGJKLMNIO.QSTUVWXY	19
UVMW..YQNZCIPH.ORTABD.GJKL	20
.GHJ..LA.MNOQSTRUDVWXYZCIP	21
YZSC..PVTH.RABDU.XGJKLMNOQ	22
LMBN..QGDSTUVWX.YKZCIPH.RA	23
PHW...AZXBD.GJKYLIMNOQSTUV	24
QSJT..VMKWXYZCILPOH.RABD.G	25

These results are not complete, but we can try to fill in the gaps. Consider the order-26 permutation that we assigned to 1 in $\mathbb{Z}_{26}$:

VWNX..ZSOCIPH.RQAUBD.GJKLM

This permutation must be a single 26-cycle. If we try to write it as that cycle, we find these fragments:

.YLPQAVGZMHSBWJCN. .TDXKIORU.

There are two letters missing: E and F. There are two ways of completing the 26-cycle by gluing the fragments together with E and F at the junctions:

(YLPQAVGZMHSBWJCNFTDXKIORUE)
(YLPQAVGZMHSBWJCNETDXKIORUF)

If we take every n^{th} letter of the first one, where n is an element of $\mathbb{Z}_{26}^*$, we find a highly ordered segment when $n = 7$. Maybe you can guess what the keyword should be, but it does not appear exactly as we want.

1	YLPQAVGZMHSBWJCNFTDXKIORUE
3	YQGHWNDIULAZSJFXOEPVMBCTKR
5	YVSNKEAHCXUQMJDRPZWTOLGBFI
7	YZCIPHFRABDEGJKLMNOQSTUVWX
9	YHDLSXPBKQWIAJOVCRGNUZFEMT
11	YBOZDQCESIGTPJUHKVFLWRMXAN
15	YNAXMRWLFVKHUJPTGISECQDZOB
17	YTMEFZUNGRCVOJAIWQKBPXSLDH
19	YXWVUTSQONMLKJGEDBARFHPICZ
21	YIFBGLOTWZPRDJMQUXCHAEKNSV
23	YRKTCBMVPEOXFJSZALUIDNWHGQ
25	YEUROIKXDTFNCJWBSHMZGVAQPL

If we take every n^{th} letter of the second candidate 26-cycle, we again find a highly ordered segment when $n = 7$. The other segment correctly gives us the keyword: CIPHER.

1	YLPQAVGZMHSBWJCNETDXKIORUF
3	YQGHWNDIULAZSJEXOFPVMBCTKR
5	YVSNKFAHCXUQMJDRPZWTOLGBEI
7	YZCIPHERABDFGJKLMNOQSTUVWX
9	YHDLSXPBKQWIAJOVCRGNUZEFMT
11	YBOZDQCFSIGTPJUHKVELWRMXAN
15	YNAXMRWLEVKHUJPTGISFCQDZOB
17	YTMFEZUNGRCVOJAIWQKBPXSLDH
19	YXWVUTSQONMLKJGFDBAREHPICZ
21	YIEBGLOTWZPRDJMQUXCHAFKNSV
23	YRKTCBMVPFOXEJSZALUIDNWHGQ
25	YFUROIKXDTENCJWBSHMZGVAQPL

The completed permutation (assuming that CIPHER is the correct keyword) is

VWNXTYZSOCIPHERQAUBDFGJKLM

Its trajectory (all powers of it) gives us all of the permutations:

permutation	**element of $\mathbb{Z}_{26}$**
ABCDEFGHIJKLMNOPQRSTUVWXYZ	0
VWNXTYZSOCIPHERQAUBDFGJKLM	1
GJEKDLMBRNOQSTUAVFWXYZCIPH	2
ZCTIXPHWUERABDFVGYJKLMNOQS	3

MNDOKQSJFTUVWXYGZLCIPHERAB	4
HEXRIABCYDFGJKLZMPNOQSTUVW	5
STKUOVWNLXYZCIPMHQERABDFGJ	6
BDIFRGJEPKLMNOQHSATUVWXYZC	7
WXOYUZCTQIPHERASBVDFGJKLMN	8
JKRLFMNDAOQSTUVBWGXYZCIPHE	9
CIUPYHEXVRABDFGWJZKLMNOQST	10
NOFQLSTKGUVWXYZJCMIPHERABD	11
ERYAPBDIZFGJKLMCNHOQSTUVWX	12
TULVQWXOMYZCIPHNESRABDFGJK	13
DFPGAJKRHLMNOQSETBUVWXYZCI	14
XYQZVCIUSPHERABTDWFGJKLMNO	15
KLAMGNOFBQSTUVWDXJYZCIPHER	16
IPVHZERYWABDFGJXKCLMNOQSTU	17
OQGSMTULJVWXYZCKINPHERABDF	18
RAZBHDFPCGJKLMNIOEQSTUVWXY	19
UVMWSXYQNZCIPHEORTABDFGJKL	20
FGHJBKLAEMNOQSTRUDVWXYZCIP	21
YZSCWIPVTHERABDUFXGJKLMNOQ	22
LMBNJOQGDSTUVWXFYKZCIPHERA	23
PHWECRAZXBDFGJKYLIMNOQSTUV	24
QSJTNUVMKWXYZCILPOHERABDFG	25

Now that we have both alphabetic keywords, we can build a tableau for the Q4

key letter	**plaintext alphabet** ABCDEFGHIJKLMNOPQRSTUVWXYZ
A	DMNOLQFSJTUVGWXYAKZCBIPHER
B	FNOQMSGTKUVWJXYZBLCIDPHERA
C	PDFGBJHKRLMNEOQSCATUIVWXYZ
D	GOQSNTJULVWXKYZCDMIPFHERAB
E	AKLMJNBOFQSTDUVWEGXYRZCIPH
F	JQSTOUKVMWXYLZCIFNPHGERABD
G	KSTUQVLWNXYZMCIPGOHEJRABDF
H	RJKLGMANDOQSBTUVHFWXEYZCIP
I	HFGJDKELAMNORQSTIBUVPWXYZC
J	LTUVSWMXOYZCNIPHJQERKABDFG
K	MUVWTXNYQZCIOPHEKSRALBDFGJ
L	NVWXUYOZSCIPQHERLTABMDFGJK
M	OWXYVZQCTIPHSERAMUBDNFGJKL
N	QXYZWCSIUPHETRABNVDFOGJKLM
O	SYZCXITPVHERUABDOWFGQJKLMN
P	EGJKFLRMBNOQASTUPDVWHXYZCI
Q	TZCIYPUHWERAVBDFQXGJSKLMNO

R	BLMNKODQGSTUFVWXRJYZACIPHE
S	UCIPZHVEXRABWDFGSYJKTLMNOQ
T	VIPHCEWRYABDXFGJTZKLUMNOQS
U	WPHEIRXAZBDFYGJKUCLMVNOQST
V	XHERPAYBCDFGZJKLVIMNWOQSTU
W	YERAHBZDIFGJCKLMWPNOXQSTUV
X	ZRABEDCFPGJKILMNXHOQYSTUVW
Y	CABDRFIGHJKLPMNOYEQSZTUVWX
Z	IBDFAGPJEKLMHNOQZRSTCUVWXY

By comparing rows of the tableau with the original key table, we can fill the table in completely. The remaining keyword is in the column under the first letter of the first keyword.

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	JQSTOUKVMWXYLZCIFNPHGERABD
1	SYZCXITPVHERUABDOWFGQJKLMN
2	WPHEIRXAZBDFYGJKUCLMVNOQST
3	BLMNKODQGSTUFVWXRJYZACIPHE

It may not always be possible to fill the Q3 tableaux and recover the keywords. If that is the case, then we can take what we do learn from one of the Q3s and use it to fill in some of the Q4 key table. Then we can work on the other Q3 and see what we can add to the Q4 key table. We may have to go back and forth to one of the Q3s and then the other, until we have enough of the Q4 table to be able to decrypt a large fraction of the ciphertext beyond that covered by the crib.

Let us try this example text from the opening of *War of the Worlds* by H. G. Wells, which we have encrypted with period six:

```
012345012345012345012345012345012345012345012345012345012345012345
NOONEWOULDHAVEBELIEVEDINTHELASTYEARSOFTHENINETEENTHCENTURYTHATTHIS
KBZFPHAPGKFRLCRBQMUMOKUNFIOAYQFVOGAQARAMPNJXOWPWKKVTPNFPQCDSOKAMUQ
```

From this crib we have this information about the Q4 key table:

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	O...U...J....KA....F.L....
1	....CR.I.....XB....KPM..V.
2	.R..O..V...G..Z..Q.A.....
3	G.TKB..M...A.F.....W....C.
4	Y...P..FU..Q.....A.D.....
5	R...W..SM....N....Q...H...

Take all possible combinations of the form $q_m^{-1} \circ q_n$ (see Equation 80.1) and merge any that can be merged. Throw out any that are empty or only have one known letter.

plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEF.HI..L.NO..RSTUVW.Y. .....DL....N..... ....A.....O..... ...N.....O.T..... .....TE.....O..... .F.....Y..... ...TO..V.....E. F.....V..... E.....T..... .....B.....H. L.....T..... .....R.H...T..... B.....R..... ....Y.....E....D.H.... .....A.....L..... .....N.....L.R..... ....T...H..... ....I.....R....H..... .....A.....I.... .A.....S..... .....I.....E.....

Applying the technique of Unit 75 increases our knowledge of this Q3 to this:

plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ ...TO..VR.....F.....E. ....F.....B..O. .....B.....F. .H...Y..... .V...E.....Y..... ....YO.....E..I.D.H.... .....E.DL....N..... ...EA..R.....L.....O..... ...NL.....O.T..... .....TE....R...O..... .F.....Y.....E.... F.....V.....T. E...D.....N..T..H.....

```
....VB..................H.
L...N......T..............
........D..R.H...T........
B................R.......
...........A.E.....L......
...I...N.........L.R......
O...T...H........V......D.
....I.........R..N.H......
.....A.............Y.I....
.A...............S........
...Y...I......A....E.R....
.......OA............F....
...O...............F....A.
...HR...N..........V....I.
...R....L.................
.......DY....I...E...T....
.......E.........A...O....
...L.........O............
...........F............L.
...........D..N...........
Y....T........D...........
I....V........H...........
...V....................R.
..............B.........V.
R...H.........V...........
.....L.....Y..............
...........I.....D........
.....R........I...........
...................I.N....
........T....V............
........O........F........
.................Y...D....
.O...................Y....
................BA.......
SR......................
```

Now use Equation 80.3 with all rows of the Q4 table and all rows of the new Q3 table. We get the following new parts of the Q4 key table. Again, we only keep rows with two or more letters and merge rows whenever possible.

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	O..XU...J..B.KA..V.F.L....
1	....CR.I...U.XB....KPM..V.
2	.R.UO..V...G.BZ..Q.A.C....
3	G.TKB..M...A.FR..D.W....C.

4	Y...P..FU..Q.V...A.D.....
5	R...W..SM....N....Q...H.K.

Now we take the Q4 permutations with the new letters and convert to the other Q3 via Equation 80.2.

plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCD.FGHIJKLMNOPQRSTUVWXYZ
BU.V.KO...XM.....A..C.F...
ZG...A....BC.....OQ.U..
RAU..W....F.L.G.....BD.K..
.Q...D...U....Y.....PA....
.....MN...R.....W.....
.ZO.....V.A.C.....G..B..
.RB.....M.W.....AC.F..
..P.....F.D.....Q.....
..W.....S.....K.N..
FKL...B.....U.V...X....A
KXM...U.....C.....I...B
WF....A.....B.D...KM...R
D..W.MQ.....PBA....F..G.
.N.....W.....S....
UCV..X.....I....B....K...
GO.Q.B....U.V....Z....A...
QP.A.VY.....F..OG.....D...
.WK..NR.....S.....
V..F.....UB...J...O.
...K.I.....CU.....
.....S.....W....M...R.
.....JK...O....U...
.....V..X....I...C...
.....B....V...O...
.....C..F...GM...B...
.....U....YF...P...

Applying the technique of Unit 75 gives

plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
BUDV.KO.NWXM....PA..C.FL.G
CD.J.L...KMIN.QARU..VWX.Z.
DVJW.MQ..X.N.SPBAC...FLIG.
FKLM..B..N....U.VW..X..SPA

```
GO.Q.B..JRUDVW..YZL..PAC..
IN....J.......WLX....S..C.
JWKX.NR.....S.ADC...FLI...
KXM...U...S...CJ.F..LI...B
LMIN..D.......VFWX....S.AC
M.N...V..S.....KFL..I...BD
N..S..W.......FMLI......DJ
O.QP.U..WACV.F...GM...BD..
P.AB....LDJFKM..OQ..RUVW.Y
QPRA.VY.XC.WFL.OG.N..BDJ..
RAUC.WZ...FXLIGQ....BDJK..
S.....L.......M.N.......WX
UCV..X...FL.I..R.B..DJKM.O
V.WF..P.SLI....UBD..JKMNOQ
WFXL..A..I.S..BVDJ..KMN.QR
XL.I.SC.......DWJK..MN..RU
Y.ZG.P..C..ABD....W..OQR..
ZG...A....BCDJ.Y..X.OQRU..
.I............JXKM..N...UV
.JFK.I...MN...RCUV..WX...P
.Q.R.D..KUVJWXYGZ.I.PAC...
.RBU.J..MVWKX.Z..P..AC.F..
.S....X.......LNI.......JK
.Y.Z.Q..U.PRAC....J..G....
.ZO..R..VPAUC.....K.G..B..
..GO....DQRBUV...YF.Z.PA..
..P..C..FBD.JK.Z.O..QRUV..
..S...F.......K.MN......VW
..Y.....AG.P.B....V...OQ..
...Y.O..RZ.QPA....D...G...
.....G....O.QR....C..YZ...
.....Y.....ZG.....R.......
.....Z..P.G.......U....O..
......I.......NS........X.
......K.......XI....S....F
......M..........S......FL
......N.........S.......LI
......S..................N.
........BOQ.RU......YZ.P..
........G....O....P....Y..
........O.Y.Z.............
........QYZO.P....B....G..
........Y.........O.......
........Z..Y.G....Q.......
...........GOQ....A...YZ..
............YZ............
.............Y....G.......
..............I.........KM
```

```
..............S.........I.
.......................MN
```

We return to the Q4 via Equation 80.4, merge rows whenever we can, and see that we have picked up a few more letters:

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	O..XU...J..B.KA..V.F.L....
1	...LCR.IW..U.XB....KPM..V.
2	.R.UO..V...G.BZ..Q.A.C....
3	G.TKB..M...A.FR..D.W.X..C.
4	Y...P..FU..Q.V...A.D.J....
5	R...W..SM..J.N....Q...H.K.

By going back and forth from one Q3 to the other, we eventually reach the point where no more letters can be added to the Q4 table. At that point, we have

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	OW.XU...J..B.KA..V.FQL..D.
1	.F.LCR.IW..U.XB...ZKPM..V.
2	.R.UO..V...G.BZ..Q.A.C....
3	GJTKBP.M...A.FR..D.W.X..C.
4	YC..PO.FU..Q.V...A.DZJ..R.
5	R...WD.SM..J.N...XQIU.H.K.

This should be enough to finish with the traditional method of Unit 61.

Reading and references

Thomas Kaeding, “Classical substitution ciphers and group theory,” Cryptology ePrint Archive, report 2023/669.

Programming tasks

1. Use your routines for quagmire 3 and implement the expansion of the quagmire 4 key table.

Exercises

1. Repeat Exercise 1 of the previous unit but with this new technique.

Unit 81
Quagmire 4: recovering keywords

Recall that the permutations of a quagmire 4 have the form

$$q_n = k_3 \circ R_n \circ k_1^{-1}$$

Suppose that we take two different permutations from the key table of a Q3 and form the combination $q_m^{-1} \circ q_n$:

$$\begin{aligned} q_m^{-1} \circ q_n &= (k_3 \circ R_m \circ k_1^{-1})^{-1} \circ (k_3 \circ R_n \circ k_1^{-1}) \\ &= k_1 \circ R_{-m} \circ k_3^{-1} \circ k_3 \circ R_n \circ k_1^{-1} \\ &= k_1 \circ R_{n-m} \circ k_1^{-1} \end{aligned}$$

The result is a Q3 permutation built from one of the Q4 keys. Now suppose we try the combination $q_m \circ q_n^{-1}$:

$$\begin{aligned} q_m \circ q_n^{-1} &= (k_3 \circ R_m \circ k_1^{-1}) \circ (k_3 \circ R_n \circ k_1^{-1})^{-1} \\ &= k_3 \circ R_m \circ k_1^{-1} \circ k_1 \circ R_{-n} \circ k_3 \\ &= k_3 \circ R_{m-n} \circ k_3^{-1} \end{aligned}$$

This time the result is a Q3 permutation built from the other Q4 key. In order to recover both alphabet keywords of the Q4, we take these two combinations and apply the techniques of Unit 76.

The shift key appears in the column under the first letter of k_1 (unless we choose an unusual indexing letter), as in this example:

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	QUAZTGMIRESBCDXFHYJKLNWOVP
1	STVPJWXYZQHUAGNMIOREBCLDKF
2	HJKFRLNOPSITVWCXYDZQUABGEM
3	PSTOHVWXYZFQUALGMNIREBKCJD
4	UAGQVMIREBTCDFYHJZKLNOXPWS

Reading and references

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; chapter XVIII.

Thomas Kaeding, "Classical substitution ciphers and group theory," Cryptology ePrint Archive, report 2023/669.

Exercises

1. This is an exercise from page 183 of Gaines's book.

```
VNUXJYZDQEMPOWCKRIATLSBFGH
HSGJRKLNFPQBUIVAWCXYTZDEMO
```

2. Recover all three keywords.

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	RAIXYNCTLOUDBWEFGZHSVJKMPQ
1	AINYZCLVOUDBEXFGHRJTWKMPQS
2	HJKEFMPUQSTVWBXYZGRODAINCL
3	UDBCLEFAGHJKMNPQSOTRIVWXYZ
4	XYZTVRAPINCLOSUDBWEMQFGHJK

Unit 82
Some other ciphers are also quagmires

Recall Equations 60.1 and 60.2, where we saw that a quagmire cipher can be factored into a Vigenère sandwiched between two monoalphabetic substitutions:

$$Q(k_1, k_2, k_3) \;=\; S(k_3)\, V(k_2')\, S^{-1}(k_1)$$

$$k_2' \;=\; S^{-1}(k_3)\, k_2$$

If we can take a cipher and show that the alphabetic permutations in its tableau have the form of the right-hand side of this equation, then we can conclude that the cipher is a quagmire. The simplest case is one that we have seen already: the Vigenère cipher itself. In this case, we take

$$k_1 \;=\; k_3 \;=\; e$$

where e is the identity permutation, i.e., the unscrambled alphabet. At the risk of being overly pedantic, this choice of substitution keys gives us

$$Q(e, k_2, e) \;=\; S(e_3)\, V(k_2')\, S^{-1}(e_1) \;=\; V(k_2)$$

For this case, since $k_3 = e$, $k_2' = k_2$.

Now let us turn our attention to the Beaufort cipher. Equation 54.1 tells us that the permutations of this cipher can be written as

$$b_n \;=\; z \circ R_n$$

where z is the reversed alphabet and the $\{R_n\}$ are the rotations in the Vigenère tableau. Hence, if we take $k_1 = z$ and $k_2 = e$, we conclude that the Beaufort is a quagmire 2:

$$Q_2(k_2, z) \;=\; Q(e, k_2, z) \;=\; S(z)\, V(k_2')\, S^{-1}(e) \;=\; S(z)\, V(k_2') \;=\; B(k_2')$$

To take the key of a Beaufort and use it for the Q2, we have to apply the atbash cipher to the key.

We also saw in Equation 54.2 that

$$b_n \;=\; R_{-n} \circ z$$

Therefore, the Beaufort is also a quagmire 1:

$$Q_1(z, k_2) \;=\; Q(z, k_2, e) \;=\; S(e)\, V(k_2')\, S^{-1}(z) \;=\; V(k_2)\, S(z) \;=\; B(k_2)$$

The shift key of the Q1 is the same as the key of the Beaufort.

In Equation 57.2 we saw that the permutations of the Porta and Bellaso 1552 ciphers are

$$p_m = x \circ z \circ R_n \circ x^{-1}$$

where (Equation 57.1)

$$x = \texttt{AYCWEUGSIQKOMZBXDVFTHRJPLN}$$

and the relationship between m and n is not simple. These permutations are of the same form that we see in Equation 77.2, and so we can conclude that the Porta/Bellaso cipher(s) is/are an involutory (self-reciprocal) quagmire 4 cipher:

$$P(k_2) = Q_4(x, k_2', x \circ z) \tag{82.1}$$

The relationship between k_2 and k_2' is not so simple, and is left as an exercise for the reader.

Now we are going to see where to place the periodic affine ciphers, or at least some of them. In Table 29.1 we saw the alphabetic permutations for the multiplication cipher. There are 312 possible permutations for the affine cipher, where each is one of the M_m of the multiplication cipher followed by a rotation:

$$A_{m,n} = R_n \circ M_m$$

The ciphers that we are considering here are periodic affine ciphers in which the multiplier is the same for each of the m, n pairs of the cipher. In such a case, we can rename $k_1^{-1} = M_m$ and put the shifts into another key k_2. Then the collection of permutations of the cipher are

$$\mathcal{A}[k_1] = \{R_n \circ M_m\} = \{R_n \circ k_1^{-1}\} \tag{82.2}$$

This is exactly the form of Equation 62.3 for the quagmire 1 cipher. These periodic affine ciphers with constant multiplier are Q1s. Furthermore, it is true that

$$R_n \circ M_m = M_m \circ R_{n/m} \tag{82.3}$$

where the division in the subscript is shorthand for $n \times m^{-1}$ modulo 26. Let us call $k_1^{-1} = k_3$. Then

$$\mathcal{A}[k_1] = \{R_n \circ M_m\} = \{M_m \circ R_{n/m}\} = \{k_3 \circ R_{n/m}\} = \{k_3 \circ R_n\}$$

where the last equality is simply a renumbering of the rotations. By comparing to Equation 67.3 we see that the same periodic affine cipher is also a quagmire 2. Finally, notice that when $m = 1$, the cipher is a Vigenère, and that when $m = 25 = -1$ (modulo 26), the cipher is the Beaufort.

We have updated Figure 60.1 to show our new results and present the new diagram in Figure 82.1. The Porta/Bellaso cipher appears as an involutory Q4. The Beaufort is at the intersection of involutory Q4 with Q1 and Q2. The Vigenère remains where we saw it before: at the intersection of

Q1, Q2, and Q3. The remaining periodic affine ciphers with constant multipliers are in the intersection of Q1 and Q2. The Fuer GOD cipher remains outside the set of quagmires.

Reading and references

Thomas Kaeding, “Classical substitution ciphers and group theory,” Cryptology ePrint Archive, report 2023/669.

> Note that the above reference may use different conventions for labeling the permutations of the Porta and Bellaso 1552 ciphers.

Programming tasks

1. Work out the relationship between a Porta cipher and the shift key of the quagmire 4 in Equation 82.1 (Exercise 1 below) and write a function to translate between the two. Do this for both versions of the Porta. Then do it for the Bellaso 1552 cipher.

Exercises

1. Work out the relationship between a Porta cipher and the shift key of the quagmire 4 in Equation 82.1. Repeat for the modern version of the Bellaso 1552 cipher.

2. Verify Equation 82.3 by trying all 312 possibilities.

3. Check that when $m = 25$ the periodic affine cipher is a Beaufort cipher, by calculating the twenty-six alphabetic permutations of Equation 82.2 and comparing them to those in Table 54.1.

Figure 82.1: Updated Figure 60.1 showing some ciphers that we have seen are also quagmires.

Unit 83
Hill-climbing attack on periodic polyalphabetic substitution ciphers

We now take the last step away from constraint and toward anarchy. Here we consider an attack on a periodic polyalphabetic substitution cipher for which we have no knowledge of its tableau and for which there are no relationships between the permutations in it. We have come full circle back to the ciphers that we introduced in Unit 37.

The main idea of this attack is an extension of the hill-climbing attack on the monoalphabetic substitution cipher. We find the period *m*, and so must work with *m* key alphabets. In the earlier attack, we swapped letters in the key alphabet and kept the new key only if the fitness of the deciphered text improved. Here, however, the key space has more dimensions, and it is easy to get trapped in a local maximum. To avoid this, we will take turns randomizing one of the key alphabets and climbing back up the hill.

Here is the algorithm:

1. find the period *m*
2. set big counter equal to 0
3. set best fitness equal to the fitness of the undecrypted ciphertext
4. set the parent key equal to a set of *m* key alphabets
   (best to choose key alphabets that maximize the monogram fitness of the plaintext)
5. while big counter is less than some large number
   a. for each *i* in 0, ..., m−1
      i. randomize the ith key alphabet
      ii. find plaintext by deciphering the ciphertext with the parent key
      iii. set the parent's fitness as the fitness of the plaintext
      iv. set little counter to 0
      v. while little counter is less than about 1,000
         - copy the parent key into a child key
         - swap two randomly selected characters in the child's i^{th} key alphabet
         - find plaintext by deciphering the ciphertext with the child key
         - set the child's fitness as the fitness of the plaintext
         - if the child's fitness is greater than the parent's fitness
            · copy the child key into the parent key
            · copy the child's fitness into the parent's fitness
            · set little counter to 0
         - increment little counter

- if the child's fitness is greater than the best fitness
    - set the best fitness equal to the child's fitness
    - copy the child key into the best key
    - set big counter to 0
- increment big counter

6. output the best key

The limit on the big counter that we like is about 1,000,000 times the period squared.

Reading and references

Thomas Kaeding, "Slippery hill-climbing technique for ciphertext-only cryptanalysis of periodic polyalphabetic substitution ciphers," *Cryptologia* 44:3 (2020) 205-222, DOI: 10.1080/01611194.2019.1655504

Programming tasks

1. Implement the attack. You will find that when written in Python, the attack takes a long time. You may want to write another version in a lower-level language.

Exercises

1. Break this ciphertext. The keys have a hidden message for you.

```
PWNYPCGUUMYYUIAGGUGFEMWLRNYAHOGGHBAEWCXWPAQSXPAGNITNIXWBQFK
MLVIFHDTLSMADKMWWXXNYCQYBVWIKHWYARVTKXXAVGBGLVITNWVNMKMUWRY
QWRNEHYBCHUTNYQKYHIXWFWTQSLVYKRKZUHXZKVMQPHCYALCRWHDKGJGTNO
KSHXPAPHAGWQVGUHCXYUIAMXPKLXQRWKWNBXVTMEMGGIXWYOMADHKGFWQQY
QKYAHMHXRBYLRNYAHWRYXMZKFBEIXXIKWYJWUCCARGTKNMNMLBGEHCXECDT
VHLKIKMWQKWYSREOYOTYAHWWMLNAUXVGPHBYAHTGLVTGMYCYYNMYALCTISX
WMYVKMCVTQXXMKLVIHYBYQRYGHSTGLXXIWXPGKWTQHAUGIOMALHDTXLBXMW
YTEROKTHDJWWBYBIWOMAWXIOWOWGXSMKMQNQWWLYBHYFMHYUDTHVXTGWCCW
KWNYQDKULYAMHKYAWDEHYGTNOKAUKQGPHCUYFMYKWFGEPEOAHWWELMWMKQX
LKXZEGMDIOWKGXPGYSYAKHVYYJMTXXPGYUDKXWLYYQKYAHLWNGMSWVCTXXP
GEWVINWOGQUQYMHVZIRVKMLDJYVCZXIMWWGDJWLUUYFDTXFXZGXTGLVUKUU
XRWXMTKVDJYXQRIWLYMKMQNQWWLYBHYFMOHQDKGYWQECNZKXPGKPXWWXPGF
HCXYJMCKLDYWQXSBXGALFXRIRCGVEIRSSBGVHLGLVXWLAPTQHBGEWBIWOIK
ZQXWYQDTXCXZKOWSZYWIWLUZLXWQLRWUHOXIBDMHHUDJWGMQYCQSHYBKGXB
TVYLYBRVXBQLGBXCUEWLGFHVYQKQQWWDYAWDYBPMCWKWNIOWSGHKYHLVKML
WYWFXSMWLYBPUGVLWYWOIZIRVEHYBHBUCYPLCKMXXEHYBUEWVGMVCAMHTQB
XMKMKWNLLVOWXPAMXQRWEMGGGMOBGMNMKWYQHCJHYTNGRDWWZMAEREWLHTB
WVESMLTEHYBXHFQGMCPAVUMAUKMNSHDAZUMAMHBQWZMQHIDGUKVTEROKUWT
XHSPKLXQOYXQTGKXCWZMWSREWVLCOHZMWSRNYAHWWMLNAUXWSVGMOBSPGKP
MSMRNKMKWXIUMOBSQYYXMNMKQXBQDWHGEOMLXSBQDJWXQRWVQSUHDJWWBYB
IWOMVLWWWDKHQWSVSTAUHUGGXXSMKMQNQWWLYBHYFMCWKWBWEMGGAXWDLVI
```

MRPGESITNWLJBHFGMKWYMHLJGRTTZLLAEVXUALCYBFWYBRVTXAPKUKQXIHW
PBQBGUHVYVHLAVHCEHYBXHFQGMCPALHHUWUQGGFMNYQMDIRVGGXQAEJBTQX
PKGXPGYUMALRNMBRLJWPQXMUIRWGQOBQMGGJQSWHBKGJYJSVQOLWVNURUUN
XWYBRVNNHQSEWBIWSWWMXXYAHQSCHLYBRVTXLKGYVWSVXMOAQQVNHCKGXXE
HYBXULMSMLNKUOQYWUWYNUMMSREWFHUMWUCYAHCGYUMJHAMBWUUGKHTEMKM
MWJQSGLVIHIGJYXGGUWVNHIXWSREWLRLKWXIYHAQYQHPAPHWQLRAGWQWQOH
QYLOXCECDTAHTUSRECBXPYAHQRIUXBWPMSMRNTMKMWXWLGMVXHSREWLRLKW
XIKMLCSHWLOBGMSMXPAMWKBYQLGLLVYWFPSHOXISWBGYFLTFSWSBHKMSWKB
YQLGLLVXNUFGBOTAGFMKMLCCBXPXNFPAVZWSUHCYAWDCWKXUWXXJWOYEHYD
TJYMQEXPGUKWTLWVNNQYWWGQOMWAKELDEMKWYIOWINHCEHYBWYFMXMWAKEL
DEYQKOHQDWHOWWWHCXWQDKYONTKYCJWUQSZEXYAREWIHXUEHCKGXXAGHGAZ
HXHURXUWUWYBRVAGGYWHVYGKLDEYVQYLWIXHQDJWWBYBIWOMWVNYVGGQLTQ
YOGASVCASXXEHYGGURUGBQYGYFMCWRNHWUUAGCOKXXCYHCXZBQBGMYBSQHX
SECWXDCXZKRAGVLMSUH

Part IV
Transposition ciphers

Unit 84
Transposition ciphers

A *transposition cipher* is a cipher that rearranges the letters of a text, but does no substitutions. Many of these ciphers involve writing the plaintext into a grid or matrix and then doing something to the rows and/or columns. The ciphertext is then read off by rows or columns. Not all transposition ciphers fall into this class, however.

Detecting a transposition cipher can be done by looking for a high monogram fitness but a low tetragram fitness. The index of coincidence should be close to that of English, since IoC does not depend on the order of letters. Note that for short texts, we cannot rely too much on statistics.

There is a type of transposition cipher called a *route transposition*. It involves choosing some special path through the text and adding letters to the ciphertext as we traverse that path. Since it does not typically use a mathematical system for choosing that path, we will not be discussing it any further in this book.

Reading and references

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; chapter III.

Programming tasks

1. Write a function to return a boolean value representing whether a given ciphertext is likely to have been encrypted with a transposition cipher.

Exercises

1. Which of these are encrypted with transposition ciphers?

    a. `IENEEMEEMIENYNEICOEMHTCAGTIAYRBETHETFEIOOEHHRLELTLESGIMH NEEOEEMIENIEEINMEMOY`

    b. `EEISEHEEISEHSIEYHJENDQNUDQSOEMAYQUEQJESRUEUJFFEMPFEQUSHO JEEISEHEEISEHSIEYHJE`

c. JMSTUYTWIBCZLCBNHIPWIBPESIUNNIQDSRSZYLDDFITCLTULTCZDPIBG
DPCZLCBNHRWMNYIDSWLTUIWMBTCSWMQEMQHYIYRYTWIBCIDMQJDAJFCO
MEDMQWIHPHODPIBIDSLCSHSYSTTPIBCIDMQYMEBHISWOBMESIDSVYTRN
PHHDPIOPIYSTTLLTUTTUWFMNY

d. ITIOIIUIALBCTIOTANTSISYDTOEOHRMLESENYROZTRNHDOTENONYSIEE
DIELAYOSSDMAIRENRBHDLFDNPMGHEATVRHNYTVITFEGHIIRREMIRUOTN
HSUIFGREALHNTKISTSSIEERGUHAOEDAEONDUENEEYPEURARSOVTSHNTT
COEMRNOPEOACIGAENEMFATUAAHNIEIDSHTIPGDSGPHDUDUETNEEAEOER
EALEHEMMNNRSYRHFGDIOHRSOAATMRHNYORTWSRIOVEUNINSNLSNMSNOE
OGNRHVMTTSEMFARTLEOIABHTURE REHPTEPLSDSRURARSGSTNDBTLUCIS
NOTMNHEEFTALTHIRBNMRNYSIEMENETEDIOFSYOAFSSEHYGRGRISYDTNO
HECELRESST

Unit 85
Permutations

Suppose we have an ordered collection of objects, such as [⌂, ○, ☼, ▲, ■, ♦]. A *permutation* of this collection is a reordering of the objects. For example, we might want to reorder them to [▲, ⌂, ○, ♦, ☼, ■]. We have moved the 0th (because we start counting from zero) to the 1st position, the 1st to the 2nd position, the 2nd to the 4th position, etc. We will write this permutation as (1 2 4 0 5 3).

We need a way of combining two permutations. Suppose we have permutations *P* and *Q*. The *composition* of them $P \circ Q$ is the permutation that is equivalent to permuting the set of objects first with *Q* and then with *P*. For example, (1 2 0) moves the 0th object to the 1st position and the 1st object to the 2nd position and the 2nd object to the 0th position, and (2 1 0) reverses the order of the three objects. The composition of the two, (2 1 0) ∘ (1 2 0) takes the 0th object to the 1st position and then keeps it in the 1st position. It takes the 1st object and moves it to the 2nd position and then to the 0th position. It takes the 2nd object and moves it to the 0th position then to the 2nd position. The overall rearrangement is the same as (1 0 2).

The *identity permutation* is the permutation that doesn't do anything. It is (0 1 2 3 ...). The composition of any permutation *P* with the identity leaves *P* unchanged.

Every permutation has an inverse. The composition of a permutation with its inverse is the identity permutation. In other words, the inverse permutation undoes what the permutation does. Finding the inverse of a permutation is just like finding the inverse key for the monoalphabetic substitution. For example, let's find the inverse of (2 0 4 1 3). First, write the permutation under the identity:

$$\begin{matrix} (0 & 1 & 2 & 3 & 4) \\ (2 & 0 & 4 & 1 & 3) \end{matrix}$$

Then, reorder pairs (shown with the same color) so that the second row is the identity.

$$\begin{matrix} (1 & 3 & 0 & 4 & 2) \\ (0 & 1 & 2 & 3 & 4) \end{matrix}$$

We read off the inverse from the top row. The inverse of (2 0 4 1 3) is therefore (1 3 0 4 2).

The advanced reader will notice that what we have been describing is a group. A *group* is a set *G* and a binary operation · such that *G* is closed under the operation, the operation is associative, *G* contains an identity element, and every element of *G* has an inverse in *G* under the operation. The permutation group is noncommutative/nonabelian, which means that if we compose two permutations it matters in which order they are done.

We state without proof that any permutation of *n* objects can be generated by a series of exchanges of two objects. This also means that any permutation of *n* objects turned into any other permutation of *n* objects by composing it with permutations that each exchange two objects. We will use this fact implicitly when we attack ciphers that are based on permutations. When we attacked the monoalphabetic substitution cipher with a hill-climbing technique, we used exchanges to move from one key alphabet to another. After all, a key alphabet is merely a permutation of the regular alphabet.

The number of all possible permutations of *n* objects is $n!$.

Python tips

The `itertools` module has a function `permutations()` that returns an object containing all permutations of its argument. To make a list of all permutations of three objects, for example, we would use

```
from itertools import permutations
list(permutations(range(3)))
```

Be warned, however, that Python grabs memory for storing them, so if you try to list all permutations of more than around ten objects, it could cause problems for your computer.

We recommend that you use lists/arrays for permutations, rather than tuples, since tuples are immutable. You might want to modify a permutation, which is not possible with tuples.

Reading and references

Wikipedia:
en.wikipedia.org/wiki/Permutation
en.wikipedia.org/wiki/Permutation_group

Programming tasks

1. Write a function to find the composition of two permutations.

2. Write a function to find the inverse of a permutation.

Exercises

1. Find the composition of (5 2 3 1 0 4 6) and (6 5 4 2 1 0 3). Take the composition so that (5 2 3 1 0 4 6) acts first.

2. Find the inverse of (7 1 3 0 5 2 8 4 6).

Unit 86
Permutation cipher

The *permutation cipher* (or *block transposition cipher*, or *complete-unit transposition cipher*) divides the plaintext into blocks and applies the same permutation to each block. If the last block is too short, it is usually padded with nulls to fill it out. A *null* is a character that carries no meaning and is used as a filler. Often, `X` is used, but other choices are underscore (_) or random letters.

Let's just do an example. Suppose we want to encipher this short message with the permutation (4 2 5 1 3 0):

```
THIS MESSAGE IS ENCRYPTED WITH A TRANSPOSITION CIPHER
```

First, we should break it into blocks of six letters and pad it if necessary.

```
THISME SSAGEI SENCRY PTEDWI THATRA NSPOSI TIONCI PHERXX
```

Then we apply the permutation to each block to get the ciphertext:

```
ESHMTI IGSESA YCERSN IDTWPE ATHRTA IOSSNP INICTO XRHXPE
```

The key of a permutation cipher is the permutation itself. Its inverse permutation is the key needed to decipher a text. Often, the permutation is written as a keyword. There are two ways to use a keyword to convey a permutation: with or without repeated letters. The way it works is that numbers are assigned to each letter of the keyword in alphabetical order. For example:

```
K E Y W O R D
2 1 6 5 3 4 0
```

If we are dropping repeated letters:

```
R E P E A T E D
4 2 3   0 5   1
```

If we keep repeated letters, we number them from left to right. So in this example, we number the `A`, then the `D`, and then the three `E`'s:

```
R E P E A T E D
6 2 5 3 0 7 4 1
```

Breaking a permutation cipher by hand is done by "anagramming," i.e., by rearranging letters and looking for meaningful arrangements.

Reading and references

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; pages 9 and 64-65.

Programming tasks

1. Write a function to generate a permutation from a keyword. Allow for both ways of assigning numbers to letters (with or without repeated letters).

2. Write a script to find possible keywords from a permutation. Allow for both ways of assigning numbers to letters.

3. Write a function to pad a text given a block size.

4. Write a function or script to encipher a plaintext with a permutation cipher.

5. Write a function or script to decipher a ciphertext with a permutation cipher. Your function that inverts a permutation could be useful.

Exercises

1. Find a possible keyword for the example encipherment above.

2. What is the permutation corresponding to the keyword `PERMUTATION` if we drop repeated letters? What if we do not drop repeated letters?

3. Encipher this text with the permutation (2 3 0 1 4).

```
ROUND AND ROUND SHE GOES WHERE SHE STOPS NOBODY KNOWS
```

4. Encipher this text with the keyword `POKER`.

```
SHUFFLING A DECK OF CARDS CAN ON ONE HAND GUARANTEE A FAIR GAME
BUT ON THE OTHER HAND PROVIDE AN OPPORTUNITY TO CHEAT THE DEALER
CAN MANIPULATE THE SHUFFLE TO POSITION FAVORABLE CARDS INTO HIS
OWN HAND THIS MUST BE DONE CAREFULLY AND QUICKLY TO AVOID
DETECTION BY THE OTHER PLAYERS
```

5. Decipher this text with the keyword `REPETITION`. We will not tell you whether repeated letters have been dropped from the keyword.

```
LTRELAITNSIIAOHIETETLATCERRYNUOEIHQUNAGSFIEAETPREITAINDILTR
EELTNRSDWIO
```

6. Break this ciphertext by hand. What keyword was used?

```
MCEOWLETTHEOOLODWRSATNFRIIOTPSHINPOCWWREESEBLEILIOPRXLWFGEN
ACHFEOTCOMNOMSEPRIHLESLAWWFSEAAMONMUCSNNEOOLYHATEOVHELANHEI
NTMCIOGNETOHMNFUSFYHLHEELTSETRETTHFEOTTBXUETCEERNVTGAEHNRFE
OHMLETROHSETRET
```

Unit 87
Heap's algorithm

There are several ways to generate all permutations of a set of objects, and *Heap's algorithm* is one of them. It swaps two elements at a time in such a way as to run through all permutations without repeating any.

Suppose we have n objects. The objects live in an array $[A_i]$ where i runs from 0 to $n-1$. In the following algorithm, the array $[c_i]$ holds a collection of n counters.

1. output A (the unmodified array)
2. set all c_i equal to 0
3. set i to 0
4. while i is less than n
    a. if c_i is less than i
        i. if i is even
            - swap A_0 and A_i
        ii. if i is odd
            - swap A_{c_i} and A_i
        iii. output A
        iv. increment c_i
        v. set i to 0
    b. if c_i equals i
        i. set c_i to 0
        ii. increment i

Python tips

Functions can be passed to functions. Here is a simple example:

```
def function1(x):
      print(x)
def function2(n,f):
      f(n) # call function f with argument n
for i in range(5):
      function2(i,function1)
```

The result is to count from 0 to 4.

Reading and references

Wikipedia, en.wikipedia.org/wiki/Heap's_algorithm

Programming tasks

1. Write a function that takes an array and a pointer to another function. For each permutation of the array, the other function should be called with the permuted array as an argument.

Exercises

1. Wrap a script around your function and add another function to print the array. Use it to print all permutations of four objects. Check by hand that they are all present. There should be 4! = 24 of them, and they should all be distinct. Now be happy I didn’t ask you to use an array of five objects.

Unit 88
Factoradic numbers and permutations

Factoradic numbers (*factorial numbers*) are numbers expressed in mixed-radix form where the radices (bases) increase by one for each digit as we move to the left. Why this is related to factorials will become clear. The radix for the rightmost digit is 1, so that the last digit of a factoradic number is always 0. The radix of the digit to its left is 2, so that that digit can be 0 or 1. The radix of the digit to its left is 3, so that digit can be 0, 1, or 2. This continues as far as we need to hold whatever number we have.

Let's look at an example of converting a decimal integer to a factoradic number. Suppose we have 12345_{10} (the 10 subscript lets you know beyond all doubt that this is an integer in base 10). What is this number modulo 1? Try it and see. The answer is 0. Hence the last digit is 0. So far, we have $12345_{10} = \ldots 0_1$. Here we have placed a 1 subscript to indicate that the last digit is in base 1. We subtract that 0 from 12345_{10} to get 12345_{10}. This may sound ridiculous, by we are establishing a pattern. Now, the next digit to the left will have base 2. Now 12345_{10} modulo 2 is 1, so that digit is 1. Our number is shaping up: $12345_{10} = \ldots 1_2 0_1$. Subtract that 1 from 12345_{10} to get 12344_{10} and divide by the base to get 6172_{10}. The next digit to the left has base 3, and 6172_{10} modulo 3 is 1, so that digit is a 1, and we have $12345_{10} = \ldots 1_3 1_2 0_1$. Subtract that 1 to get 6171_{10} and divide by the base to get 2057_{10}. The next digit has base 4, and 2057_{10} modulo 4 is again 1, so now we have $12345_{10} = \ldots 1_4 1_3 1_2 0_1$. Subtract that 1 and divide by 4 to get 514_{10}. The base of the next digit is 5, and 514_{10} modulo 5 is 4, so we now have $12345_{10} = \ldots 4_5 1_4 1_3 1_2 0_1$. Subtract that 4 and divide by that 5 to get 102_{10}. The next digit has base 6 and we get a 0, so $12345_{10} = \ldots 0_6 4_5 1_4 1_3 1_2 0_1$. For the next step we have 17_{10}. The base of the next digit is 7, and we get $12345_{10} = \ldots 3_7 0_6 4_5 1_4 1_3 1_2 0_1$. Since that leaves 14_{10}, when we divide by 7 we get 2. We can go no further, and the result is that $12345_{10} = 2_8 3_7 0_6 4_5 1_4 1_3 1_2 0_1$. Some might write this as $2{:}3{:}0{:}4{:}1{:}1{:}1{:}0_!$. There is another way to find the factoradic representation of an integer by working left to right; we save its discovery for the reader. To convert our example back to a decimal integer, we use factorials:

$$\begin{aligned}
2_8 3_7 0_6 4_5 1_4 1_3 1_2 0_1 &= 2 \times 7! + 3 \times 6! + 0 \times 5! + 4 \times 4! + 1 \times 3! + 1 \times 2! + 1 \times 1! + 0 \times 0! \\
&= 2 \times 5040 + 3 \times 720 + 0 \times 120 + 4 \times 24 + 1 \times 6 + 1 \times 2 + 1 \times 1 + 0 \times 1 \\
&= 10080 + 2160 + 0 + 96 + 6 + 2 + 1 \\
&= 12345_{10}
\end{aligned}$$

Notice that

$$\begin{aligned}
1_2 0_1 &= 1! \\
1_3 0_2 0_1 &= 2! \\
1_4 0_3 0_2 0_1 &= 3! \\
1_5 0_4 0_3 0_2 0_1 &= 4! \\
1_6 0_5 0_4 0_3 0_2 0_1 &= 5!
\end{aligned}$$

etc.

Factoradic numbers are related to permutations in a natural way. Suppose we have n objects. There are n ways to place the first object. The name of the box into which we put the first object is a digit (as we call it, even if it is bigger than 9) from 0 to $n-1$. There are $n-1$ ways to place the second object, and we can record that choice as a digit from 0 to $n-2$. By the time we get to the last object, there is only one box in which to place it, so there are 0 choices, and the last digit is 0.

The mapping from factoradic numbers to permutations is called a *Lehmer code*. When the integers 0 to $n!-1$ are converted to permutations of n objects, they come in *lexicographical order*, i.e., in order as if in a digital dictionary and the numbers in the permutations are their letters.

Python tips

The `pop()` function deletes the nth item from a list. The `remove()` function deletes an item based on its value. For example, this sample code removes the letter '`X`' from the list, because it is the third element (counting from zero). Then it removes '`B`'.

```
letters = ['A','B','C','X','Y','Z']
letters.pop(3)
letters.remove('B')
```

Reading and references

Wikipedia, en.wikipedia.org/wiki/Factorial_number_system
and en.wikipedia.org/wiki/Lehmer_code

medium.com/@aiswaryamathur/find-the-n-th-permutation-of-an-ordered-string-using-factorial-number-system-9c81e34ab0c8

stackoverflow.com/questions/1506078/fast-permutation-number-permutation-mapping-algorithms

2ality.com/2013/03/permutations.html

Programming tasks

1. Write a function to calculate the factorial of an nonnegative integer (maybe zero). Avoid using recursion.

2. Write a function that converts a nonnegative integer to a factoradic number. We suggest that you store factoradic numbers as arrays of integers.

3. Write a function that converts a factoradic number to an integer. Be careful that only one object can be placed in any one box.

4. Write a function that takes a factoradic number and a length *m* and returns a permutation of *m* objects. Be careful to add 0 digits to the left if the factoradic is too short. Be careful that each factoradic digit *x* represents the xth empty box, which is not necessarily the xth box overall.

5. Write a function that takes a permutation and returns a factoradic number.

6. Write a function that uses your other functions to take numbers *n* and *m* and returns the nth permutation of *m* objects.

7. Write a function that uses your other functions to take a permutation and returns an integer *n* to indicate that the permutation is the nth permutation.

Exercises

1. Find 1000! How many zeroes are at the end of it?

2. Find the 101st permutation of seven objects.

3. Which permutation (zeroth, first, ...) is (4 9 2 1 6 8 6 7 0 3)?

Unit 89
Brute-force attack on the permutation cipher

To perform a brute-force attack on the permutation cipher, we need to be able to try every possible permutation. At this point, you already have the tools that you need.

Programming tasks

1. Implement the attack. Use tetragram fitness.

Exercises

1. Use your attack to break this ciphertext from the 2006 British National Cipher Challenge:

```
rlboy itdvs tennc rmaid toafl ubhle cneda nmoam nrdie
ficeh shoif sjmea hstsy nsiap sedsv mseel eopyl tndda
meoeb eopyl hndti ieedm artre eanne cettc oceyt hruea
yeebr iqrue ndrae eidrd tecdt eaokt hetsh asinp ndmie
aetmh uirng yenrd orocu damnm rdaan moeet frbka etohr
tdmie nrear nneaa eodst toeuk edadn yrsot hetsh fsiop
retfh fcehn aelte lotut etohn rproe gftao atemn tirnt
ihoet dorrl sisph etohn rpiom entca tionf pceer nntgi
eoalp hsocn eeisn teang tshsa drier tegar encco tirnn
rwhoe todfs rfhie rlsot ahdet repap ocaen ibfra ssthi
raqdu tionn dmhee reirt neaan oaics iintd woonn tchhi
tfhae ufeeo mpreo taaty oshmi dnmte deenp ruyao toest
eirna nrvye aveer ymnad rciun incso brdae zhlae ndair
cfeef intgi eetrh rnoof ouyno onrya mofuy ifaay oslya
liulw esawn hortt toenc arrya rotuy lrpie odafn oords
hgitn asihs yblel rroou irdge nnvoe dabro ietvh yocrt
rstio trhao lnieo onsco oomdm etrhe unvga ydabr amcmo
tonfd mahde lailr tdosr envci xxnxt
```

Unit 90
Hill-climbing attack on the permutation cipher

When we developed the hill-climbing attack on the monoalphabetic substitution cipher, we took a parent key, which was a permutation of the alphabet, and swapped two letters to form a child key. If the child key resulted in a better fitness for the plaintext, then the parent was replaced with the child. We will do something similar here.

We will make two important modifications to the algorithm. First, there are two ways in which we will want to generate a child key from a parent. One is by swapping two randomly selected elements in the key. The other is to roll the key a random number of steps. By this we mean to cut the key somewhere between two elements, and then to swap the front and back pieces. So, for example, if we cut (0 1 2 3 4 5) between the 1 and the 2, we would get (2 3 4 5 0 1); we have rolled the permutation two steps to the left.

The second modification to the algorithm is to add a margin of error. With permutation ciphers, it is easy to get trapped in a local maximum fitness. But we seek the global maximum. So we add a little leeway to the hill-climbing and allow it to take small steps toward lower fitness once in a while. A margin that works well for us is 0.15, and the probability of stepping downwards is taken to be 5%.

The algorithm does not find the length of the permutation. Usually, the key length divides the length of the ciphertext evenly, but not always. The key length must be input to the algorithm.

The algorithm:

1. set the best fitness to the fitness of the unaltered ciphertext
2. set an initial random parent key
3. set counter to 0
4. while counter is less than about 100 times the key length
    a. copy the parent key to a child key
    b. flip a coin
    c. if heads
        i. swap two randomly selected elements of the child key
    d. if tails
        i. roll the child key a randomly chosen number of steps
    e. find the plaintext by deciphering the ciphertext with the child key
    f. calculate the new fitness of the plaintext
    g. if either (new fitness exceeds best fitness) or ((new fitness exceeds best fitness minus the margin) and (we roll a 1 on a 20-sided die))
        i. replace the parent key with the child
        ii. replace the best fitness with the new fitness

            iii. set counter to 0
        h. increment the counter
    5. output the parent key

Python tips

The `shuffle()` function from the `random` module does what it says. For example, this block of code will fill an array with the integers 0, ..., 9 in random order:

```
from random import shuffle
x = list(range(10))
shuffle(x)
```

Programming tasks

1. Implement the attack. Use tetragram fitness. Experiment with changing the margin and the probability of stepping downward with some ciphertexts of your own choosing or making.

Exercises

1. Use your new attack to break the exercise from the previous unit.

2. Break this ciphertext.

```
STIOTASDGUTNINEFILMISTGEMEINNSATDESETASKSILUTTOLBSNLCITESYL
NOELFVOETORMHRLYUCGIOVNEROOEKGTTPNETECOLERMOIRBDEOMERGEITNT
HERIPMWAIIDNRNKHEGMTOSESOWMNTNEHBETHCELSAKNODSHWULMNIDTEAEI
TDHVOUBWEOLDLNCGALITOLTESDTEHWEIMPAAIRAGTUNSISJJPTTAUMHEOFT
ELNHTEADTSPNTATEHIOGTERWHHYTITRNODUHANUSROYOPOHUISYIYBORNGK
EUSRNETHITNIGTSBTUITPVHIEELHSCTTRUAETAHTRYILVLDRYIENSOUNEST
AELOESTDTHERIPMWAAXAXGIN
```

Unit 91
Matrix transposition

A *matrix transposition* is the simplest form of columnar transposition cipher; hence, it is also called the *simple columnar transposition*. Encipherment is done by writing the plaintext into the rows of a matrix, then reading the ciphertext from the columns. Typically, the plaintext fills the matrix; if not, then nulls may be added. The key is the size of the matrix. Since we know the length of the text, we often only need one number to specify the key.

This cipher is also called a *scytale cipher*. The *scytale* was a rod around which a ribbon was wrapped. The message was written on the ribbon. When the ribbon was unwrapped, the letters on it would be in the same order as after a matrix transposition.

For example, let's encipher this short message with a 6×8 matrix:

```
THIS MESSAGE WAS ENCRYPTED WITH A TRANSPOSITION CIPHER
```

The message contains 47 letters, so we will add one null to the end. Written in the matrix, it looks like this:

```
T  H  I  S  M  E  S  S
A  G  E  W  A  S  E  N
C  R  Y  P  T  E  D  W
I  T  H  A  T  R  A  N
S  P  O  S  I  T  I  O
N  C  I  P  H  E  R  X
```

We read off the columns to get the ciphertext:

```
TACISN HGRTPC IEYHOI SWPASP MATTIH ESERTE SEDAIR SNWNOX
```

Reading and references

Wikipedia, en.wikipedia.org/wiki/Scytale

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, page 82.

Programming tasks

1. Write a function or script to encipher a plaintext with a matrix transposition for a given size of the matrix. Allow for the possibilities that nulls are or are not added to fill the matrix.

2. Write a function or script to decipher a ciphertext with a matrix transposition for a given size of the matrix. Allow for the possibilities that nulls are or are not added to fill the matrix.

3. Write a function to find all factors of an integer. We are using small numbers, so it is not a problem to do an exhaustive search over all numbers less or equal to the square root of the integer.

4. Write a function or script to break a ciphertext that was encrypted with a matrix transposition. It would be reasonable to check all factors of the length of the ciphertext first, and then to check all other possible matrix sizes, in case nulls were not added. Be careful that you place the empty parts of the matrix correctly.

Exercises

1. Encipher this text with a 12×15 matrix. To be clear, this means twelve rows and fifteen columns.

   > Call me Ishmael. Some years ago, never mind how long precisely, having little or no money in my purse, and nothing particular to interest me on shore, I thought I would sail around a little and see the watery parts of the world.
   >
   > (from *Moby Dick* by Herman Melville)

2. Decipher this ciphertext with a 13×17 matrix.

```
TITOSNHSSSFSIODONUDIESOEESMEFGRAMTPFEETAWWHVLATIBTMEPAHAELN
LROHENPLINYGDEIAIDCEKTDELADTRGHEAWELDAMDTDHOBRIWETDEOHFLLUE
TADHNLWIEILTDHSBEEANRNFOAOAHRFSNUTCEWBNBAUESCPYETAUTUNSNLHO
YNONRHCDHCEONAIHDDEKAHEFLHRNIEESELEATYIDEMXN
```

3. Break this ciphertext:

```
OONEOLDOSHEHMOSNKDDNISMTADIPFHEEHIHFEEIRRSAHEMFINITEDFDEMRI
LORMTSEHAFLAAEMOROSOADINSYDNDWONMEARHSDEAYYWAKITLHMIBDCBTLI
VENRFOOSRITLOETEDGOTRUHOVIESGHDWWURRREWIOTLSTAHHBAILANDNOIP
HBAELNBIDBESCDIEOTNESLKAEDTOETSUSGDFEELLBHVOIITHRDOVBILYEEF
FZHAERREATYABRFUEEPGEMRCTSREIALOLPOAEMKLLCDTNLFPERMDIAEIHDA
YYTLNSSINNHGEINMTHEEAHNHDEHSNDOHESDMEHEICTIGSMIRSTSFILFOLNW
EENELOAOSAHUYTAENCSYMWUBYELDOSMTOTAE
```

Challenge

Think about triangles.

```
IAAUSNPOETOHAUNRGSXTGNTQDSRDERENLSILXROGTUOOEAAYTDAOAEIOLHA
FNWNSARTTFNEDEARTEEDHBIRITDSSTEHOLCOOAIOHTBIFIFLARULBAOUAON
YRNTTSRRTMRGOOWSTDYIJTPUURTTATUHEINIOBTSARSDTOUTTHMHURTKAIT
NOINTSOKROOIF
```

Unit 92
Twisted scytale

The *twisted-scytale cipher* was introduced in the 2011 British National Cipher Challenge. It is a modification of the matrix transposition (simple columnar transposition). Here's how it works. First, write the message into a matrix, with padding if necessary:

0	T	H	I	S	M	E	S	S
1	A	G	E	W	A	S	E	N
2	C	R	Y	P	T	E	D	W
3	I	T	H	A	T	R	A	N
4	S	P	O	S	I	T	I	O
5	N	C	I	P	H	E	R	X

Then, we roll each row to the left a number of steps equal to its row number (starting from zero):

0	T	H	I	S	M	E	S	S
1	G	E	W	A	S	E	N	A
2	Y	P	T	E	D	W	C	R
3	A	T	R	A	N	I	T	H
4	I	T	I	O	S	P	O	S
5	E	R	X	N	C	I	P	H

Read off the ciphertext by columns:

```
TGYAIE HEPTTR IWTRIX SAEAON MSDNSC EEWIPI SNCTOP SARHSH
```

The key for such a cipher is the number of columns.

We can modify the cipher to "twist" the matrix by an arbitrary number.

Programming tasks

1. Write a function or script to encipher a plaintext with the twisted scytale. Allow for an option to change the twist.

2. Write a function or script to encipher a plaintext with the twisted scytale. Allow for an option to change the twist.

3. Implement a brute-force attack on the twisted scytale. Remember to allow for different twists.

Exercises

1. Encipher this text with key 6 and twist 1.

> One morning the old Water-rat put his head out of his hole. He had bright beady eyes and stiff grey whiskers and his tail was like a long bit of black india-rubber. The little ducks were swimming about in the pond, looking just like a lot of yellow canaries, and their mother, who was pure white with real red legs, was trying to teach them how to stand on their heads in the water.
>
> (from *The Happy Prince* by Oscar Wilde)

2. Decipher this ciphertext with key 7 and twist 2.

```
OHVYLLINEYKBHEBHVMSUAMLRAFNATLNSABDRTPKONEETILEDAPSEEDEEEIT
LDEYIGLTFELDATYEWHPDHERRHTOHSWTIFHDHWWTBRETINAYHRROHDTHRAEE
TENELETWIHATXOASIAIHEEDHNHSIEIAWOBRESIHDORIFECLSSAYOWRBTNSN
TATHETEHNDVBMTEATLSASTXGTWIEWFDOEEEUADOLHUIERASEGOEIYHECEET
HTAXHTOTAARGNGESTYFROEOFHERPWSWRGEANODNTAOLT
```

3. This is a ciphertext from the 2011 British National Cipher Challenge. Decrypt it.

```
TsormynhemuitlThesynpddmrdtcryefclseanpinptbsslenednulfteom
atesdepaeestvsnysdsnmesirutnitgapsarmdxcdusrVerlrnitestixpu
rhepobakmsiarieiyrettFGcpgalitdwideddihdeivpotreesrtCFDdtha
ahthVaptetftddthrthapngteycitfehplceCraebinatektadrytioosor
ngoxihIecfgestnoeeIesphaaetoreeiecaaabetvdrotliusiiWcrgmran
hrnmuitiinongrvheenitiecrsicmIhcohenhotiiTtrystdXcuyatehyfh
tboseatoneietictysievenheraoomunnasctXsbyfafspelaeSxtiagsGw
enepnefaanredsfwasleitenhotocoipscWiseemoncblsstetiniccsevr
nstldtfwmomuoisTecinmwcTtirvrrheerneserrntchemofhooningiteo
evecenalnowncgsdraieddanesanysnalntrnasoymennhispiaADisssIl
tworhylsqnrsstnnhcaiTgsaeptelssrndsotndahrenairlaelinyoahca
mhfouodhdXatnbesioeryaeseysoybhtuftheoitsdtpesyeeteessbgloi
anemachduimaertsrnonmocsyhaecoaetatimodeinsaseaeforieitonte
nhavntieiatcinorlttemcyevcgmgudpamufthypntgarwtgetetnndFpu
ofclhorbesybudaitegfemlohuicdrineAtreiithealVeptmepatcyednf
memlealaltisVDctdXrenfdesltocotgruihrihelurhiehesphenomseca
dsmmlsiatreerierisstdditpmatsinmpncimoeruusnntasIluyhuetexa
feelciserslontiocsnkuhallsehaiustoietgesoreetderaeheleenega
aGhalaawekdosixiatnsiptroudnedshbesgtDhesncessreanreorsnrta
ptooplufnrptdegrFTthecenloinaittufegatnesterfostmviTtiavadi
tarneanelinfiowvensx
```

Unit 93
Columnar transposition cipher

The *columnar transposition cipher* generalizes the matrix-transposition cipher by permuting the columns before reading off the ciphertext. The key is the permutation, which may be represented by a keyword. Repeated letters in the keyword may or may not be dropped.

Let's do a quick example. Suppose we want to encipher the following text with the keyword `KEYWORD`.

```
THIS MESSAGE WAS ENCRYPTED WITH A TRANSPOSITION CIPHER
```

First, we arrange the message into rows with the same length as the keyword:

```
KEYWORD
-------
THISMES
SAGEWAS
ENCRYPT
EDWITHA
TRANSPO
SITIONC
IPHER
```

The columns are then permuted. The permutation corresponding to the keyword is (2 1 6 5 3 4 0).

```
DEKORWY
-------
SHTMESI
SASWAEG
TNEYPRC
ADETHIW
ORTSPNA
CISONIT
 PIR EH
```

Notice that some columns are shorter than others. The ciphertext is then read off in columns.

```
SSTAOC HANDRIP TSEETSI MWYTSOR EAPHPN SERINIE IGCWATH
```

Of course, we remove the spaces to hide the column lengths from an eavesdropper.

```
SSTAOCHANDRIPTSEETSIMWYTSOREAPHPNSERINIEIGCWATH
```

Reading and references

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; chapter VI.

Practical Cryptography, practicalcryptography.com/ciphers/columnar-transposition-cipher

Abraham Sinkov, *Elementary Cryptanalysis: A Mathematical Approach*, 2nd edition, revised by Todd Feil, published by Mathematical Association of America, 2009; www.jstor.org/stable/10.4169/j.ctt19b9krf; chapter 5.

William F. Friedman, Military Cryptanalysis, Part IV: Transposition and Fractionating Systems, Washington D.C.: U.S. Government Printing Office.

American Cryptogram Association, www.cryptogram.org/downloads/aca.info/ciphers/CompleteColTransposition.pdf and www.cryptogram.org/downloads/aca.info/ciphers/IncompleteColTransposition.pdf

Fred B. Wrixon, *Codes, Ciphers & Other Cryptic & Clandestine Communication*, New York: Black Dog & Leventhal, 1998, pages 152-153.

Fletcher Pratt, *Secret and Urgent: The Story of Codes and Ciphers*, New York: Bobbs-Merrill, 1939, chapter V, sections III-IV.

Programming tasks

1. Write a function or script to encipher a plaintext with a columnar transposition cipher.

2. Write a function or script to decipher a ciphertext with a columnar transposition cipher. Be careful of where to place empty spots in the matrix, if any.

3. Make a copy of your brute-force attack on the permutation cipher and adapt it for the columnar transposition cipher.

4. Make a copy of your hill-climbing attack on the permutation cipher and adapt it for the columnar transposition cipher.

Exercises

1. Encipher this text with the keyword `PERMUTE`.

> "You are old, Father William," the young man said,
> "And your hair has become very white;
> And yet you incessantly stand on your head—
> Do you think, at your age, it is right?"
> "In my youth," Father William replied to his son,
> "I feared it might injure the brain;
> But, now that I'm perfectly sure I have none,
> Why, I do it again and again."

(from *Alice's Adventures in Wonderland* by Lewis Carroll)

2. Decipher this ciphertext with the keyword `COLUMNS`.

```
WNIONOIENNANNACTTEITTAITWLWCAHOEEHANSOYKSGCLSKTHLCSYEOEAEGE
AAEEOBNMOAEBEWGATLPEISEFEUMANDBOSAWUPODTCREAONLDRTTDDAETTNT
IRQSTAYDDENOCELIUORAOTCWOIWOTSALIMSEANCADCNTASGLMIHHRNHYVNL
HLIAYTSNMOGETNODHOIUCIISIECSFTWLNRFUSMAKSRGHKLSJINOKECWAPOD
BEWEUICUFTPUCRSDFSLDEFNBANEEGHI
```

3. Brute-force this ciphertext.

```
OTENAYOSTAEPFLEUSEHKTFQEEIROGOSAEUCODWNREETFUTRYFOTGHTAYETN
ANTELETCBMTMSEAAMINFHUSRCEAAEASLVALSNNASNUEUSTEBSKPTLBEVAHH
UTUOEUFHOEELNINSEETELUEEOIRLWEOHTRHHMSNYURAEKYYTSPSOELOUDTR
SISZRTEPETEEHOESITAEEHSNBSOENYEATSETVTOMASHNMFEDONUTHFEEAIG
ISOEAUNMEMHEOOPOSEIDEEOEOEHHLNAADOSMIHTTQEMORJETORSEOSEEUGI
HNJNVMTEUTMYGEUDTRSIUYYDLOUEISLOLYCRPDENHYUYYGTOITHNCROVTIO
SZYGITBEGIUHOIHROROTIDOAYNHOEOSIESNREDG
```

4. Break this ciphertext with your hill-climbing attack.

```
UHOEIHSHIBMEWTRNWBETDHTOHSYSHTHNNBETWEROHTWAOTDDROTLCOWHOWY
ETEIFWACTENEDEEIEQTSTRCDFMIESTLRSELITUIRTDRRETRLDCAAETOBECD
MATAOTWNTSGLIDVSKISAGTSCTEENHWNLESUEDBSTFESTRHSESWLEWYYLEUU
LSSIRNHEEDYLDRRACTHTNGSAENIOEWEIHMEOAOPHUTEDCDTHENARHUEAAKS
LGICNWRFGAANINANAMESEABSHIDAEIHLETNIUBEOHAOEEAHAIYFEHASUHSR
YUSOACAEBEIRHRRLANAEWGATPANEUIAHELASATNWFFDUTRDHLTIITPADETR
BHSLAANRAAOONOHEESNEIIHHDRTTLODTIDUAIFGMSNIAEHUOSTEEAOVDMIE
ETBARSUNABEUNYDFOAEOTHUHEEKOAYITOCTOIWLRYESEPTHEOOWSACEEUAC
TSIEWDELTPNAHYWOTTUVDHSSGSNTITHYOHOOBASDSSDTHOSGLUTTNTIORRY
NEESDATCENEDCOCENDTNSLVDWBOESENRISNWKHSHIFFEYEINWVSIAAUSAAT
OTABIAEHTAAIEEWLTLEBETHHRETHAIHHWLGIETELMNHHOAWMOIEAIKCHGSG
USHTDDSUIASWEDSEAODEDSOIKREGERNSTRTPWLLHELYTUNSDSNAAVDSOPRY
YPESHYGLOSENDBTESSPNAAKTNHNIEOINC
```

Unit 94
Double columnar transposition cipher

The *double columnar transposition cipher* is the application of two columnar transpositions in sequence. Its key is two permutation, which can be represented by keywords. It was a popular cipher during the two world wars.

A hill-climbing attack on the double columnar transposition can be constructed by modifying our attack on the single transposition. There will be two parent keys (one for each transposition) and two child keys. In each step in the key space, we randomly choose one of four options: swap two elements of the first child key, roll the first child key a random number of positions, swap two elements of the second child key, or roll the second key a random number of positions. Unfortunately, there is no way to determine in advance what the two key lengths are, so we often have to try several before we succeed.

Reading and references

NOVA Online, www.pbs.org/wgbh/nova/decoding/doubtrans.html

Wikipedia, en.wikipedia.org/wiki/Transposition_cipher#Double_transposition

Solomon Kullback, General Solution for the Double Transposition Cipher, Washington D.C.: U.S. Government Printing Office, 1934, www.nsa.gov/Portals/70/documents/news-features/declassified-documents/friedman-documents/publications/FOLDER_439/41751169079035.pdf

Fred B. Wrixon, *Codes, Ciphers & Other Cryptic & Clandestine Communication*, New York: Black Dog & Leventhal, 1998, pages 157-159.

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, pages 301-303.

Wayne G. Barker, *Cryptanalysis of the Double Transposition Cipher*, Laguna Hill, California: Aegean Park Press, 1995.

Programming tasks

1. Implement the cipher. Keep in mind that the two transpositions are done in opposite order when deciphering.

2. Implement the attack described above. Copy the attack that you made for the single transposition and make the appropriate modifications. Experiment with the margin; can you manage without it?

Exercises

1. Encipher this text with keywords `SAMPLE` and `KEYWORD` (in that order).

   At the Carlton news stand West bought two morning papers, the Times for study and the Mail for entertainment and then passed on into the restaurant. His waiter, a tall soldierly Prussian, more blond than West himself, saw him coming and, with a nod and a mechanical German smile, set out for the plate of strawberries which he knew would be the first thing desired by the American. West seated himself at his usual table and, spreading out the Daily Mail, sought his favorite column.

   (from *The Agony Column* by Earl Derr Biggers)

2. Decipher this text with keywords `TOMATO` and `SOUP`.

```
EOONPGAEPOGOCPEPCRWEUNNELMRDNIOOLFEIAOTTASCRCCATERWDGOSNNSK
NGIDOEIDFEELLPALIHNNTISYDNNRSHPFEANOINDOEIELVDERSMLOIRRDROC
SLYETEEWFRGEIEHWILIERNALNOSHCIHACEEBIUEBROPVNNREDEOPRMPIROO
EAMHUIOCAESIOJSSSOODNROTKONAESONAADWRIGDOLPEERETOEERTAPHETA
DEOYRTFPKOLGDDHCTTDETVAAELLSCUAEDTRRRUPAALSUOECDLTCDNPNCQND
ARSAIUANEPSEOOSAKYYBIRRORRLHKIFTDCEDNPGOCIRETROYTCOOOTYOGNO
DHTLOSUUGPPNBACONWSTBRHGMPEETEDENIMSCGTAETAOOUAUNPDTEDEOR
```

3. Break this ciphertext. The key lengths are five and six.

```
VNRWGNCARTTNONRRWOODRLIOOSWPAEHTHUNDCTTIAHIRLASDIHNTHIAHOHA
NMSNFTEUECSNDEKRROSFGNUEFSRFODGIHONNIEETHEROAOALBREEUIGONFE
SINAMTOFLBETTWWNSNAFMTOECLEEAEAEHTYTNIGVETARRNILTOEAHENATHG
LIDDJDADGHWTOTRSEOATTSLDHNHFWCOLEIWIEDIDAAARSOBRATIHGFSEDAR
OCORAISYRLVTIEFTSAERERFNLRATVTSNTRAIWNSESENVQTDODTSOOLLNGFA
FIMBAARENEOMEIEWEUKUONLOSISALHOVINSTBEESLRETHBDISESGDAEVTOR
GHRAEDOOIGRHDSAOITITOEIATHTATBHUALTAISHBTLTHDERAETTJKEDOELC
EWETOSREYEEDGTGTSRWTHSLIWNWNADNOEOPOETOIIEONOAWCYNITITEOTBW
EOAFRRIHEAFAHPETOIRTTIOSADAAOHOAHLIWETYNDEESIHICNCOOEFUAEAT
WFAETTKEHESOOOITUINTSIAJLDEHWUWEYS
```

4. Break this ciphertext.

```
TTFNOEINTDDRNTBEOSROHOYEIAEFAIEYFEOREECTTHSITEROTOPPTNRPROC
GHSNIFTJPOEEGHSCPUKENDDIHSIIGETILSEOETDIEOPHIHBIANDDENMUITM
HNTEOLTCEAACSPTOUIETBHTOUAOOASOOHEUVRRHRNOTNYRNMCIOOAEOGHEE
MTTOSNDTENDIAORNSVOAESDNROTPLCHTTMRAHHOSCNIMCFEMRLMNGIERETT
EDTEMZBTMRHPNTVHESGSHEEEIVUEHEAYUIPLHIEAITHRLHNNOWNEEEWTEDE
PRTTRUBIIETOONOAAEITSEFSODDLMCALWOERTHUTIYELDTOBWTTVDIATURT
RRYIOYIOTEIFTPSTORTAWFOGIEOSTESDOSBAIIIO
```

Unit 95
Improved hill-climbing attack on the double columnar transposition cipher

Remember that a double columnar transposition cipher has the action of one transposition with key k_1 followed by the action of a second with key k_2:

$$c = T_D(k_1, k_2)\ p = T(k_2)\ T(k_1)\ p$$

Between the two single transpositions is an intermediate text:

$$c \leftarrow T(k_2) \leftarrow t \leftarrow T(k_1) \leftarrow p$$

There is a newer attack on the double columnar transposition in which hill-climbing is done on the second key. Each possible key k_2' is scored based on the highest digram fitness that can be found for the intermediate text if we break it into columns based on the length of k_1 and putting those columns alongside one other. When the text does not fit into columns that have the same length, we need to try several possibilities for how we break t up. Once the correct k_2 is found, the first key is attacked, although we might already know what it is at that point.

We can further improve the attack by adding new ways to modify possible keys. Heretofore, we have been swapping elements in the key, which we can view as a sequence of integers. But it can be fruitful to try lifting a chunk of the key from one place in that sequence and putting it into another. This has the benefit of occasionally preserving a portion of the key with a large contribution to the fitness.

Reading and references

George Lasry, Nils Kopal, and Arno Wacker, “Solving the Double Transposition Challenge with a Divide-and-Conquer Approach,” *Cryptologia* 38:3 (2014) 197-214,
DOI: 10.1080/01611194.2014.915269

Programming tasks

1. If you are ambitious enough, implement the improved attack.

Exercises

1. Break this ciphertext. Do so with the naïve attack from Unit 94 and with the improved attack in this unit. Were both able to succeed? Was one significantly faster than the other? The keys are random and have lengths nine and thirteen.

```
COHMIETOSBOASTNXWOPDAUEONSDYEREOEELTNEOGRUAKUMBGEFOIISENSOC
YHEODATPEHIHIHOQAAETHEPIEINRTASNHKAOBEETTTMCEWHASOROIRAOSUC
CMEILAAEHLTCGUONSLAAEETTODTHIHOUTCERUMIEGSATMUNHIWEEATTAPBR
CEAAIHHITGLEUMKTSGOLNIOLDNBWCROOITITCEOBLINFYEOOOLTOIHTNUII
FRSMEFSGTAAHTSTBNDDAGAONGHLOAUITRRIHEOTARHAAENEVEHYSRSCSHOU
CSISSOHHKSNOAOTFUBREGHRRIFOERTTBSAIAMGEITOHTOSAHFTSRUEYRDTC
SIMMMTWTTOOSARCNNTDLNICOANATORYOTNSEAOTMNHHAQLTBPASRPRTERMO
AEEFYTNNEHSNNALUSTGSYAERTDWWEAFROASESWOAABPNRDLTREP
```

Unit 96
Nihilist transposition cipher

In the *Nihilist transposition cipher*, the plaintext is written into a square matrix, with padding if necessary. Rows and columns are both reordered with the same permutation, which can be represented by a keyword. It does not matter in which order the two operations are performed. The ciphertext is read off either by row or by column. If the plaintext is too long to fit into a grid whose dimensions are the length of the key, then it is broken up into blocks, each of which is enciphered separately.

Do not be confused and assume that this is a permutation cipher followed by a columnar transposition. Both the permutation cipher and columnar transposition shuffle the columns when the text is laid into a matrix. However, in the Nihilist transposition, rows and columns are both reordered.

Reading and references

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; chapter IV.

American Cryptogram Association, www.cryptogram.org/downloads/aca.info/ciphers/NihilistTransposition.pdf

Fletcher Pratt, *Secret and Urgent: The Story of Codes and Ciphers*, New York: Bobbs-Merrill, 1939, chapter V, section VII.

Programming tasks

1. Implement an encryptor for the Nihilist transposition. Allow for both choices of how the ciphertext is read from the matrix.

2. Implement a decryptor for the Nihilist transposition. Allow for both choices of how the ciphertext is read from the matrix.

3. Implement a dictionary attack on the cipher.

4. Implement a hill-climbing attack on the cipher. Use parts of your attacks for the permutation cipher and columnar transposition, and make the appropriate changes.

5. Now suppose the matrix is no longer square, and that rows and columns are reordered with different permutations. Copy your hill-climbing attack on the double columnar transposition cipher

and modify it for this new cipher. Again, allow for both choices of how the ciphertext is read from the matrix. Also remember that a long plaintext may have been enciphered in several blocks.

Exercises

1. Encipher this text with the Nihilist transposition with the keyword `BOYNIHILIST` (do not drop repeated letters). Use '`X`' for padding. Read the ciphertext off by rows. You will need to encipher it in two blocks.

   > But outside of his being an enthusiast and a lover of liberty, he was not known, and had never taken any prominent part in any of the social or political movements of the day, beyond sympathizing with the struggles of the working men and women of the world in their struggles to better themselves.
   >
   > (from *The Boy Nihilist* by Allan Arnold)

2. Decipher this ciphertext with the Nihilist transposition with the keyword `SIBERIANGULAG` (do not drop repeated letters). The ciphertext was read off by columns.

```
RHFHLADAEXGTIHDCOFNCRDDMETSHHPHOIANXEDMOOERECSYDXRFEUEDTUHA
HTGNAHEHANFCOOEDANFIIEOLDHYEEEATOVEICWRGONIFRFUEIDHMTIPORBN
ARMTTELDXMHMJAPWIEMUVIKDOESRMTNEWAYVNOTLSSIAYHSAFIU
```

3. Break this ciphertext with your dictionary attack.

```
OIUDYOHENTTIAVAFMELLWIORILDWANDOOULYYOSTRUONEYUBRTBELIITANY
CEOUSCAANUMFHLIHETTNDSAVEUKYOENHAWTNOUBDODAWHNDTANDNAMENDME
WOSOEDDRVERAFBRTBELIUNFHYOLLWITHWEUSYOSTRURTAIDFANDUENEPOUN
YPOSBERETNOGKURODYBERTPNSYIBETOAREVUETRNEEALBATYMASANIXOWIL
TWDIINPRWSSTMACEMSHICHGTINEENDDAINOAPRAPORRFTEOFUTYOHTIGLRY
OLLWISSPAURALRTPOOUTYHAYOVEHAEIOPELHAISNHETACPLNVSEHIARWEUS
TOMETONDHAORXXITLELSEEONEEOSETLOALNDNONGWI
```

4. Break this ciphertext with your hill-climbing attack. What are possibilities for the keyword?

```
THTATEMMORAUBTILILKEOTYODVETRENUTRNOAHNTIHIILATSIHNNISATNOI
SILOUTAASNDSSSASXSWXXSOHSETADLLFROTOOEFLL
```

5. Break this ciphertext which was encrypted with the cipher described in Programming Task 5. What are possibilities for the keywords?

```
AMSAIBSOIHERQKOCNEISTATHTERDUSERFPAPMTETRWIOTERSIUATTSUIEMC
AITCUBUEPVIHWANONOEETCNEOCONTNASTHISTSHSEEBAYENIELTNULNUDLR
UOFYOGOAMNOSIHNTYOPLANOGFNIIRTWREVRNITNMERSDAOONADSGFFEIDNA
YMYNAYMOASPYAAFCERNISDNTNWERFFEIDDNEETOTTECERAEHISNIEDOURTI
ONTDRIIAETKHOOOFLTATRHEDEHBTHIHCWNBTESUIARTAATCTASWMOLWTHUB
EKNATMMOOCRLEACSNTOCISNEHROITSHOITNTSONOIREAGSOTTSTCIURVTAC
```

ATSRBAEJSOTSSIERFOANOEISROMIPEHOTTSTTDONAAHDTISDCAEROMTSECW
NAONYNAHTETSCUTDOWNHOITNIEITNRIFTROOSLKIOOLSIISHTAEKAHBCIEE
DNSHTIEKAMNAHYAWSOHSOOHIHCWTDNTIFHGTIMOTDEVESAUOONFOTHDALUA
THARVIELEEMSOSSKAOORRAAWLASOCDNBAHNODCELAWRETSSACADGNAESVTN
ROEFRHTEAETNIAUTOKNOHWOONTAQERITDSUFODYTNAWMALEFSMAESLPCUHS
WHEIKADLMUOHEOHTAPRERROTFETTONAROPDATSEEMITOPEAODLAXXXXXERC
EHEXDIMSFOON

Unit 97
Railfence cipher

In the *railfence cipher*, we write the plaintext down in a zig-zag pattern that runs over a number of rails. With no offset, we begin on the top rail. With an offset, we skip some positions before beginning to write the text. The ciphertext is read off one rail at a time.

Let's do two examples. We begin with the plaintext:

```
THIS MESSAGE WAS ENCRYPTED WITH A TRANSPOSITION CIPHER
```

The first example will not use an offset. Let us take four rails. We write the text onto the rails:

```
T-----S-----A-----Y-----I-----A-----I-----I----
-H---E-S---W-S---R-P---W-T---R-N---S-T---C-P---
--I-M---A-E---E-C---T-D---H-T---S-O---I-N---H-R
---S-----G-----N-----E-----A-----P-----O-----E-
```

The ciphertext is read off in rows, starting from the top rail.

```
TSAYIAIIHESWSRPWTRNSTCPIMAEECTDHTSOINHRSGNEAPOE
```

Now let's repeat the process with an offset of five. We skip five positions when writing down the plaintext.

```
•-----H-----S-----S-----P-----T-----N-----T-----P---
-•---T-I---S-A---A-E---Y-T---I-H---A-S---I-I---I-H--
--•-•---S-E---G-W---N-R---E-W---A-R---P-S---O-C---E-
---•-----M-----E-----C-----D-----T-----O-----N-----R
```

Read off the ciphertext to get

```
HSSPTNTPTISAAEYTIHASIIIHSEGWNREWARPSOCEMECDTONR
```

Reading and references

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; page 12.

Practical Cryptography, practicalcryptography.com/ciphers/rail-fence-cipher

Wikipedia, en.wikipedia.org/wiki/Rail_fence_cipher

American Cryptogram Association, www.cryptogram.org/downloads/aca.info/ciphers/Railfence.pdf

Fred B. Wrixon, *Codes, Ciphers & Other Cryptic & Clandestine Communication*, New York: Black Dog & Leventhal, 1998, pages 139-141.

Programming tasks

1. Implement the cipher. The number of rails should be a required part of the key, but the offset should be an optional parameter.

2. Implement a brute-force attack on the railfence cipher. Note that the number of rails needs to run from one to the length of the text, and the offset needs to run from zero to twice the number of rails minus two (with a maximum of the length of the text). Use tetragram fitness to determine when you have found the correct plaintext.

Exercises

1. Encipher this text with four rails and no offset.

```
DO YOU SUPPOSE THERE IS ALSO A PICKET-FENCE CIPHER?
```

2. Encipher this text with five rails and offset seven.

```
HOW DO YOU SUPPOSE A PICKET-FENCE CIPHER WOULD WORK? THE PICKETS
ARE VERTICAL, AND THERE ARE A LOT OF THEM.
```

3. Decipher this ciphertext with three rails and offset two.

```
YEYITTIAAELSEEOIHAESEBERADUTOORALBLEEHTHSSHTRIFNEOKLKULSYUH
NTELITXILKTEABDIERPEAONIDULEVAIWALCOINSTKPNTIHRWWPRD
```

4. Break this ciphertext from the 2014 British National Cipher Challenge.

```
PRSAO EGERA UIADM WEHDN ISNRA SAWUA AESSR EFGDO SOGVO RBEEE
AARTE SCTDF MENUI BRTTL MEYTU MTMEU AIKWH UTKWE RWAHM NPWRA
EESON ONESE BATOI HACIN EETBR OTADA KTGFE ESYIO FLTTL STIIA
EOSVI EONSR RTAUP MNNOA ENCOC NUVRS CLVDR GCTAI IHRIC IAIHR
SDUOM RLEMC RNGLE OMARF HIUEW HALCS ASRAC UFRAW WSMEH ULSTO
AOHCE LETMT OILSE PDMUM TPTRS LYRHH NTPAN WPMOA DPPDW BESEO
ASSLT MLPES LETUN CORER LCLIT AOSVS INIIF WSEAF ORTAA DUYEN
ENONN SOPFH ONTWK OERTC SLYVO EIOHL UFOEI OETST HTSBR ENEVE
AOUEP GIEES OBDUO RSFEE RCDYA DUTAE PEADR DIGSE EBFUO GGOPO
GALYF EWSOE EMDNT OHREB HAAES NEWOR GNFIA ULNLW ADUEO DCOTR
ARGVU ENEWH IERTL AUILM SONIO TMUIN EWAIU EWLOE RSTTT ISDRS
ASNUS SIESM ERDHE TRYRH PNLRT EREAD MREDE BNNTR NENWM OUTRD
```

```
OSANE OWOMC GIDCI ASAON TIIOI ASCES ISSUP CRMOY BRINE YWEEL
AYLEW TYRTI LHSTO
```

5. Break this ciphertext.

```
MICYRLHHTAHTLZHAENOHIEALEOBFGALETETOSAZGTAFHRAGATEFLVNYATHT
ETARFRNDINIEDDNGSNCGPOTOEEPIGTFMLEWNIEOERGGTEESERILFTNBNEIK
IASTSRVPZGPEOINIALPLGNOINWTTSWSMRAYIRSSZTNHDAOEEIENRNWTTSOD
OAOSLCSRCTEFOESLYALIATRENISDHNKCFGOHUORITLPEUHREAINEELTGEDO
RT
```

Unit 98
Redefence cipher

The *redefence cipher* is a modification of the railfence cipher in which rows are read off in a different order. The key for the redefence is the inverse permutation of the rows (we use the inverse so that it lists the order in which they are read) and the offset with which the plaintext is written in.

Here is the example from the previous unit. We will take the key to be (1 2 0 3) and 5. The plaintext is written down with the offset:

```
0    •-----H-----S-----S-----P-----T-----N-----T-----P---
1    -•---T-I---S-A---A-E---Y-T---I-H---A-S---I-I---I-H--
2    --•-•---S-E---G-W---N-R---E-W---A-R---P-S---O-C---E-
3    ---•-----M-----E-----C-----D-----T-----O-----N-----R
```

The rows are read off in the order 1, 2, 0, 3 to get the ciphertext.

```
TISAAEYTIHASIIIHSEGWNREWARPSOCEHSSPTNTPMECDTONR
```

Reading and references

American Cryptogram Association, www.cryptogram.org/downloads/aca.info/ciphers/Redefence.pdf

Programming tasks

1. Implement the cipher. The key is a permutation and an offset.

2. Implement a brute-force attack on the redefence cipher.

Exercises

1. Encipher this text with the redefence cipher. Use offset 2 and permutation (3 2 0 1 4).

   > I have committed sins, of course; but I have not committed enough of them to entitle me to the punishment of reduction to the bread and water of ordinary literature during six years when I might have been living on the fat diet spread for the righteous in Professor Dowden's Life of Shelley, if I had been justly dealt with.
   >
   > (from *In Defense of Harriet Shelley* by Mark Twain [Samuel Clemens])

2. Decipher this text with the redefence cipher. Use offset 3 and permutation (2 1 0 3).

```
HJCTIOIILRTRTUPAESFAFAIAYNPTOWODRTININWHASLCRLNNECSLYAEGRYN
PETPUEHRWIUTVTYAPNNHEIFHOFITDYEBTFSOSOUTEUTRRPDEOWRMLTSAONS
OWNCRNOTOHCYAYCEGDASCSUIVDYMNDOSNITOTUNCMIEALLUUOEVNOANADAT
OOBTSOEENROITIHUECIIEONCUFNBATECFIHNBLSEGTRNTEHKLAUNRSRMRDT
HEADSMIUABELDEARAREHSIFOTNTTSA
```

3. Break this ciphertext.

```
AITEENSTIMUTOOSEEEIYDEAATMAATWTNTTSDEIRAOESPSOIIFATRICUINHD
SFATRAIFRRWITDAVLHYENLREANARGYLTHGEOPENDHYINUOQAICIOINFHMSN
TANLMFCOEACOLDEMPPLESWRHTUANPUEOOHCSYSNHTILTNJCISDUOSRGNIAD
OISUTHLVLYETSIEPINREEVOHIIIDHRFLFLYASOAFGDMIAOALAENOUIKNSOT
ELRTOOCLTTSEARSNRCINTSRIBNIELTONCILTSTRWHOISOTTWBSTETSANTEB
LHINNSPTDNAELEACUHNNVPTAOTSETEDOWOTA
```

Unit 99
AMSCO cipher

The *AMSCO cipher* was invented by A. M. Scott. How it got its name is still a mystery.

To encipher a text with the AMSCO cipher, we are going to write the text into a matrix. We must first decide whether the first box should contain one or two letters. After that, we fill in by rows in such a way that when looking across a row or down a column, entries in the matrix alternate between holding one and two letters (the box containing the last letter need not be padded to hold two letters). The message is not padded, so some columns may remain incomplete. For example,

```
TH   I   SM   E   SS   A
 G  EW   A   SE   N   CR
YP   T   ED   W  IT   H
 A  TR   A   NS   P   OS
IT   I   ON   C  IP   H
 E   R
```

The key is the order in which the columns are read off to form the ciphertext. If our key is (2 5 0 1 4 3), then the ciphertext for our example is

```
SMAEDAON ACRHOSH THGYPAITE IEWTTRIR SSNITPIP ESEWNSC
```

Notice that the key is the inverse of the permutation that acts on the columns. The key can also be represented by a keyword.

Reading and references

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; page 51.

American Cryptogram Association, www.cryptogram.org/downloads/aca.info/ciphers/Amsco.pdf

Programming tasks

1. Implement the AMSCO cipher. Allow for the choice of using one or two letters in the first place. Be careful how you handle incomplete columns in decryption.

2. Implement a brute-force attack on the AMSCO cipher.

Exercises

1. Encipher this text with keyword `YACHT` and two letters in the first box.

   > This unlikely story begins on a sea that was a blue dream, as colorful as blue-silk stockings, and beneath a sky as blue as the irises of children's eyes. From the western half of the sky the sun was shying little golden disks at the sea—if you gazed intently enough you could see them skip from wave tip to wave tip until they joined a broad collar of golden coin that was collecting half a mile out and would eventually be a dazzling sunset.
   >
   > (from *Flappers and Philosophers* by F. Scott Fitzgerald)

2. Decipher this ciphertext with keyword `PARADISE` (drop repeated letters) and one letter in the first box.

```
AMRAMEEOSSTPRORUNEELRYREDERRCANIHIGDAEEDETHIMOSORLDTMEOGAVI
ETDNGNIVEYRVGMUINGREASFYOOSANTLELAWCHSINOYNNEWEFYOGOFE
```

3. Break this ciphertext. What might the keyword be?

```
EHEBWIASHITLONWWEDNEROEYEESENDETERIMPEETHSSAETSTEGNTTARYKZI
HECSTHHSPIESUHEHHALTHDETESNAPPRCEEASMGAORORSTTNDNDRTRAFEPRY
BPASNMAINSYPIHHOUERORNEFITAKUCOAESSERFITEFSCGHBREIMERICADSH
HIGIPOGAHPBENSTIBSUCOLEINDAIPSJEPAGRFTFINRSTAGTIIMIWITTOROK
GSNRTATOEILHPAEUTEDURARARINCHTELCFO
```

4. This ciphertext is from the 2015 British National Cipher Challenge. Break it. What is the keyword?

```
UOCAD EDMMA EDAEC RRSOI COIPI IFEBO BDUSI SDSBE ENTTY ANETA
DCOUD KOOTT RBELE DRAOA USYYO HISNL EDDSR ISHRS HEEOS ETROS
TLSEU NRSCC OUWRI MOLHN STERS EDEEA RINTE WPENN OMSFT AITOF
TOILL CAPEC ESEPN DUETN DEEUS ESOMF AKOGT INLGE LYARU ITSIC
RIOIR SECOR ETNEU EATMT ALIHL EICRV EASME NCIED ATHEO LKWHE
OHNWO NFIMH THUBE AIBND ATTHU EKTDO OECTW ITABA OULCA QWIUP
IYPOC ODEEO ONSHI TCIEE ATAIA CLTTA RCHGH OFHET DPAEC HMAHV
EHLOL YARLE IKEDR ODTEI DYSKE EYCFS HFOFE HTGEJ NDBBR YPHON
IOSRO LMREF RWFII GHYPH NMEHR ELASO NDEIV TDTIR HEROT URECH
ORHEG YNDED MEPYT WVIID OMOUT YAION AUFUC ATWOU OKTEF LTHMU
TGNEC TITAR AOWEN NGSST EFRRE SCLFS PSATO EHIHE LASTV CHIFI
IUTET EYEEC STNHE OILDS TNEPM ROSTT SRAFL LRHED ISESS RETIE
NASU
```

Unit 100
Myszkowsky cipher

The *Myszkowsky cipher* is a modification of the columnar transposition. It requires a keyword that has repeated letters; otherwise, it degenerates to a regular columnar transposition. The plaintext is written into a grid with the same number of columns as letter in the keyword. The columns are labeled by numbers representing the alphabetical order of the letters of the keyword, where identical letters have identical numbers. To read off the ciphertext, we start with the first row and read all letters in a column labeled 0. Then all letters in the second row in a column labeled 0. When we have finished all rows, we start again at the top with all columns labeled 1. Then 2, etc.

An example couldn't hurt. Let's encipher this short message with the keyword `TATTOO`.

`THIS MESSAGE WAS ENCRYPTED WITH A TRANSPOSITION CIPHER`

The keyword `TATTOO` is converted to numbers 2, 0, 2, 2, 1, 1, because 'A' is alphabetically first (zeroeth), 'O' is next, and 'T' is last. We write the plaintext into a grid with those column headings. There is no padding.

```
2  0  2  2  1  1
-------------------
T  H  I  S  M  E
S  S  A  G  E  W
A  S  E  N  C  R
Y  P  T  E  D  W
I  T  H  A  T  R
A  N  S  P  O  S
I  T  I  O  N  C
I  P  H  E  R
```

We read off all of the letters in 0 columns: `HSSPTNTP`. Then all 1 columns: `ME EW CR DW TR OS NC R`. Pay close attention to the order of the letters. We take all 1 columns by rows. Now, all 2 columns: `TIS SAG AEN YTE IHA ASP IIO IHE`. The cipher text is

`HSSPTNTPMEEWCRDWTROSNCRTISSAGAENYTEIHAASPIIOIHE`

Reading and references

Émile Victor Théodore Myszkowski, *Cryptographie Indéchiffrable basée sur de nouvelles combinaisons rationnelles*, Paris: Société Française d'Imprimerie et de Librairie, 1902, gallica.bnf.fr/ark:/12148/bpt6k1265620p

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; pages 50-51.

American Cryptogram Association, www.cryptogram.org/downloads/aca.info/ciphers/Myszkowski.pdf

Wikipedia, en.wikipedia.org/wiki/Transposition_cipher#Myszkowski_transposition

Merle E. Ohaver, "Solving Cipher Secrets," *Flynn's*, September 17 and 24, 1927,
toebes.com/Flynns/Flynns-19270917.htm,
toebes.com/Flynns/pdf/Flynns-19270917.pdf,
toebes.com/Flynns/Flynns-19270924.htm,
toebes.com/Flynns/pdf/Flynns-19270924.pdf

Programming tasks

1. Implement the cipher. During decryption, be careful about incompletely filled columns.

2. Implement a dictionary attack for a ciphertext that was encrypted with the Myszkowsky cipher.

Exercises

1. Encipher this plaintext with the keyword `ABRACADABRA`.

   > Here began the experiences that quickly ripened Houdini into the World's Handcuff King and Prison Breaker, which he is, has been and always will be. In exploring his wits for exploits to amuse and entertain the audiences, Houdini hit upon the feat of escaping from ropes tied round him in every conceivable way.
   >
   > (from *The Adventurous Life of a Versatile Artist: Houdini* by Harry Houdini)

2. Decipher this ciphertext with the keyword `DOLITTLE`.

```
OAYOGHLIEOHALLMHAOKLNNGRTEHHDDWTODTSDDOEMAENSTRWONOOTEHOOWO
COIAEAHUAARRTCDTEATNSEDTJDTMMSTARRTNEHLNTYWREILRCISEIDAPCNE
UPEMRSNODFWELEENASRAAMLIHNLEANEWPERAAWT
```

3. Perform a dictionary attack on this ciphertext. The keyword is an English word between five and ten letters in length.

```
WHUHWHTIBEPSIIIYOAHNDNTLHMTBRESOHATENESLYLAHNTNLRSAFBATOTAT
EIGIUESLOTEROLWTATWLDAYSEGTOGMWHGNMICDWBIVIONEOFAWCCGIERAHK
GTKBAYMESTRLROTOMIOHTAOBENOEEEUWBYFTBIOFRTTOAEERDLVRAEEKNTS
KTEOAOEDNCUALHEITUNBEHOFYFERERHTINEHLYBDTHSSMSTOELIEHEENGFE
IEHAIRNSBENMEAIEEIDYLVOATIWAREDHCLCNTSANITORLTNY
```

Unit 101
Cadenus cipher

The *Cadenus cipher* involves both a columnar transposition and a rotation of the columns. The key is expressed as a keyword in which repeated letters are dropped. The plaintext must be a multiple of 25 times the length of the keyword after repeated letters are removed. These are the steps in encipherment:

1. Divide the plaintext into blocks that are 25 times the length of the keyword (after dropping repeated letters in the key).
2. For each block:
   a. Write the block into a matrix as though for a columnar transposition. Each column has a corresponding letter from the key. Each column has 25 letters.
   b. For each column, roll it downwards by an amount determined by its letter in the key. Use `A` = 0, `B` = 1, ..., `V` = `W` = 21, ..., `Z` = 24 (`V` and `W` take the same value so that there are only 25 values).
   c. Apply a columnar transposition using the keyword, without repeated letters.
   d. Read off this block's part of the ciphertext by rows.

An example seems necessary at this point. We begin with a plaintext that has 375 letters and the keyword `ORATIO` (because Shakespeare did not pronounce his `H`'s).

```
TO BE OR NOT TO BE THAT IS THE QUESTION WHETHER TIS NOBLER IN
THE MIND TO SUFFER THE SLINGS AND ARROWS OF OUTRAGEOUS FORTUNE
OR TO TAKE ARMS AGAINST A SEA OF TROUBLES AND BY OPPOSING END
THEM TO DIE TO SLEEP NO MORE AND BY A SLEEP TO SAY WE END THE
HEARTACHE AND THE THOUSAND NATURAL SHOCKS THAT FLESH IS HEIR TO
TIS A CONSUMMATION DEVOUTLY TO BE WISHD TO DIE TO SLEEP TO SLEEP
PERCHANCE TO DREAM AY THERES THE RUB FOR IN THAT SLEEP OF DEATH
WHAT DREAMS MAY COME WUTEVUH
```

After removing repeated letters, our key is `ORATI`, which has five letters. So we break the plaintext into three blocks of 25 × 5 = 125 letters. We lay the first block into a matrix of five columns under the key (shown on the left below). The numbers of steps for rolling each column are `O` = 14, `R` = 17, `A` = 0, `T` = 19, `I` = 8. The columns are rolled downwards (shown in the middle below). Then a columnar transposition is performed (shown on the right).

O	R	A	T	I
T	O	B	E	O
R	N	O	T	T
O	B	E	T	H
A	T	I	S	T
H	E	Q	U	E
S	T	I	O	N
W	H	E	T	H
E	R	T	I	S
N	O	B	L	E
R	I	N	T	H
E	M	I	N	D
T	O	S	U	F
F	E	R	T	H
E	S	L	I	N
G	S	A	N	D
A	R	R	O	W
S	O	F	O	U
T	R	A	G	E
O	U	S	F	O
R	T	U	N	E
O	R	T	O	T
A	K	E	A	R
M	S	A	G	A
I	N	S	T	A
S	E	A	O	F

O	R	A	T	I
T	O	B	T	E
F	I	O	I	O
E	M	E	L	E
G	O	I	T	T
A	E	Q	N	R
S	S	I	U	A
T	S	E	T	A
O	R	T	I	F
R	O	B	N	O
O	R	N	O	T
A	U	I	O	H
M	T	S	G	T
I	R	R	F	E
S	K	L	N	N
T	S	A	O	H
R	N	R	A	S
O	E	F	G	E
A	O	A	T	H
H	N	S	O	D
S	B	U	E	F
W	T	T	T	H
E	E	E	T	N
N	T	A	S	D
R	H	S	U	W
E	R	A	O	U

A	I	O	R	T
B	E	T	O	T
O	O	F	I	I
E	E	E	M	L
I	T	G	O	T
Q	R	A	E	N
I	A	S	S	U
E	A	T	S	T
T	F	O	R	I
B	O	R	O	N
N	T	O	R	O
I	H	A	U	O
S	T	M	T	G
R	E	I	R	F
L	N	S	K	N
A	H	T	S	O
R	S	R	N	A
F	E	O	E	G
A	H	A	O	T
S	D	H	N	O
U	F	S	B	E
T	H	W	T	T
E	N	E	E	T
A	D	N	T	S
S	W	R	H	U
A	U	E	R	O

We read off the ciphertext by rows. The remaining blocks are processed in the same way, and in the end we get this ciphertext:

```
BETOTOOFIIEEEMLITGOTQRAENIASSUEATSTTFORIBORONNTOROIHAUOSTMT
GREIRFLNSKNAHTSORSRNAFEOEGAHAOTSDHNOUFSBETHWTTENEETADNTSSWR
HUAUEROOSEOTSAAAEYLNAOSKHPDNTAYLMHNDOEITEEEIACHMBTDRNNSHESP
SNHTNFUUWTIHNTORTAAOTLCHPLSATRDTSOBPREDEGETRSHBUOEDOAHESEOE
TNEIHAEIDOEYLTCSAEDMOOLSOTMETOTENETMHDCICFAEOUTREOHLETPNFOS
EAHHURDDEICTSDTRBOWPYHTRAEISMARLSMAROTEYAPEAWEHLUUETEIOHADV
MEHEYNATEWUEBSTOVNPTS
```

Reading and references

American Cryptogram Association, www.cryptogram.org/downloads/aca.info/ciphers/Cadenus.pdf

Lanaki, lessons and tutorials, lesson 16, www.cryptogram.org/downloads/LANAKI%20Crypto%20Lessons/LESSON16.pdf

Programming tasks

1. Implement the cipher.

2. Implement a dictionary attack.

Exercises

1. Encipher this text with the keyword `INTERNET`.

   > While my internet is down, I have to write my own plaintexts. It is very frustrating, but this is the cost we bear when we steal internet from the store down the street. I'm sure you don't want to hear about that. Perhaps you would like some historical bits of information about the Cadenus cipher? Unfortunately, I don't know any. Let me look some up online...

2. Decipher this text with the keyword `HEART`.

   ```
   DARHNAIIRTNRAEIWEDTLOAWANJSERHHAHTNIACUIRPUHWIYVTBITSEESWAI
   EVOBODMGACNIRSHRPEFIESRSXHFOFESHHDNXDSEENCIHADNAGESTSAOSWEL
   AMSCDFN
   ```

3. Break this ciphertext with a dictionary attack.

   ```
   EWOIETEHTDOEINPEHAEAAERRRDNWIOTDHYHLEDONRPWTEPWSRFMNAEHHHEH
   NLQAORRGECAXEUHLHREHRSATNRASEESRWSLITAMITAPIIEPSISAHWPWEWJO
   PEIDWUEDDSNATSSTTANTOARATEAAGLTISAIRNGENBUUINFSEEFOXIODWATG
   NOEAFESEHDEIITIRDYSGSOGRENCBNIRUDLCSNTOAGBATTOITALNHNIGIIPL
   VDESHEHTTVIIHEUDTNTNPTAINSEPTPWEOSHETLWESLEEWAMHWBSLIRNCRRU
   UTTCSGRHEEYTLLMROMERENCOTHATEGNFHWAAIDGINUFYNRSNSAMORAUGSWB
   NTIFEREDILLTILEOEAERSHEYNTDDNEVIRRNSBXSTXWHERDGVITISNAMEPSO
   SHBHWWBWDTERTEEATAEDTSSLSTTRIATESUOEOSELOEENSNWEANEOAATASHA
   IRPPANWEOSIEHONLCHNNTUDTISEE
   ```

Unit 102
Hill-climbing attack on the Cadenus cipher

In this attack we unlink the rolling and columnar transposition and vary the keys of the two operations separately; at the end, we put them back together. Therefore, we need a decryptor that applies them separately. Before we apply the attack, we have to determine the key length (or guess it). In the exercises in Unit 11 we found a cutoff in the tetragram fitness above which we are confident that we have English text; we will use that cutoff in the algorithm. We will be using the usual technique of working with parent and child keys. To avoid local maxima, we also need to use a margin of error. For this cipher, we will vary this margin so that it vanishes when we reach the fitness cutoff.

0. set the alphabet to be `ABCDERGHIJKLMNOPQRSTUVXYZ` (no `W`)
1. initialize the parent shift key to an array of 0s
2. initialize the parent permutation key to (0 1 2 ...)
3. calculate the best fitness as the fitness of the unmodified ciphertext
4. set a counter to 0
5. while counter is less than 1,000
    a. copy the parent shift key into a child shift key
    b. copy the parent permutation to a child permutation
    c. randomly choose one of these ways to modify the child keys:
        i. change one member of the child shift key to a random number in 0, ..., 24
        ii. swap two randomly selected members of the child permutation
        iii.
            - randomly choose a number *n* from 1 to the key length
            - roll both keys leftward *n* of steps (with rollover)
            - subtract 1 from each of the last *n* members of the child shift key
        iv.
            - randomly choose a number *n* from 1 to 24
            - add *n* to each member of the child shift key, modulo 25
    d. decipher the ciphertext with the child keys
    e. calculate the new fitness of the new plaintext
    f. set the margin to be
        i. 0 if the new fitness exceeds the cutoff
        ii. the square root of the cutoff minus the new fitness, all divided by 10, if the new fitness is less than the cutoff
    g. if (the new fitness exceeds the best fitness) or
       ((the new fitness exceeds the best fitness minus the margin) and
       (we roll a 1 on a 20-sided die))
        i. copy the child shift key into the parent shift key

        ii. copy the child permutation into the parent permutation
        iii. set the best fitness equal to the new fitness
        iv. set the counter to 0
    h. increment the counter
6. convert the parent shift key to a keyword; for each member of the shift key:
    a. take *n* to be 25 minus the shift
    b. the letter of the keyword is the nth letter of the alphabet (without `W`)
7. output the keyword

Programming tasks

1. Write a function that deciphers a ciphertext with separate keys for the shifts and for the permutation.

2. Implement the attack. Use your function from Exercise 1, and use tetragram fitness.

Exercises

1. Use your attack to break the example from the previous unit.

2. Break this ciphertext from the 2014 British National Cipher Challenge. It may take several attempts.

```
AFCAEUOTTACTHRIOLETCSERTHSHTRAHKYORPFRGEOADPPJNGLTERNEFEOFI
ORTSDDOEEUMSCRUERNFETLAAFSTWIENTRVOONERHUAHRAVEREETSVSIELHL
OSTDOALOYAESMNNDIGNNRHOHHTSNAOILNCNSSICREANNEEIIIERWTANESRV
OGIEIYWSSDGPVOIAISAOAEOAEDRNITRNXEIGRPSSHADHDTOIPAATEXENNES
AGROBTLESRNROIRYPBGEDCLLIWALALEENIGRRNWYRLIMLPSTOLEFTRDMUAR
IEEEIIAOLNEWSAOHRTLSTOBETNSLVFIVDOVTPOAEEISCIOHIPSEVEEDTEWF
ARNHEBLEAOTOHTTTEPNCKAONHWETMVYPRREONNASGDEDOEEEOAAMTCICTTI
FNADRESRTSEROSETRHCICTPSAAEHLDHSFXSOAOTCTBBSOEIRNSADLYTRRUN
RCEPTTHREUHNKTACECEELRWNIREEEAESEEEIDISOGCEOMNRTEJHAGABSENI
TLWTRNBMIELSARETESRNGSNHEBIOSDIENAFLEISAHOCIFEVMFATANATRNIA
GNHATNMIBNIUFENRTOTTRNYPAIDYIEGDNMERHHIOTRETCESSEILDRBCEPRI
GAESOADLTAHIEVEBRCENLEVASADNNTHNEITEIISAHUHHUAMONEFYHLONWHA
EEEEOSNEEYANEISETOGYITERLIHTCMIOIRARFDOETNIHTNEHIIKAMRDMNAD
ANAODSESEIYCLSIANTAOLTCIYMIDENTTHLTNDXTTTMASBLEAEETLISIRTWT
URPFAILTEAOEFEISIIIYISIKVTWISPRBSINELPHRMOHIAGNLSLVITODAISD
PNYDDCAAOTAHCEHTUEIRREDAECTOSNRHVNAODOIKOETCINENEURRISDCOUR
AGLVIMMUPPDITEANDITMAAIAIELEONNREEDAODBOIUMELROTNTTTGITNRLR
IENNIKLYSOGSTCIFYPIPVIDVSSMNCEIASIITSNNEATITOMRHBNHNIDPRLRE
POYNALSNVSDOSANESITFAENLTGODATTEEAISICROOTMSMFHAUENIRSGHYNW
EINTEGODIILEEDTARNOSRCAAENDTCUTTFDRBEHTMFITOORDRUIAOYAANOEE
LDOINHUSGITEAORIECEVEMNTRATMTFPEUCUTAHAMTNEWONICDEEMRPAOLIT
OAFESOOSSPFNLNEEOOTACHLLIRSSXSOFPDFTFRNPRAEEAYLONAHAUTNTCNT
```

CBAWLONEFTOATECVOWDLWVNNEEDTIIOIGTEGMTAHEEATEFAAEPRRCROSHEE
RRPALEDIENGIDRREOUHVESUROYTNSOSINUIUIOFPRDA

Part V
Grid-based ciphers

Unit 103
Polybius square

If we cram the alphabet into a 5×5 matrix, we have a *Polybius square*. Unfortunately, the English alphabet has more than 5^2 letters, so we may have to jettison one of them. Usually, we will put `I` and `J` in the same spot, but the choice is arbitrary. Now, if we add labels to the rows and columns, we can use those labels to represent the letters, by using the row label followed by the column label. The choice of labels is also arbitrary, but the usual custom is to use the digits 1, 2, 3, 4, 5.

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

An example, of course, is mandatory. If we take this message,

THIS	MESSAGE	HAS	BEEN	POLYBICIZED
44232443	32154343112215	231143	12151533	3534315412241324551514

then its transformed version is

442324433215434311221523114312151533353431541224132455151 4

What we have done is not strictly a cipher, but rather a *code*, since we have merely changed the representation of each letter (*encoding*). We will explore that topic more thoroughly in Part VIII.

If we want to communicate with the neighbor in the next apartment, for example, we could replace the digits with sequences of taps on the wall. The example above becomes

....
. ...
.. ..

Reading and references

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, page 83.

Fred B. Wrixon, *Codes, Ciphers & Other Cryptic & Clandestine Communication*, New York: Black Dog & Leventhal, 1998, pages 190-191.

Wikipedia, “Tap code,” https://en.wikipedia.org/wiki/Tap_code

Programming tasks

1. Implement the Polybius square. You may choose to do so as a Python class. You will need routines that convert from letters to row and column labels and back again. Allow for the choice of labels to be arbitrary. If you anticipate what is coming in the next unit, you will also allow for the packing of letters into the square to be done arbitrarily.

2. Implement the tap code, i.e., create functions that convert text to dots and spaces and back again.

Exercises

1. Encode this text. Use labels 1, 2, 3, 4, 5.

   > Conversation died in the speeding car, partly because of the concentration required by the troopers, secondly because all transmissions whether intercom or radio, on a code two or three run, were taped and monitored by Control. In the center of the instrument panel, an oversized radiodometer was clicking off the mileage marks as the car passed each milestone. The milestone posts beamed a coded signal across all five lanes and as each vehicle passed the marker, the radiodometer clicked up another number.
   >
   > (from *Code Three* by Rick Raphael)

2. Decode this text.

```
443435451231244323111334141534212334333442443422345115423 32
433131143154334212433142451241445113113343212114432242223 44
431515324434242325315444231144442315354512312443231542521 14
311331114513413114415342114451531312433221133145224432315 14
443424334442341445131524441143442315354234351542323414153 42
114151324142433221131313515424334331131142421212413453144 24
154311331432244345331415424344113314243322434345132324323 53
124131144243433523445311414343215224215114424332445434424 13
151245442421442315414515434424343312151424421513443154354 54
444343215522315442315424423154215114215333444131143154352 23
154215144515314311421542242223441133143542343515422452344 53
114453323154324441144243322315411334352154244231542151142 15
```

3. The labels for this Polybius square are not the usual ones. Can you still decode the text?

```
ELEWLOBBLLEELLEWBBOOBEEWBLEEBBEEOOOOLBEEOOOOLOBLEWBOLEEBBOO
WOLOBEWLOBBEWWEEWOLBLEWBBOOLOBOBBEWEOOOLBEWLOBBEBLOEOEWBBOO
OOBOEEBBWOBOBBEWELOWOOOOLBLOOBBLBOOLBBLOBBLLLOWLEEOBLWLOEBL
WEWEOBOOWOOBOLEEEOBEEBEBOBOBBBLWOEBBOEOEWBOLELBBOBBBOOWOLLO
OBEEBEBEBOOWOOEEOOOOLBEWEWBEELBOWLOBEEBBEOLOEOOBEWBEBEBLWOB
LBOOOLBEWOLOBBWOLEEWOEWOLELBOBOLWLEBOOLOOLBOLEWEWLELOBBLLEW
OLOBBOLEEELLOWEEOLEOLOEWBBOOEWLOBLBBBOOOBWOWOOOOLOBBLLBOBBO
OLBEWOBEBOLEWWLOBLBEEOLEOLOOOBOOWLLLBOOOOBOELEWWLBOOLOOLBEE
OOLBBOOWOBEEBBEOOOBOWOBOOWLEBOOLBLEWOOBOLBEEWEEWOBBEEWBWOOB
OBBOOLBEEOOEBBOOOOOLBOLBOOWLLLBOOLBEWWLLBBOBEEWELOWOBLOBBEW
OBOBWLLOOOLBBOOWOOWLEELWLOBBLLOWBWEEBBEOOBEWEWLOBBLLEEBBWOO
OLBLOBBLL
```

4. What is my neighbor trying to say?

```
.. ... . ..... .... .... ... .... ... .... .. ..... . . . .... . ..... . ..... ... ..... . ..
.... .. . ..... . . .... .... .. ... . . ... ... . .... .... .... .. ... . ..... ... ... ....
.. . . .. .... .... ... . ..... . .... . . .. ... . . ... ... . .... . . ... ... . .... ..
..... ... ... ... .... . ... .. ..... . ..... . .... ... .... ... ... .... .... .. ... . .....
. .... ... .... ... .... .... .. . . ... . .. .... .. .. .. ... .... .... . . ... . ... ..
... .... .... ... .... .... . .... .. .... .. . .. . .. .... . .... . ..... ... ... .... ....
.. ..... ... ... ... .... . ... .. ..... ... ... ... .... .... .... . . ... ..... . ..... ....
.. . ..... ... .. ... ..... .... .... ... .... .... .. ..... .... ... .... ... ... . ..... ..
... . ..... ... .. . . . .... . ..... .. ... .. .... .... ... ..... . ... .... .. .... . ... .
..... . . .... ... .... ... ... .... .. . .... .... . . .... ... .. ... .. .... .... ... ..
..... ... ... ... .... . ... .. ..... .. ... . . . .... . .. . ..... . ..... ... ... . . ...
... . .... . . .... ... .... ..... ... ... .. . .... .. .. .... .. .. .. ... .... .... . .....
... ... .. .... ... ... .. .. .. .... .... ..... .... ... .... .... ... . ... .... .... ..... .
.... . ..... ... ... ... .... .... ..... .. .. .. ... .... .... ... .... . ... . . .... .. ....
.. ..... .... .... .... ... .... .. ... . ..... .... .. ..... .. . ..... .... ... .... .... .
..... .... .. ... ... .... ..... ... ... .. .... ... .... ... ... ... .. . . . . ... .. . ...
... .... ... . ... . . ..... . ... .... .... .... .... . ..... ... . . ..... .. .. .... .. . .
... .. .. . .... .. ... .... ... .. ... ..... .. .... .... .... .... .... .... ... . .. ....
..... .... .. .. .. .. ...
```

Unit 104
Polybius cipher

Suppose we pack a Polybius square with a mixed alphabet. The result is the *Polybius cipher*. There are many ways to mix an alphabet with a keyword, as we saw in Unit 34. With this new cipher, we add a new dimension: There are many ways to lay the mixed alphabet into the square. Here a just a few:

- by rows

- by columns

- boustrophedon (by rows or columns in alternating directions)

- spiral, outside-in or inside-out, clockwise or counterclockwise

At this point, you should realize that no matter how we mix the alphabet and no matter how we place the mixed alphabet into the grid, all we have done is replace each letter with a two-character string (a code word). By listing all two-character strings in the ciphertext and assigning a letter to each,

we convert the Polybius cipher to a monoalphabetic substitution cipher, and in Unit 36 we saw a method for solving monoalphabetic substitutions automatically.

Another way to think of the cryptanalysis of the Polybius cipher (P) is to factor it into a monoalphabetic substitution (S) followed by a (keyless) Polybius square (Q):

$$P(k)\, p \;=\; Q\, S^{-1}(k)\, p$$

Here, p and c represent the plaintext and ciphertext, and the substitution is inverted because to go from the mixed alphabet to the keyless square, we are *un*mixing the alphabet. Decryption is

$$P^{-1}(k)\, c \;=\; S(k)\, Q^{-1}\, c$$

So now it becomes obvious that to break a ciphertext, we undo the Polybius-square encoding as if the square contains the straight (unmixed) alphabet, and then we break the substitution. Remember that we have to invert the substitution key to reveal the mixed alphabet in the square and the keyword.

Finally, we might want to think about other grid configurations. For example, to use all twenty-six letters we might take a 13×2 grid. We could recognize that a nonstandard grid was used if the number of labels is not five (or ten if rows and columns have different labels).

Reading and references

Practical Cryptography, “Polybius Square Cipher,” http://practicalcryptography.com/ciphers/polybius-square-cipher

Programming tasks

1. Generalize your function from the previous unit that fills a Polybius square to allow for many options on how to generate the mixed alphabet and on how to fill the grid.

2. Implement the Polybius cipher, given a keyword, an alphabet-mixing method, and a grid-filling method.

3. Implement the attack mentioned above.

Exercises

1. Decipher this ciphertext with the keyword `POLYBIUS`. The mixed alphabet is constructed by adding letters that come alphabetically after the last letter of the keyword (in this case, start with `T`). The grid is filled by columns. Labels are as in the example above.

```
2410023143323102322124312110433244030232210131001020304001 1
1214201004313442132441313322130311042443242043220203010112 4
1313030121324201004313443113123131433231012120102433132410 1
1334323104430101131104044133204314313141024103220004332401 3
3101422111402131013111310110244201004313 44
```

2. Break this ciphertext from the 2019 British National Cipher Challenge:

```
FBGAI AGCFE KEFEK CIAGC FCGAF CIBHD HEFCF AFBFA GDFCH DFEKC
IAKCI BGBGC IAHAF EKCFA KAIAG CFBFA GBFBI AFBHE IAGCK CIAFC
IBHDF EGAGA FCHDI AHEIA FCKDF CFAIA KCFBF AIAGC FEHEF CICFB
FEIAH EFDKA HBHDF CIEKA IDKCH DHEFB HEKEF CFCHA FEKEK CHEHA
KAGEF CKCIA GCFBF AGBFC GAIAG CFEHE FCICF BFEIA FEHAH BFBHD
FEIDK CHEHE IAFCG DFEKE FDKAG CFBHE KEFEK CIAGC IAGCF EIBHE
HEHDG AFEFE KEHEF CFAIA GCFEK CFAGB FEHDI AGCKC IAIDF EHEIA
FBHDI BHBID FBIAG CFEKC KDGCI DKCHD IDFEG AFBGB GCIAF BFAFB
IAHEH BIBHB HBFEI AHEIA KCIAF EHEID FEGCK CICFE IAHDF BFEKE
IAFCK DFCFA IAKCF BFAFB IAFDI BIAFB FAFCI DHDFE KCGEF BHEFE
IAGCK CIAID FEKDK CFAFC FAGEK AKEFE GAFEK CIAFB IAIAG CFEID
KCKAI DFEGA FCIBG BGCIA GAKCH EKDFB HEHAF DKAIA KCGDF BFAGB
FCFAF BIAHE HAFBG BGCIA FBFAF CHBFE FAIDK CHDFC IBHDH BFCGE
FBIAF BKDFB KCFAH EGCKC ICFEH EGCHD IBFAG DGAHD FCHAI AGCFE
KDFCF AGAGE FBKDI AKCFA KEIAG CFEKA FAFEF EKEKC HDFEK CHEFC
FAIAF CKDFC FAGAH DFCFA IAIAG CFEFE ICFBG E
```

Challenge

❄♥☮✤☮❄★♥♥❄✖♥✤▲✤♥●❄☮☺✖♥❄♥✤●❄☮✤♥★♥☮❄●☺♥❄✖☮❄♥✤●
☘☺☺☮☮✖❄♥●✤❄☮☘✖❄♥✖●♥✤❄●☺☮♥✖♥❄●✤❄☮♥✤♥★●✤☘☺♥❄●✤
♥❄☘✖●✤☺●♥✤★♥✤●★▲✤♥✤●☺☘☮❄♥★★☘☺●♥❄✤☮❄☮★♥❄♥✖♥▲✤
❄▲♥❄✖♥❄☮❄♥✤☮●✖☮☺☮❄●☺♥❄▲❄❄♥✖♥☮❄♥❄✤☮✖●☮☺✤●☘☺♥❄
✤●✖☘❄♥★♥✤●♥❄☘✤☮☺✤♥♥★☮☺♥✤☘❄✤●▲★♥✤▲❄❄♥✖●✖▲☮☺☺●
●✤▲✤●❄❄☮☘✖♥❄✖●●✖▲✤✖☘❄♥✖●✖●☺☮☺♥★☮☺☮✖♥✤♥♥❄★♥☺♥
✤♥★♥☺☮✤♥✖♥✤●✖♥✖▲☮☺❄♥♥★☺♥☘❄♥❄✖●☺●☮☺❄♥☮✤❄☮♥★❄♥
✖♥▲✤☮✤❄☮♥★❄♥♥✖▲✤✤♥❄●☮☺♥✖❄♥✤●☮❄✤♥★♥☮❄☺●♥❄☮✤❄☮
♥★♥❄✖♥▲✤✤♥❄●☮☺✖♥❄♥●✤☮❄♥✤★♥❄♥☘✖●✤❄☮★♥★☘♥✤♥★●✤
▲★✤♥☮✤❄☮♥★❄♥✖♥▲✤▲❄♥❄♥✖☮❄❄♥✤☮✖●☺☮●☺✤●☘☺☮☺♥✖☮☺
❄♥♥✖☺☮●☺☮❄▲☺●✤☺☮☮☺★♥●❄♥✤☺●☺●❄☮☮✤●✖☺☮☮✤❄☮♥★♥❄
♥✖▲✤✤☮❄☮♥★❄♥✖♥▲✤♥✤❄●☺☮♥✖❄♥✤●❄☮♥✤★♥☺●★▲☘☺☮❄☘✖
☘☺☘✖♥❄★♥✤☮☮☺✖☘●✖❄♥●☺☺●❄☮❄☘❄☮☺☮♥☺✤☮✤▲●✤☺☘☺☮❄☘
♥✤▲✖✖♥▲❄♥❄✖●▲✖☮☺●☺☮☺♥❄✖☘☘☺♥✖☺☮●✤▲✖✖♥★♥☺●❄♥☺●
❄♥❄☘▲✖★♥✖☘✤●❄☮✤♥♥★✤♥☘❄✤●☺☘☮☺☘❄♥✤✖▲✖♥❄●♥✤☺●☺●
☮❄✤☮●✖☮☺❄☮♥★❄●▲✖✤●●☺●☺♥✤☮✖☺☮☮✤☮❄♥★❄♥♥✖✤▲✤☮❄☮
★♥❄♥♥✖✤▲♥✤●❄☺☮✖♥♥❄✤●❄☮♥✤♥★☺●●✤☺☘♥❄✤●❄♥♥✖☮☺❄☘
❄♥✖☮❄☮●✖❄☮❄♥♥✖❄♥♥✖☺☮☘✖❄♥●✖●✖☮☺♥☺❄♥★♥☺♥✤♥✖♥☺▲
♥✤♥✖●☺✤♥☮✖☺☮●✖☮☺●☺☺●❄☘❄♥✖☮☮❄✖●❄☮❄♥✖♥✤♥●❄☮☺♥✖
♥❄✤●❄☮✤♥★♥☺●♥❄✖♥☺☮❄☮✖☮❄●✖●☮❄☘✖❄♥●✤☮❄✤♥♥★♥★♥✤
✖♥♥★♥❄♥★♥☺

Unit 105
Playfair cipher

The *Playfair cipher* was not invented by Lord Playfair, but rather by Charles Wheatstone. At any rate, it is a *digram substitution cipher*, which means that it makes substitutions two letters at a time. It has little tolerance for double letters, so before we can apply the cipher, we have to prepare the plaintext by putting an `X` between all adjacent pairs of identical letters. Since it works on pairs, we also need to add an `X` to the end of the plaintext if it has an odd number of characters. (We do not need to put an `X` between letters if they are in different digrams.)

The main engine of the Playfair cipher is a Polybius square. We fill it with a mixed alphabet that was generated in whatever way we like. Since there are only twenty-five cells in the grid, `J` is merged with `I`. The plaintext is processed two letters at a time, according to these rules:

- If the two letters appear in the same column of the Polybius square, then each is enciphered to the letter below it. If the letter is on the bottom row, then we use the letter at the top of the column.
- If the two letters appear in the same row of the square, then each is enciphered to the letter to its right. If the letter is in the last column, then we use the first letter in the row.
- If neither of the previous two cases hold, then a rectangle is observed in the grid such that the two letters are at two of its corners. They are enciphered to the letters in the other two corners. Each is enciphered to the letter in the other corner in the same row.

(Mathematicians in the audience recognize a torus when they see one.)

Now for an example. Here is our plaintext:

```
THIS MESSAGE WAS ENCRYPTED WITH A GRID CIPHER
```

First, we prepare it with nulls, one between the two `S`'s and one at the end.

```
TH IS ME SX SA GE WA SE NC RY PT ED WI TH AG RI DC IP HE RX
```

Let's use the keyword `POLYBIUS` to mix our alphabet, and let's not do anything fancy about how we lay it into the square:

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

The first two letters of the ciphertext, TH, are enciphered to ZD:

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

The next two, IS, are enciphered to UA:

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

This continues, and the ciphertext is

ZDUAVMAWACHFXSUFRSQBIPFETSZDGQKCHIDIDFQZ

To detect whether we have a ciphertext that has been encrypted with a grid-based digram substitution cipher, we look to see if there are at most 25 different letters and whether the length of the ciphertext is a multiple of two (to disguise the cipher, however, one might change some I's to J's, so be careful). If there are any long repeated sequences of characters, then the starting character of each needs to be an even number of letters apart. Furthermore, if we plot the index of coincidence as a function of period, as we did in Unit 39, we often see a slight increase in the even periods over the odd periods (do not take this to mean that the cipher is periodic, just that this is a differential tool). If we calculate the digram index of coincidence (see Unit 13), we should find a large value to indicate that the ciphertext was enciphered two letters at a time.

There are a couple of variations on the cipher that you should watch out for. One gives I and J each their own cell in the grid, but puts Y and Z into a single cell. Another changes the way that double letters in the plaintext are handled. A null (X) is not inserted between them; rather, a digram containing two copies of the same letter is encrypted to the letter one column to the right and one row below the plaintext letter. The usual rules about crossing the edges of the grid apply, both horizontally and vertically.

Reading and references

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; chapter XXI.

American Cryptogram Association, www.cryptogram.org/downloads/aca.info/ciphers/Playfair.pdf

Wikipedia, en.wikipedia.org/wiki/Playfair_cipher

Practical Cryptography, practicalcryptography.com/ciphers/playfair-cipher

United States Army, Field Manual 34-40-2, chapter 7, "Solution to Polygraphic Substitution Systems," Basic Cryptanalysis, U.S. Department of Army, www.umich.edu/~umich/fm-34-40-2/ch7.pdf

Fred B. Wrixon, *Codes, Ciphers & Other Cryptic & Clandestine Communication*, New York: Black Dog & Leventhal, 1998, pages 217-219.

Fletcher Pratt, *Secret and Urgent: The Story of Codes and Ciphers*, New York: Bobbs-Merrill, 1939, chapter XII, section I.

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, pages 198-202.

Joseph O. Mauborgne, *An Advanced Problem in Cryptography and Its Solution*, Fort Leavenworth (Kansas): Press of the Army Service Schools, 1914, www.marshallfoundation.org/library/digital-archive/advanced-problem-cryptography-solution

W. W. Smith, "Solution of the Playfair Cipher," in part IV of André Langie, *Cryptography*, translated by James C. H. Macbeth, London: Constable & Company, 1922, HDL: 2027/uc1.32106002774104 and 2027/uc2.ark:/13960/t0tq62t29

Programming tasks

1. Write a function that returns a boolean value representing whether it is likely that a given ciphertext has been encrypted with a grid-based digram substitution cipher. Your function for the index of coincidence based on digrams may be useful here.

2. Implement the Playfair cipher.

3. Implement a dictionary attack. Remember to allow for many possibilities for the method of mixing the alphabet from a keyword and for the method of laying the mixed alphabet into the square.

Exercises

1. Encipher this text with the keyword `TRENDING`. Fill the square with whatever method you like.

   The point I advance, if it need confirmation,
   I'll prove by a witness that few will dispute,
   A pink of perfection and truth in the nation
   Where fashion and folly are all of a suit.

   (from *Nothing to Eat* by Horatio Alger and Thomas Chandler Haliburton)

2. Decipher this ciphertext with the keyword `GROCERIES`. The alphabet was mixed by starting at the beginning of the standard alphabet after the keyword. The mixed alphabet was laid in by rows from left to right.

```
MDSOASOGTGKCDRBZEQVSKYMHFVIBDSKYMHCOROCEGODGABUICQMRORAOEAI
HPEVFHPDMQCXCNDPUMRKBBPASZKGQPLABKENPNBVIQCASYQWBGZGUAEKYKB
SHIQBUFSCPVLEQOEGUPBBNEQRFQYQCKSZGDCGUQSSIDCKGOGKRXZEQDKFVS
AUCOCLNMRRCHWCMOBVFPDNBLVXCPEDRMHFVPDFVOVRCEAHRFSRLXCZMGQUQ
BXKGGSOBPUNPMDSHQBUIFNSGDUDUDCOWGSRFYTCYMRDSLTRDBXARZRQGKDQ
ITVPLFVOIASDPQWQRDRXCPEGECRVFEDPLCDSDMCBAIQDQPLCOBNVBOZURBY
XCNURQBXNQWSEKQUTCIQAELTFICZEQSHOGHWGENLTMTCPLEKBAUNAEOW
```

3. Apply a dictionary attack to break this ciphertext.

```
CQAHUBVNTIZNTODRBAFRCEAWRKKRNODRVTMCNZITVTUBKHQCXEPRBNTZFRZ
HMBABNOXEQGBTWRTZPODRCUQZOPKVUFDWONTZDRNKFWFKDEDRBANOPGVBIW
MTAWRKXEPRVTLQPUDKOMKATZMXZIKDVTLUDARPTBISBAQCTPMKCKDBZNBTY
CGNRLXVBNRABHFRLZKRQFKHQFPDOPZNDKMOQRTZPVBLLHBFOTVRADTBZNVP
YCDVTOMBADTBISCLOTEVILTODRBNRAVNMCNZFRRAEARKNKTOELRKCQBVKRI
QMHRQDKQKBLITETKBDBQCPQRKQFKHZNVPZBLFDBQCZNVPITZDABDEVQHASI
AWQPOYBAOPUKKHRLGPZIVQFREHVQQERAKVTZ
```

Unit 106
Hill-climbing attack on the Playfair cipher

We are going to begin with Cowan's attack ("simulated annealing") and make some improvements. In his method, fitness is measured as the sum of tetragram frequencies. A parent key (in the Polybius square) is set, and from it a child key is generated by swapping individual letters, swapping two rows, swapping two columns, or flipping the entire square. If the fitness of the decrypted plaintext from the child key is greater than the parent, then the child becomes the parent. This is a step upwards. However, it is easy with the Playfair cipher to become trapped in a local maximum fitness. To allow the algorithm to escape such a fate, a "temperature" is added by allowing a step to go downwards in fitness if the distance down is smaller than a random variable that drops off exponentially and which depends on temperature. As the temperature is reduced, the jitter in the motion in key space gets smaller and the distance downward that is allowed gets smaller. The temperature is allowed to slowly decrease to zero. If the algorithm has not found the global maximum fitness at that point, then it starts over with a high temperature. The algorithm actually does not know if it has found the global maximum, and requires human interference to stop it.

The first improvement that we make on this algorithm is to use tetragram fitness as described in Unit 11. Our definition of it is an average over all tetragrams in the decrypted text. As an average, it is not dependent on the length of the text. Therefore, it can give the algorithm a clear indication that it has found the maximum and can stop. A threshold above which we are confident that we have English text was found in the exercises of Unit 11. For the Playfair cipher, we remove all X's before evaluating the fitness, since they would have been added between any double letters.

Our second improvement is to use a constant margin of error for downward steps. In our algorithm, a downward step is allowed if the distance downward is less than a fixed amount and if a random variable is within a predetermined interval. That margin is 0.5, and downward steps are allowed only if the random variable lands in 5% of its range. The global maximum will be steep enough that the algorithm cannot walk downward out of it. So we can terminate if we have reached a point from which we cannot step upwards within a large number of tries (around 10,000).

Here is the full algorithm:

1. set the parent key to a Polybius square with an unmixed alphabet
2. set the best fitness to the fitness of the unmodified ciphertext
3. set the counter to 0
4. while the counter is less than 10,000
    a. copy the parent key into a child key
    b. randomly choose one of these modifications to the child key:
        i. swap two randomly selected elements
        ii. swap two randomly selected rows

iii. swap two randomly selected columns
iv. flip the square around the diagonal that runs from upper left to lower right
v. flip the square vertically
vi. flip the square horizontally

c. decipher the ciphertext with the child key to find a new plaintext
d. calculate the new fitness of the new plaintext
e. if (the new fitness exceeds the best fitness) or
((the new fitness exceed the best fitness minus the margin) and
(we roll a 1 on a 20-sided die))
i. copy the new fitness to the best fitness
ii. copy the child key into the parent key
iii. set the counter to 0
f. increment the counter

5. output the parent key

Reading and references

Michael J. Cowan, "Breaking Short Playfair Ciphers with the Simulated Annealing Algorithm," *Cryptologia*, 32:1 (2008) 71-83, DOI: 10.1080/01611190701743658

Programming tasks

1. Implement the attack. If you used a different logarithm base, you will want to experiment with different values for the margin.

Exercises

1. Break this ciphertext. What is the keyword?

```
UDSDAEEPVFHPKNNMPILPBMNGDOOGHPGDVFHIVQRSURBETIREAFHPAVKFREH
RRMFANFPUDMRAAUPIAGPEXFTGRUODWRBNFNDOTGPWQGDMNLQVWEUWHGLDFS
AUNOQPUALZSDZDGUFABEZDRBDFDVVQRSGMBEIZTDFNOPPLPUUVRBAUGTVEH
RARFKDRBEODUDVEUCAWRBPRDSNEBXRSLTPWQGRAFAKGLPUWHGLZBFREIERE
ZLRETZWGYNLPDUFPECPZZDMUGUUTICGUIARAODOQGDUGIZGHUALZMUBVZDD
MZDRBGUFAEFBRDMARDFOPBRPRNLOMSDSRXNOREBRADMGKANMHKIDZUMOAPL
OAAFUNRZARSIGHUSMGZDRDEPUWBEIFREOPLSFNBWAUMPTLMGNLRZARSIPIA
UXGZDPR
```

2. This ciphertext is from the British National Cipher Challenge before it was national (2001). Figure out how it was disguised. Break it. Make sure your decryption is a clean one; otherwise you may have made a mistake with its disguise. Can you find the keyword?

```
NZTFM YKDID MYLCY NSGZK VXKMX ALZDP MYLCY NSGYK VXKMX AGKOG
LCYUR EGPNY TFMZK DMNGL HWCLL DKYAB IYYKV XKMXA DYOSJ PCDHU
```

```
GDKIK UVXKM XADYO SJPCD KUJAS DLBEU VTNZT FMZKD ZIKMG JLCEU
DAYNV XKMXA IYGKP EJGHL HMAII YYKVX KMXAI ZQSNJ OLHKJ VIKVX
KMXAL YNJYG PDHLI YMCBS YLKCE GYOSX XANJD NLKDP LIQZD KHGJG
XANJD NLKDP LIQZL YDSGK DWDOE JANEB SDAVV GKDCT GIZLY LSDFG
DOEJA NEBSD AVXAG LXADS TFEKI MCDMX XAOSD SJELX KMPEH CQDKS
GZKDQ SIZGI VMDSJ EGXKC ZMGPG LKUYM JGKYK IMZFN XADPN YKIZK
IMKZY GELJN YMIDN JDMOK IKYOL HDPOL LEDPC LSDYJ EUAGK DQISO
QDMGJ ZGEGO SDGLH FGPMC SBPEJ GUGXE JKSII MXAKY KZCMG FWKKN
PHJSK DSMBS FKHMD GQYKD MGMZG JIKJT GIZGE BYNAG KDSNB DEDGL
AWKDJ NIGYG GIXAK OAGBQ XKGIM TGEKB ZKJTG IYNGJ CHYGX RKYOL
LEHQK IBGEW GJQOK THCBS XALKB SMZNH PDJMG FWKKN SMGMB GSDYQ
NZTFQ IZIBX NEAXJ QBJZN GWKDK USMVX DPDET FXD
```

3. Break this ciphertext. What is the keyword (hint: it is a name)?

```
BIPAFKWLIFETHACPEGKWCKKPIUDQCWPMLKBIAMSFFSAFQDANHOBQTBBOKOS
CQBTBQCFWUHHACPATIRFYKWUBQOIRHOOKUPZPQVONLCPKWOMKGRCRKQVIPU
BICMHACPATNLHWHFPMHOIQQTQNKNWOIXSFRCCRKQPRHIOQQTQNEHQOOBQBU
QGICPKPAPBTBTQOWOBIKUEFKMRCCRKRQYQYUQKTCBCPKPAFGYSLPKCMQEKP
UHAUDAIFKQBIAQQMINQCBICUKUOKMROWGYFSAFQDHORFPUPKKFAIXBCPGAR
UZCUHXBKIMKWOFHQTRABNWOARTAOBOUINTGUCQYOKMRQBCPKPFEKLPKCMQE
KPUHAUMIHACPTQAQNEQKPAKPUKQHHOFXQALNQAPZPUHDKRUKOKVNBIKUAZB
FFKFYKWYLAMQNVNHOFSQDUCQYOKMRQY
```

Challenge

Not square.

```
G0IJVEFCDLZMDECBNAV4VIWGBRCEACKTVETFCKEWDBIZPMRFNETSZOVCJPL
KIEIPYCICKTDIZUHTNBUEFTGTCAFCNTIJTSGFCGATKLVFVE2FUY2FGAJTNZ
GZKTCIFVTFEXACKTJCUYUVNUTCRFRFKT2FJBENIVGLJEGDDLCTPOOBCACBV
GJCCKVIFCEAILFCUFNELFIEFCPL2FTQCENGIRFCUFNELFSFVNCEGNBGNAQC
TFGAFCCIKTEYSA2FCFBJFBGAFCFBETBFIVKLPHNGPLFATAG0NGPL2FTQCEF
RFBRKZFCKFVJZEXVIWCIVKLNGHRDTH3FYANGTF2HPECWNODBCFJVENGTCVJ
NZGONJTFANK0FBNEYFNBBCENRKFJKYFGAQGFNJCNNZOZ5JGTFSDLCIFVNBO
PVEXCLHEXTHHZZEVMTKGACEDBCAKTAYJCAVF2NSCRVCVILK5JK0FBNEVEJC
NVCICNNZOZ5JGTGFRJFSDLCIGFILJPLKIEFCPL2FTQCEJCUFVIFCEAPLNSC
GLIHLCQFREJNTLISTEBKZPMRFRCTNVUTFBKENTAHTCNNZOZE0CESTEQUFNE
CGCNEBVGFCWEFGOPIVKLCINAPHCINSCNEAIDMUVFTNEZJBTCNGIRJOCNECH
PXCIRPLCTOZ2FGZLPXCHQJ0VIICIJYFILJPLKIEFCPL2FTQCENGFPGBWGGN
OPBDIPVULHPGKTNAJPLKIEFCAVPJZEFSIVCKCECGIVGAFCNSACEGNBZJEBJ
QFCEXJBTCNGIRJOCNECECBCLNACIRILFCDTDBPDCEZFQCTCCILFE0CESTEA
FCZJEBFPKENAQCASIJHUECJGKTWEK0FBNEUYVEENGLCBZVEKGJOTJTERVCV
IFVKTECNBVIZUJOOBJHFTJRRKCICSRKCEGZKTCYTKGVPHCIBFTDGTJDNBDI
CGPOCGTJXEJZTBCNEBQGTSCNBCKQ
```

Unit 107
Vertical two-square cipher

The *vertical two-square cipher* is a digram substitution cipher that uses two Polybius squares, one above the other. The plaintext is padded to an even length and divided into digrams. The first letter in each pair is found in the upper square, while the second is found in the lower square. The ciphertext letters are taken according to the rules:

- If the plaintext letters are in the same column, then the ciphertext letters are the same as the plaintext letters.
- If the plaintext letters are in different columns, then we find the rectangle that has them at two of the corners. The ciphertext letters are at the other two corners. The first ciphertext letter of the digram is taken from the upper square, the second from the lower.

  An example couldn't hurt. Take this plaintext:

```
THIS MESSAGE WAS ENCRYPTED WITH A GRID CIPHER
```

It has an even number of letters, so we do not need to pad it. Let's use the keywords `POLYBIUS` and `KEYWORD`, and fill the first square by rows and the second by columns.

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

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

The first plaintext digram is `TH`. It defines a rectangle and we find the ciphertext digram `WY`:

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

K R F M T
```

```
E D G N U
Y A H P V
W B I Q X
O C L S Z
```

The last plaintext digram is `ER`. They appear in the same column, so the ciphertext digram is also `ER`.

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

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

The full ciphertext is

`WYAOKDALSNDBASGDUTYPTEDWCKEVEMUESXLYER`

Decipherment with the vertical two-square cipher is the same process as encipherment.

Reading and references

Félix Delastelle, *Traité élémentaire de cryptographie*, 1901.

Wikipedia, en.wikipedia.org/wiki/Two-square_cipher

Crypto Corner, crypto.interactive-maths.com/two-square-cipher.html

Warren Thomas McCready (“Machiavelli”), “The Twosquare Cipher,” *The Cryptogram*, Nov-Dec 1972, 152-153.

Programming tasks

1. Implement the cipher. Remember that there are many ways to mix an alphabet with a keyword and many ways to lay a mixed alphabet into a Polybius square.

2. Implement a dictionary attack. Remember that there are many ways to mix an alphabet with a keyword and many ways to lay a mixed alphabet into a Polybius square.

Exercises

1. Encipher this text with keywords `HUDSON` and `EXPLORE`. Use the same methods for mixing the alphabets and for laying them into the squares as in the example above.

   I take for granted that you are tolerably well acquainted with the different modes of life and traveling peculiar to European nations. I also presume that you know something of the inhabitants of the East; and, it may be, a good deal of the Americans in general.

   (from *Hudson Bay* by R.M. Ballantyne)

2. Decipher this text with keywords `MAPLE` and `LEAVES`. Use the same methods for mixing the alphabets and for laying them into the squares as in the example above.

```
XCSOKGSOMYHBMQBWSOLYEWLYMXPRHZSTNZQCMZLGMBMPWILWQQBPVGICHHV
PPRQKIQMAMHMZXBHZAYHUHDBWVDTIMBAYNYVDNTIYUHHTKEHPPGCNSCEVXC
SOSMKXPWIRACTTHEQCVDSNNZELDBIYPGYTKEITKGSOLZOWHTMFLZHELOITD
IITWRACTOSCMZSOLGFLRDLYAXSOLYHFLYSOAZTBTWIYDBMXMKIQBXRXVTNL
MNKZSOPYPCHTTRMWQQIQTBVGTWKZTBVGIYYBIXPZNLQZMZMBPPBINTICMBK
GBISIDZVDMOKGDWNTHEKEDUVDZQ
```

3. Perform a dictionary attack on this ciphertext. The keywords are short.

```
DUDAMXTENUICFGMPRLUBGGAMFQQIMXOLLWQQOMESITRSSGLQLIHMESDAXQO
IRXTMBDNHKDOUMODUWCRLFGGSBPILBOSXECTERWECNLVPNSMKQDOFESQQLW
HMESBALMQQBOSZDALOMNDOXGSPEGREFGECGXCBAFVUMIDNNCGGQQSPKDATD
BXDSPGSOUHCNXASHTFQSINITSFEKIASLISPGGBHFFBPDZGCDGTLQXHTFQMN
QNREMTDHHDRXOMLSAZQBFGIXGGSPEGREATLPECQDHMFQMNQNMISPHFLQSPM
TSQMEDGSIPKBWSXFRCPEGLNWEDAVGYTESELVETGMBQIELBPSPSPATESMOSP
HEQQMLGGLWAEHEBZQEFEWTCBFFGGTYHEDUECLAOXEDDQWEWCUDESLNRLBWM
IBOSXECQBNSMCEOPQPIBPFOLNLGNNDBLVDEAEFEYSGFGEBPQQAFTHREECIX
TLLPGUHKSYGCEDGGQQSPKDATDBXDSPBPTXHTDUQPHEOUMIEDBSQDTLGUDAB
VOSRMQDOFESDIRETMNUFDSPDBBALM
```

Unit 108
Horizontal two-square cipher

The *horizontal two-square cipher* is a digram substitution cipher that uses two Polybius squares, one on the left and one on the right. The plaintext is padded to an even length and divided into digrams. The first letter in each pair is found in the left square, while the second is found in the right square. The ciphertext letters are taken according to the rules:

1. If the plaintext letters are in the same row, then the ciphertext letters are the same as the plaintext letters but in reverse order.

2. If the plaintext letters are in different columns, then we find the rectangle that has them at two of the corners. The ciphertext letters are at the other two corners. The first ciphertext letter of the digram is taken from the right square, the second from the left; i.e., the first ciphertext letter is taken from the same row as the first plaintext letter, etc.

Note that there are other versions, which involve changes to the rules. We can change rule 1 to keep the letters in the same order; this may be a weakness.

1′. If the plaintext letters are in the same row, then the ciphertext letters are the same as the plaintext letters and in the same order.

We could also change rule 2 to reverse the order of the ciphertext letters.

2′. If the plaintext letters are in different columns, then we find the rectangle that has them at two of the corners. The ciphertext letters are at the other two corners. The first ciphertext letter of the digram is taken from the left square, the second from the right; i.e., the first ciphertext letter is taken from the same column as the first plaintext letter, etc.

For the rest of this unit, and for the examples, we keep to the version of the cipher in the original rules.

An example couldn’t hurt. Take this plaintext:

```
THIS MESSAGE WAS ENCRYPTED WITH A GRID CIPHER
```

It has an even number of letters, so we do not need to pad it. Let’s use the keywords `POLYBIUS` and `KEYWORD`, and fill the first square by rows and the second by columns.

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

The first plaintext digram is `TH`. It defines a rectangle and we find the ciphertext digram `LD`:

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

Then `IS` → `NT`, `ME` → `WU`, and `SS` → `NW`. But then `AG` is on a single row, so it is enciphered to `GA`.

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

The full ciphertext is

`LDNTWUNWGAYMNXPUDBMGOIYKUPAHAYDIGRFDAO`

Decipherment is the same process as encipherment, with the two squares swapped.

Reading and references

American Cryptogram Association, www.cryptogram.org/downloads/aca.info/ciphers/TwoSquare.pdf

Wikipedia, en.wikipedia.org/wiki/Two-square_cipher

Crypto Corner, crypto.interactive-maths.com/two-square-cipher.html
(Note: They reverse the order of each digram, compared to our method of encipherment.)

Fred B. Wrixon, *Codes, Ciphers & Other Cryptic & Clandestine Communication*, New York: Black Dog & Leventhal, 1998, pages 219-220.

Programming tasks

1. Implement the cipher. Remember that there are many ways to mix an alphabet with a keyword and many ways to lay a mixed alphabet into a Polybius square. Use the original rules 1 and 2 as the default behavior, but allow for other variants.

2. Implement a dictionary attack. Remember that there are many ways to mix an alphabet with a keyword and many ways to lay a mixed alphabet into a Polybius square. As with Task 1, use the original rules 1 and 2 as the default behavior, but allow for other variants.

Exercises

1. Encipher this text with keywords `BACON` and `CIPHER`. Use the same methods for mixing the alphabets and for laying them into the squares as in the example above.

   For months and months the eye has been assailed by paragraphs and pages in the literature of two worlds, contending for or against the existence in the Shakespeare plays of a cipher that would assign the honor of their authorship to Lord Bacon.

   (from *The Little Cryptogram* by J. Gilfin Pyle)

2. Decipher this text with keywords `POLYBIUS` and `SQUARE`. Use the same methods for mixing the alphabets and for laying them into the squares as in the example above.

```
FLHSHXESUORPOHLSVUAMYBIVMFQONAXPQYTDQVQFGHCELDBSEGMOSKKHKLL
IOOSOFTNYSEBLPMQTUOLSBDORDBBBCPUKDNLSBDICWYCUCOPWCBUNTEQVNT
OVCBDONFISQPTDKHNAQNDNDOQYQVNIOXCBYUNTOZCBQFEPKHSVUXUMTIUON
FUWMOOVCBDBQXCUNVLIDBOVURFKUOUVFOXOOUTDEBNFNHPMQXCMNAOPUMQF
DBUKDNLIXMQXOVCBBLLIOMUVHFNABWSVBGUVRKNIELKLLIEVLDDOFLDBMDQ
LUOBLODUNUXUMFPPMQTUMDNODKEKFMULIIVBGCMBYPEKFNAQNOWUVNHPWOU
IDKLOFABNMDBAMWRNIUXCOBBOFPBNAGYOUUVNXNDALLSOULIXMIDIPOVNTC
BENNTLFLISEFKQPSDFPOFDCQXIVMXFLVLXMUMABLIOOBGCMMUQKALEURKNI
CMODKEKFMUNIUEPHEPUWMOLIIVNAUPMHDRUVNIOGPWCBBQKHNXBDTDLICMN
ABUUMDNVRLICMUORTBELDMUEVLIOMUVHFNAOOGDDBLIXMBXFPYCQXUPBQCP
OTMVURTTCBODKEKFMUEVQYQKLIUVMONANHASBONWMULIOELIUVURDNRNNSM
VSKPZQKVF
```

3. Perform a dictionary attack on this ciphertext.

```
NIUGOTANORETOTQLWAQNLATSKAASARQNTHSARPQVLTCIEMXUNELTLANTPMQ
DLGKTDEBBDEESSHINKGADXSSEINKGMMLPTSKAASKCSFKBNDAODRLTHRRFZU
SLXSRGBKUOUDNELSFHGOIBWFRGWAKZNPNOKARFLEONDNSTSASUUPPDBKTZN
PEBHRLATSEMAENEUOSEGDGOLZNPIHNTKAWBLNUGAETTDNQNTHSAUGONFORF
UDNEUGBGPDSSOTZUSLROOELROTQFTHECNTARRGODNTKAVHQLSEHDKGFORGD
KSURGNTGOLZMMRFFHHVNICNLZLEUDNETSKUMLROAEAMVNNGHUBREOMPOOHH
RBEMDESEUGANUHLUEOPOOMLP
```

Unit 109
Hill-climbing attack on the two-square ciphers

A hill-climbing attack uses two mixed alphabets and the parent key/child key paradigm that we saw for the Playfair cipher. When it comes time to modify the child key, we randomly choose to swap two characters in the first or in the second alphabet. It is not necessary to flip squares or swap rows or columns. To avoid being trapped in a local maximum, a margin of 0.2 works well.

Here is the algorithm:

1. set the two parent squares however you like
2. set the best fitness as the fitness of the unmodified ciphertext
3. set the counter to 0
4. while the counter is less than 10,000
    a. copy the parent squares into two child squares
    b. randomly choose which child square to modify
    c. modify that child square by swapping two randomly chosen elements in it
    d. decipher the ciphertext with the child squares to get a new plaintext
    e. calculate the new fitness of the new plaintext
    f. if (the new fitness exceeds the best fitness) or
       ((the new fitness exceeds the best fitness minus 0.2) and
       (we roll a 1 on a 20-sided die))
        i. set the best fitness to be equal to the new fitness
        ii. copy the child squares into the parent squares
        iii. set the counter to 0
    g. increment the counter
5. output the parent squares

Once a key is found with a hill-climbing attack, we can recover the keywords by swapping rows and columns. For example, if we find this (half) key:

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

then we can rearrange columns to make the last row more orderly.

```
O P Q R T
```

```
A M U S E
H I K L N
D B C F G
V W X Y Z
```

Now we can reorder the rows:

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

It appears that one of the keywords is `AMUSED`.

Programming tasks

1. Increment the attack for the vertical two-square cipher. Remember that there are many ways to mix an alphabet with a keyword and to lay the alphabet into a square.

2. Increment the attack for the horizontal two-square cipher. Remember that there are many ways to mix an alphabet with a keyword and to lay the alphabet into a square. Be aware of the variations of the cipher.

Exercises

1. Break this ciphertext which was encrypted with a vertical two-square cipher. What are the keywords?

```
MEXESAEPQHNPLAOKMGBSUNUTXTDCALPTOUERPTGHNVGHNBOQISPQFTREFOI
HREKLFAIHREGRDHQFOQISQXMEIBHUMQLHNAUNNACEIHMTIHREWEREALPQRD
RESADCRDREPQNFMNHSDKKLEQNWHIBTPTFOUNRHCAREHSCPDIFOSLYTHISLK
LNXCQBSWHRENXMEHHDHPTGHEPTYFHOKDMIZGHKLNXMEHBAERMGIVEICIVCQ
ALREADWSTRPMGCONMESATRTNGCHQCWMGWTLLIZKLGADBNEMEIHCEIHKHRDR
RGKORPTEHDHPTGHHMDCDBPTEISNVTDHPFTRPTEBTRQTCCHLFOPTOUERPMST
UTMCIAUNBSMFORSTAIDHDCLTNAESCEIHAMTNTOMGUTNHSADCHQINDBHQGCO
KDNXTDCREGRFHGSLLXTQYLHPTADRHGIKLHNTXAHBBHLGCHLTTIAKHHUBGTR
XEULIZADCWKLFQPSFWBTKLNELDFOGHQTNAWHECYTIHGAKDONDMIESX
```

2. Break this ciphertext which was encrypted with a horizontal two-square cipher. Can you untangle the keywords?

```
QMHCPVUNGIPBYNTNDKCIXGCITHMBKNVLQMEBGITRBPKNTSSFGPAPISHPTSM
FQMFNMVFNHCMDSNICHCOPFHMOESFCPETIRRHCQINGBATIESWHSXNITQCNLP
NAIMPBGDBPBOEDQPDGEWTIPPCRADLQPPWRKPPIDDCTOCWHMBKNQMDHRPUAY
PHPGDPSILWHBWESSNDHEDMEPBLIEHRIPSBHMDQAQAEBICDFILGHCTOYDISO
WNECESQMDHWYILWHADIHHPDGEYYSGNMDQADFGNSHCOOCPSBDEQPBGAXCBIM
```

BESASUCTMDQTSQSPIBDEYZSQSMHIDQPSXIDCSMVFPEIEXEAESRBPEZEIHTI
CQVPAPCRKNFUFPTYRAPEHBPVMCUPEHAPTSQMYNWYMFPIQEHPMBSTIADIAHP
TCVRBBNADAPUSCIZEIHAHEBLFDHKUHPSRXCOIPKQAHPWHMBQIDKFPFUUFZP
EDUPAHQIDGTIERMBSPETOYAKIVKCGPMNDAHCGUQMDHDFDYEHLCRMCIKNOCS
NMIYPPIBDELVPHHQMFPAKIVKCGPPCWHUSRHPVQPUCQMDHQAQMFEEIDLQMYN
DEHPPSNMBOERURPEWNQMFNSOQIDRUCNAQMIHWHTSQMDHAHESWHPSMPCIEHD
IEHODHIADMFCSEISPHU

Unit 110
Four-square cipher

The *four-square cipher* uses four Polybius squares, arranged in a two-by-two layout. The upper left and lower right are the plaintext squares; their alphabets are not mixed. The other two are the ciphertext squares and contain mixed alphabets. The plaintext is padded to an even length and divided into digrams. The first letter of a plaintext digram is located in the upper left square, the second in the lower right. They form the corners of a rectangle. The other two corners hold the ciphertext letters, the first in the upper right square and second in the lower left.

Let's encipher this message:

```
this message is encrypted with a grid cipher
```

Its length is even, so no padding is needed. We will use the keywords `POLYBIUS` and `KEYWORD`, and fill the ciphertext squares in an unimaginative manner:

```
a b c d e   P O L Y B
f g h i k   I U S A C
l m n o p   D E F G H
q r s t u   K M N Q R
v w x y z   T V W X Z

K E Y W O   a b c d e
R D A B C   f g h i k
F G H I L   l m n o p
M N P Q S   q r s t u
T U V X Z   v w x y z
```

The first digram `th` is enciphered to `NB`.

```
a b c d e   P O L Y B
f g h i k   I U S A C
l m n o p   D E F G H
q r s t u   K M N Q R
v w x y z   T V W X Z

K E Y W O   a b c d e
R D A B C   f g h i k
F G H I L   l m n o p
M N P Q S   q r s t u
T U V X Z   v w x y z
```

This continues, and the final ciphertext is

`NBSQHENPOROZLMLLOPZIRWOXAQIYUNAWYAFCOS`

Reading and references

American Cryptogram Association, www.cryptogram.org/downloads/aca.info/ciphers/Foursquare.pdf

William Maxwell Bowers, *Digraphic substitution: the Playfair cipher, the four square cipher*, American Cryptogram Association, 1959, page 25.

Wikipedia, en.wikipedia.org/wiki/Four-square_cipher

Practical Cryptography, practicalcryptography.com/ciphers/four-square-cipher

Crypto Corner, crypto.interactive-maths.com/four-square-cipher.html

Fred B. Wrixon, *Codes, Ciphers & Other Cryptic & Clandestine Communication*, New York: Black Dog & Leventhal, 1998, pages 221-222.

Programming tasks

1. Implement the cipher. Remember that there are many ways to mix an alphabet with a keyword and to lay an alphabet into a square.

2. Implement a dictionary attack. Remember that there are many ways to mix an alphabet with a keyword and to lay an alphabet into a square.

3. Implement a hill-climbing attack. Except for the decryption routine and margin, it can be the same as for the two-square ciphers. The margin should be about 0.2 when the fitness is lower than a threshold, but near zero above the threshold. Experiment with it until you find a threshold that works well for you.

Exercises

1. Encipher this text with keywords `WINDY` and `WEATHER`. Use the same methods for mixing the alphabets and for laying them into the squares as in the example above.

   > We all held the string as fast as we could, and tried to pull down the Kite; but it was impossible, for instead of bringing her down, we were all three dragged along down the meadow slope.
   >
   > (from *Adventure of a Kite* by Harriet Myrtle)

2. Decipher this text with keywords `NUMERAL` and `CIPHER`. Use the same methods for mixing the alphabets and for laying them into the squares as in the example above.

```
CTBTLONUTEISIVRHSBOINKEUHTMNMMPNUUNGSBISDPEKEYFDMCDMDPPKMUO
PDITPZCOIHCDMDPQQXTTHHQGUMUUSLHEUAPOHFGKSKIOIHOBMQDUEIHYMAC
KIQSURUNQUPHMUKEFGEEKTOHIPXAAMEUDPTIBTILEMPEPHSUQDEOQDBTBMT
STIPBTIEOUUMCRSOIEXCTNGSLLQNEOMYDHPTIMOMUQDNNMNCQCLBOQELLMU
OPDIQX
```

3. Perform a dictionary attack on this ciphertext.

```
EQVOMWHHQSPEHMLWCGEINSBEPEILCQFDCDXCSBAIROTWTBPPPLQCRWXCGLF
HQPFTYTCDIPLARQROMWROQPOKBRMITABAFWSKGGUKHMGLHMNSIZLAPNFHRO
PEGGHMFQAWQNUKGSPEGSILLHGGTWBEROFLRWPEGSLAVOMIAELAPDTAQKAMF
WMLHMSKUCBRPPIRROTZGFHRTOGLGZKAHPHATNQEFWLFOBDWHLARZIGAMBQE
DKEQOWGKFQHIKOQIPEIABBNKGLZOROCBFTAOHQKQRWPEHFQHIMQPAWZQRWH
QAWCVLLDTCYLZISSKQIKNGDPEHMPELLGGTBROFGCDTGKAIMHAFTAAIRKMLL
SKHLFPAWEIQCYLFPAWCWGAIGHSROKNFGPSGGHEFWSQQEDWMBFWETBWBGVYG
GSQAWHEFPAWFCQPQIUGCQFDCD
```

4. Perform a hill-climbing attack on this ciphertext. What are the keywords?

```
DSVNOOSGDPFOHMLETLEOLWSFFLMHSTCIAKNSMRSVACAILZRGGKUNAMTHTNG
KCIENSFRMGSEOSAMHNOISACMSFOTFHKUORALTLZAPLCACLUCSLMACUEMHVF
GLFNLKNKKILZDHTOYFSNACLWTNGPBXAPAMNSCDSIFOPCPZLTYINOXDKAGKU
NAMCIFKGRGLYDRARAUMATSANKKOPLMOAPTRAGNOXQYFOMLAMSBRNSCDRAPM
ACPWMOPZLTYINOYFOMGSPCELGFSTDSHNMXYFOMLWPMFWZKHNMXYFOMGWNSG
QUQQIACWYNSMKFNSIMRFOBSVNVWRRMIHOTHKSSGPFGTKSULDTFOYDTOHNMR
FOFONCLTFSSRMEFLRACSRACSMVRAONMRBRNSCDRAPMACGSEOOAONBQACQIY
SDHBEMSLCGPFOUQGWEORALZCSLMACHARAPBLCEIYICPUEVWFOMMRRATTOAS
MSFOUNMEPZLTRRGEGSFODGCDATTWHHBADRAMNNMRFOTNNNINLTGLDGTHTNY
ENSOHEYATFOEOLSPLAGGLRAAMBSVKHOACAIPZLTTNTSRFGLCISIRAALXWNK
VNRHPZLTLEGEKRQLNNQDGLCIRALMQTDZSDZKSFFCDXSGPOHEISKQHNMXYFM
OLTFPHKMORMUNMEPZLKNKRACICQZNTNTNMMSQGLHADGKDABGKNLODISNNLI
BSAKNKRGGKHYLTGHNNRAGLPZEIEOASMMFLMRMSFODFASUQRACSAMLMFWOOM
OGAQOPMFWZKRQOODWRACSDFOWHNVNOOVKCITOUSUORDLZNSMKTHAMRACSPW
XSCINPZRSVROISNOGSFOHAFKSDIGQXAPYDRANSOIPLRFCIENMOCMLZNODSF
OTNZORNCSATMHRQPMCKAPSAENSAKI
```

Unit 111
Phillips cipher

In the *Phillips cipher*, a Polybius square is filled, and from it, seven more squares are generated. Each square is used to encipher five letters at a time. The second, third, fourth, and fifth squares are generated from the first by shifting the top row downward one, two, three, or four rows. The sixth, seventh, and eighth squares are generated by shifting the top row of the fifth square downward one, two, or three rows. A square enciphers a letter by replacing it with the letter that is in the next column and next row, with wrap-around.

An example is essential when the explanation is as poor as the one above. Let's begin by filling a Polybius square with the keyword `POLYBIUS` and generating the remaining seven squares.

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

     4              5              6              7
I U S A C      D E F G H      D E F G H      D E F G H
D E F G H      I U S A C      K M N Q R      K M N Q R
K M N Q R      K M N Q R      I U S A C      T V W X Z
T V W X Z      T V W X Z      T V W X Z      I U S A C
P O L Y B      P O L Y B      P O L Y B      P O L Y B
```

Notice that squares 0 and 4 have the same effect, as do squares 1 and 7. Now let's encipher a short message. The first letter enciphered with each square is highlighted with pink in the square and below. Notice the wrap-around used in enciphering 'T' to 'O' in square 0.

```
plaintext:    THIS MESSAGE WAS ENCRYPTED WITH A PHILLIPS CIPHER
square:       0000 0111112 222 233333444 4455 5 55666667 777700
ciphertext:   OKEG WNYYBRL AHG LYDPZVONM YEOI R EIVGGVEY POEKNT
```

Reading and references

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; chapter XIX.

American Cryptogram Association, www.cryptogram.org/downloads/aca.info/ciphers/Phillips.pdf

Programming tasks

1. Implement the Phillips cipher. Allow for several choices on how to mix the alphabet and how to lay the mixed alphabet into the square.

2. Implement a dictionary attack.

Exercises

1. Encipher this text with the keyword `EDGAR`. Use the same method for mixing the alphabet and for laying it into the square as in the example above.

   > Now the irony is this. In this walk, so many times repeated, the world's greatest master of the terrible and the bizarre was obliged to pass a particular house on the eastern side of the street; a dingy, antiquated structure perched on the abruptly rising side hill, with a great unkempt yard dating from a time when the region was partly open country.
   >
   > (from *The Shunned House* by Howard Phillips Lovecraft)

2. Decipher this text with the keyword `RHYMES`. Use the same method for mixing the alphabet and for laying it into the square as in the example above.

```
UBSSOOWGUHWHTWFUVHVYDIVLSPGFOWPGFWSNKHAXSAQWAUWUIFKNGUHQLRQ
YVGWOIAWGLFGWPFSIYUWNFMQYSHUYNFGIONVYRGYVSIFQNKUWSFGWSFUYHU
IFCMSASQVUQLNGXIGRCUEASGUQWVMIFUIFGWAKWVFQVUBSGFRYKHGIBNIBU
VXPAIYSIVUCZHWGKZBANRGFIFUMWEIYXUNOGWHRWUZYAUSIAUIHGFMQSIBA
UYKYMV
```

3. Use your dictionary attack to break this ciphertext.

```
RAPSZUHLHMRIXQGAPPHIZURAQKOHXLGHLYGOLKAUPSFALURHMSZUXNKAESX
ZZBRAYHUBHIKCLYDCDOZUDKALSBSMHMNTUSZVBVZIXAIZARIBUKXHFCAVPS
HMIFLZAXBSIHLKHSTVUIZAGIBUCXHXUBSFGHWLUGKLDVDPHKEIKAXZQKAEI
ZVWAZZIXQKLHYSTHYRHUHPHVRDFAZHXBSRQNUDIMSZERAVCKYDIZSHFYSLU
HXHZUZ
```

Unit 112
Hill-climbing attack on the Phillips cipher

Our hill-climbing attack on the Phillips cipher is similar to the one for the monoalphabetic substitution in Unit 36. The parent and child keys will be mixed alphabets, not sets of filled squares. To avoid getting stuck in a local maximum, we will use a margin, as we have done before. A good value for the margin is 0.5. Here is the algorithm:

1. calculate the best fitness as the fitness of the unmodified ciphertext
2. set the parent key to the alphabet without `J`
3. set counter to 0
4. while counter is less than 10,000
    a. copy the parent key to the child key
    b. randomly swap two letters of the child key
    c. generate the eight squares from the child key
    d. decrypt a plaintext using the eight squares
    e. calculate the new fitness of the plaintext
    f. if (the new fitness exceeds the best fitness) or
       ((the new fitness exceeds the best fitness minus the margin) and
       (we roll a 1 on a 20-sided die))
        i. copy the new fitness to the best fitness
        ii. copy the child key into the parent key
        iii. set the counter to 0
    g. increment the counter
5. output the parent key

Once we have a key, we may not have *the* key. The reason for this is that if we roll the entire square to the left or right, then the cipher has the same effect. For the example in the previous unit, these versions of the first square (square 0) are equivalent. No matter which of these you choose, when you generate the remaining seven squares and encipher a text, the ciphertext will be the same.

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

If we find, for example, that the key is `LYBPOSACIUFGHDENQRKMWXZTV`, then we can lay it into a square and roll it until it appears to be in order, and thereby recover the keyword.

Programming tasks

1. Implement the attack. Use tetragram fitness.

Exercises

1. Break this ciphertext. What is the keyword?

```
DWCFPHQAZAXMZZLZPLYARDZDMEHHDUUGLFGZPSKMFDWLHOIKIHRFOMTLCQX
CDZPCSLBREHQQZKDLNMAQLFEABIGVZHSMTNMWXBSASBZCWUKUDGHSOZFHQA
FQDHFLOAKRATQSLZLLZTZKMLQLFKAAIVBDFHULICCGZFCBOAKFCATTILKVT
ALQLQDHIUULRIKUSLPZIQGKHCFQSZUGLZMZGMSOWQVWQSZQLGDHCCGWPSOA
LCGFDFHMLTHQSWCLMAMZHZLYOCVZZICUDUVHGDLHAWGHQCWLBICGHNRMGPL
YWQLQSELHWULGLPHIHGUANGKACGBQLTGKWCRNTLRLWYVBLMLHGGKGRGAHIG
DZHKGMSDBCOSZSHQNZKQVASZAVFHQDOUGBIIWKZFH
```

2. Break this ciphertext. What is the keyword?

```
BVGHUUWZKGDQLDDFZUFYVMUIVLNZKUXALMRWGHKWGHVABNNSNGHERZFYFOB
FRBCBILBPNVMHPMUZIKSZFVMULFZZWOUWETFPXIFZAMZECGHMMVFMRXEIFV
HKFABURURHVQRQIPNOZYZUVMURHZPFVHKUGHMRURHVQCXIKNHODARQVFRYV
XNNZFGHBVIVLFAUHPBADNHOKICBVWKKZWGHQVRAIZUIFMZEFZQVLCXEVFAE
GHNNVMUPNFAZHKDQINZZFOHIERHZGFVOLFAFMPNFAZHKRNHAHVLHAKUQLFP
XNNIUUHUUHHVUCNHEUALKHPNSAHVEBAKUHOFTNVNWHWWHBVURFHZNVLKMRU
BWFXNWABNNKWAVSDWFUNGUBFRHVEMZPFFUUCUGHKUISVURFZFPMZWPZSGFR
DQYZFPXVBUBMVKUFZUQINXUWRHKCXFFLRSVLRQRRXFFLRWUZFQINQUNCHKX
ZMHFVKRWBCVFSWZZWOOWFRRXIRXYLRMBLNFAGFELK
```

Unit 113
Phillips-RC cipher

The *Phillips-RC cipher* is a modification of the Phillips cipher in which rows and columns are shifted when generating the eight Polybius squares. The encipherment proceeds as in the Phillips.

Here is an example starting with the same first square as our example for the Phillips cipher:

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

    4                5                6                7
U S A C I        F E G H D        F G E H D        F G H E D
E F G H D        S U A C I        N Q M R K        N Q R M K
M N Q R K        N M Q R K        S A U C I        W X Z V T
V W X Z T        W V X Z T        W X V Z T        S A C U I
O L Y B P        L O Y B P        L Y O B P        L Y B O P
```

If we can break a Phillips-RC ciphertext, then the keyword may not be obvious. We can roll the square until it becomes apparent, as explained in the last unit, but for the Phillips-RC we must roll horizontally and vertically.

Reading and references

American Cryptogram Association, www.cryptogram.org/downloads/aca.info/ciphers/PhillipsRC.pdf

Programming tasks

1. Implement the Phillips-RC cipher. Allow for several choices on how to mix the alphabet and how to lay the mixed alphabet into the square. Feel free to copy and modify your implementation of the Phillips cipher.

2. Implement a dictionary attack.

Exercises

1. Encipher this text with the keyword `STORY`. Use the same method for mixing the alphabet and for laying it into the square as in the example above.

   > Once upon a time there lived a cat of marvellous beauty, with a skin as soft and shining as silk, and wise green eyes, that could see even in the dark. His name was Gon, and he belonged to a music teacher, who was so fond and proud of him that he would not have parted with him for anything in the world.
   >
   > (from *Japanische Marchen und Sagen* by David Brauns)

2. Decipher this text with the keyword `FRIENDS`. Use the same method for mixing the alphabet and for laying it into the square as in the example above.

```
ZQCUBXHDEFKYZOHTOVMLZZQCBQMPXKSUHOAWFMNMBDPLMUMZXOUNXIOPFQL
FQCUBSZMQVYULZCYDEHIUCZQBOQXFXPUZORULZFVXZQCUBXGEPTVYDQUZLH
EZALXLDIZOMOWPZUQKAQNMCKBACQZXOPMMRULPEULACZWZQMTLHQZOULIYP
FNLZACCIBHCMZUHRNHHZLHFCLAGQXKEDEEEYPZNHLXXNQBOWGHVTZSUHGSO
MULZZUCOWWFMOEXXUPUMFNILKKMNLGQTMMLGQSUMFEIVUBPQZLHDOLHIZIU
OYNWALKICDZVYZXULOYGIFXXHPDLICIEDECZIUQYIVKFQLZMWVGQZBMTLGQ
AKZOMNCLKZK
```

3. Use your dictionary attack to break this ciphertext.

```
NGQFUQBVQLVAVWKHKTXAEPGFZETVMHMRUOVYQZZVSPZQOQMQZZRPOAMAQDK
WUSBUQAXVBZGNSRYRZVUBBCNUIQPRLWPRMFETEXDWQUSQIMVBMUNSRGOTRV
FTQZPRUELAZVUAMHRWPNTNQZGQVUUGBFOXUGVEXRVSQCBFEZIEXRSRZRZUV
OFYIBZBFINCEUGVPNNEWRZKAURPYXQHRYQVSRZYVUTXFQBYUGYLMORNSRIF
MMEWQZNFUSRFTTKMZKBXYYTNGQYQZMCBNRRVPLMAZRRPPYOQFVBYFRZDKVV
STYVRUNFHPCUGBHSPMRMELAHEZCNDQVSRMFLVQMHQRQOTQFPYTNLZHMIRNP
HEZSVSQTBCCEHED
```

Unit 114
Double Playfair cipher

For the *double Playfair cipher*, two Polybius squares are used. Each is filled with a mixed alphabet from its own keyword. The two squares are set next to each other. In addition, the key for the cipher includes a period, which is simply an integer. The plaintext is first padded with an `X` if its length is odd, then divided into blocks that are twice the period in length. The last block is not padded to twice the period. Each block is written in two equal-length rows; this is called *seriation*. Each pair of letters formed by taking one from the top row and the one below it from the bottom row is enciphered together. The method of encipherment is to identify the top letter in the first square and the bottom letter in the second square. Then,

- if the two letters are in the same row, encipher them to the letter just left of the second plaintext letter in the second square (with wrap-around) and the letter just to the left of the first plaintext letter in the first square (with wrap-around)
- if the two letters are in different rows, form a rectangle with them at two corners; the ciphertext letters are the letters at the other two corners, with the one from the second square first

The resulting pair is enciphered *again* with the same rules.

Time for an example. Take this plaintext:

```
THIS MESSAGE WAS ENCRYPTED WITH A GRID CIPHER
```

It has an even number of letters, so we do not need to pad it. Let's use the keywords `POLYBIUS` and `KEYWORD`, and fill the first square by rows and the second by columns.

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

Suppose our period is seven. Divide the plaintext into blocks and write them in rows:

```
THISMES  ENCRYPT  GRIDC
SAGEWAS  EDWITHA  IPHER
```

The first pair is `TS`. They are on the same row, so we take the letters to their left, with wrap-around, and get `LZ`:

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

Repeat with `LZ` to get `TW`:

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

This continues until we have the completed ciphertext:

`TWFAATNIOYRAXMTAMZAOMRIVASEAPRIGAAFQAK`

Reading and references

NOVA Online, “Decoding Nazi Secrets: The Double Playfair Cipher,” www.pbs.org/wgbh/nova/decoding/doubplayfair.html

Noel Currer-Briggs, “Some of ultra's poor relations in Algeria, Tunisia, Sicily and Italy,” *Intelligence and National Security* 2:2 (1987) 274-290, DOI: 10.1080/02684528708431890

Programming tasks

1. Implement the cipher.

2. Implement a dictionary attack. Remember that there are many ways to mix an alphabet with a keyword and many ways to fill a Polybius square. You will have to input the period or try several periods in your attack.

Exercises

1. Encipher this text with keywords `DOMESTIC` and `FOREIGN` and period 5. Mix the alphabets by adding letters after the keyword from the beginning of the standard alphabet. Lay the mixed alphabets into the squares by rows.

   > Not being, at this moment, in the pay of any press, whether foreign or domestic, I will not, at this my third landing in English country, be in haste to accomplish the correspondent’s office of extroversion, and to expose all the inner processes of thought and of nature to the gaze of an imaginary public, often, alas! a delusory one, and difficult to be met with.

(from *From the Oak to the Olive* by Julia Ward Howe)

2. Decipher this ciphertext with keywords `GRIDIRON` and `FOOTBALL` and period 6. Mix the alphabets by adding letters after the keyword from the beginning of the standard alphabet. Lay the mixed alphabets into the squares by rows.

```
DTQFQAKMGIMEAEQHRVZDATAANIFOHUTMFIBXTTRFQMFIFIHDGURWTIPSSBI
KTEDFLAKVANUSOHIMQBVASSACONLDVEALAEHNGIGUELPEESGPCBFNFIRFSI
KVITNGQMDXQBXISIAHIASHAIGKLBECCQAFMMGUTTFDNKEIDBSIHUCDNCOMK
KGIMMGXTTTBRCBCRBDPBDSLZZLDQEDPBMSMRAFFELMESSBDOCEEREPELBII
PGQBTFTTKBTBXSKKGUQHFNUDQRDYLU
```

3. Perform a dictionary attack on this ciphertext from the 2004 British National Cipher Challenge. The keywords are taken from this list:

```
ANSCHLUSS
BLITZKRIEG
DEUTSCHLAND
DIRSCHAU
FATHERLAND
FEUERZAUBER
LEBENSRAUM
NORDWEST
RHEINTOCHTER
SONDERAKTION
WASSERFALL
```

```
ZONOP UXRFO VMNUS VERUZ XPPLS VOHMZ XGZBK TTQWL LFWAC FTKTA
HULIP LYBUP DUURL FXHXW TOSTZ IBODK WYLFQ FWYNF EDZVQ RBOME
SFHGT AHUUV QBIZR GFZNE WXWMV FCXMF WBLST DISQA NGTPM CHISA
CLVWX IKLFM OZCKW XHRNW MELKB GSNSA MECOL KWEYP TPZDI DWKCW
VFWOI ZSCID GLMTT PNUIS TVMII SEKMI WLZBT CXXLF ZADTT BFQAE
UGWMM XRWME VLVVF ZTDNP ICPIZ LLICO GIHDN UBIOI OHNGZ WLGWU
QFMBT PEWBO CDZPU TZBKS PXFFT GVYUG ZUVEV LCAAQ FMPSO OMBVE
TLZEW ISAQL CPKIH ZVDSU TLVEL FCQUV VIMFS WWYOZ ICTSZ MSVZN
HBNOX SQFTD LFMCQ AMMLI MXLLF ZCICO YGEFU TCABO WRAQQ IYXLI
PUHIS ACLWM UVDGH PZISR QIWQT AUFQF SLOWV XWTWQ VMNOA HCFME
ZKFRC WFAMF QWFQM ZUFUU TPMHA QHFFF CNGGS UKDWL EIIIQ ITKQI
KDIMB OVUXP FMSLC PYXZM UMLIS W
```

Unit 115
Nihilist substitution cipher

The *Nihilist substitution cipher* begins with an alphabet mixed by a keyword and laid into a Polybius square. The row and column labels are 1, 2, 3, 4, 5. The letters of the plaintext are converted to two-digit numbers by taking the row label followed by the column label. A second keyword is used in a manner similar to the Vigenère cipher. Its letters are also converted to numbers with the same Polybius square. Those new numbers are added to the plaintext numbers. Optionally, any sum that exceeds 100 is written without the leading 1; this does not lead to any ambiguities.

Your are probably expecting an example at this point. Let's begin with the keywords `POLYBIUS` and `KEYWORD`. If we fill the square in the least imaginative way, we have:

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

Our usual plaintext for this part of the book:

```
THIS MESSAGE WAS ENCRYPTED WITH A GRID CIPHER
```

And here are the gory details (at least some of them):

plaintext:	T	H	I	S	M	E	S	S	A	G	E	W	A	S	...
plaintext numbers:	51	35	21	23	42	32	23	23	24	34	32	53	24	23	...
keyword:	K	E	Y	W	O	R	D	K	E	Y	W	O	R	D	...
keyword numbers:	41	32	14	53	12	45	31	41	32	14	53	12	45	31	...
ciphertext:	92	67	35	76	54	77	54	64	56	48	85	65	69	54	...

The full ciphertext:

```
92 67 35 76 54 77 54 64 56 48 85 65 69 54 73 75 39 98 26
56 82 73 63 67 74 63 80 55 75 77 35 84 37 66 42 76 64 59
```

Reading and references

Wikipedia, en.wikipedia.org/wiki/Nihilist_cipher

American Cryptogram Association,
www.cryptogram.org/downloads/aca.info/ciphers/NihilistSubstitution.pdf

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, pages 619-621.

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; pages 164-168.

Merle E. Ohaver, "Solving Cipher Secrets," *Flynn's*, March 28 and June 27, 1925, toebes.com/Flynns/Flynns-19250328.htm and toebes.com/Flynns/Flynns-19250627.htm

Programming tasks

1. Implement the cipher. Remember that there are many ways to mix an alphabet and to lay it into a square.

2. Implement a dictionary attack on the Nihilist substitution cipher.

Exercises

1. Encipher this text with keywords `RUSSIAN` (in the square) and `FREEDOM`. Use the least imaginative way of setting up the Polybius square.

   > O God, how easy it is for a king to kill his people by thousands, but we cannot rid ourselves of one crowned man in Europe! What is there of awful majesty in these men which makes the hand unsteady, the dagger treacherous, the pistol-hot harmless? Are they not men of like passions with ourselves, vulnerable to the same diseases, of flesh and blood not different from our own?
   >
   > (from *Vera, or The Nihilists* by Oscar Wilde)

2. Decipher this text with keywords `ANARCHY` (in the square) and `NIHILISM`. Use the least imaginative way of setting up the Polybius square.

```
44 77 59 47 45 66 78 57 36 53 56 83 47 76 89 76 44 83 38 63
58 67 65 79 53 44 26 76 66 47 55 87 36 76 60 43 79 56 67 80
53 53 56 83 45 77 67 67 37 57 39 45 58 44 89 80 44 67 39 76
66 85 55 79 34 56 60 45 45 53 68 58 34 53 50 43 78 77 85 88
44 86 68 64 79 47 97 50 53 67 47 85 45 76 68 47 43 43 46 56
57 85 76 80 27 73 60 47 58 45 88 67 24 43 57 66 75 77 55 66
23 64 27 76 79 44 76 49 27 73 49 43 78 46 99 46 25 73 40 45
85 76 88 67 23 64 68 43 78 85 85 48 57 47 26 67 66 66 78 67
53 44 56 57 47 83 66 69 36 76 26 44 57 77 59 59 65 47 56 66
58 73 69 67 57 64 57 66 58 55 75 59 35 77 56 77 49 56 58 46
63 76 39 73 59 47 95 70 23 44 49 64 56 56 57 80 33 64 27 45
```

```
85 76 88 67 23 67 26 76 79 73 97 67 66 65 27 56 87 77 59 67
56 43 27 55 49 56 55 69 56 73 48 44 58 85 89 50 23 77 56 77
49 56 57 70 36 67 37 56 47 76 85 60 57 47 39 76 75 46 76 59
57 53 30 43 57 66 55 48 43 56 59 83 69 76 89 50 63 76 57 66
58 55 75 59 35 43 40 77 58 45 55 88 27 64 49 56 66 47 55 69
37 76 66 76 76 56 58 80 36 55 30 64 69 43 56 58 56 73 59 56
48 45 68 80 36 55 50 53 65 73 78 58 44 44 26 74 68 43 58 59
45 44 56 85 46 73 56 69 33 77 56 67 55 76 68 69 37
```

3. Break this ciphertext with a dictionary attack. Both keywords end in `-IST`.

```
46  86  52  67  74  45  74  42  36  65  45  66  36  45  57
35 103  54  56  55  68  73  52  64  48  38 106  52  64  35
74  85  55  74  44  46  86  52  64  56  38  73  43  56  64
54  94  42  64  47  74  74  42  64  54  45  74  42  64  46
57  83  34  37  74  47  66  63  47  45  35  64  63  47  35
65  64  44  36  45  37  97  72  37  54  44  94  32  43  46
54  75  55  53  64  46  64  44  56  54  44  97  55  34  74
45  83  43  36  65  48  75  55  53  68  54  84  33  67  54
46  86  52  53  65  56  67  42  44  57  54  74  64  53  36
44  66  36  37  77  46  86  52  35  38  37  74  43  43  37
64  74  64  53  37  58  73  63  55  35  55  66  66  53  37
38  93  33  44  46  55  64  66  35  54  48  75  33  36  45
45  64  34  43  37  46  83  52  64  46  44  97  52  37  77
75  73  44  56  54  37  65  55  34  46  57  83  66  36  65
48  84  33  67  64  37  74  33  67  46  35  84  34  34  38
35  94  75  66  74  44  75  52  36  66  37  97  44  54  68
35  93  63  36  46  44  63  52  44  35  36  73  52  45  77
44 104  35  44  55  35  97  44  73  65  37  75  52  53  65
35 103  54  56  46  35  93  35  57  54  45  64  62  53  37
36  96  72  36  44  65  75  35  64  36  54  74  35  63  35
65  85  44  56  54  64  74  33  34  65  37  84  44  53  68
44 104  52  64  46  35 103  44  36  65  48 106  33  73  68
64  64  44  56  54  68  66  63  47  44  75  83  66  53  64
74  65  55  63  35  68  83  42  64  68  74  74  43  43  37
65  74  33  35  44  54  75  75  45  57  37  94  42  44  74
45 103  35  37  75  44  75  55  34  74  68  65  33  73  65
46  97  75  63  54  65  75  55  53  68  54  73  53  34  74
65  77  54  67  54  37  75  35  47  34  37  94  44  36  56
54  84  66  34  64  44  75  35  64  48  45  86  35  37  38
47  83  54  37  37  48  84  33  67  77  35 103  44  34  48
35  75  55  53  45  37  93  52  76  35  74 104  42  37  38
57  66  32  53  35  65  83  32  53  68  77  85  66  53  37
46  66  46  33  37  65  75  35  55  54  46  86  35  45  77
35 103  73  43  38  38  76  52  36  47  38  83  44  34  45
66  83  35  57  68  74  74  43  43  37  65  84  36  73  54
65  75  36  76  44  65  66  43  56  35  68  75  44  43  64
54
```

Unit 116
Attack on the Nihilist substitution cipher

To break a ciphertext encrypted with the Nihilist substitution cipher, our first task is to determine the period. To do so, we will try to guess the period *m*, divide the text into *m* slices or columns, and check whether there are more or less than 25 distinct numbers in each slice. If there are more, then we know that we have not guessed correctly. If there are less or equal to 25 distinct numbers in each slice, then we may have found the correct period. We should also check that there are no more than five different digits in the one's place and no more than five different digits in the ten's place in each slice (or six if the one's place of any number has a zero, indicating a carry into the ten's place). If we satisfy this criterion, then we can make check further if we wish by replacing the numbers with letters, using a different substitution key for each slice, and combining the slices to form a temporary text. Then we can graph the index of coincidence for various choices of dividing this new text with a new period, as we did in Unit 31. The peaks at multiples of the true period will be at a value like that of typical English text, but the valleys will be shallower than they were when we analyzed polyalphabetic ciphers. For an example, consider this ciphertext:

```
37 75 68 77 64 59 38 54 55 53 63 60 37 55 59 75 35 39 44 48
95 65 42 67 56 65 58 83 42 29 47 57 65 56 35 47 56 44 89 75
36 69 66 58 58 67 56 40 66 48 85 43 64 40 34 76 67 65 35 50
56 44 85 55 64 56 64 46 86 65 32 56 65 66 76 65 56 48 34 74
58 74 45 29 65 47 59 55 53 69 56 75 89 64 54 26 68 65 87 45
52 47 65 54 67 53 32 26 37 48 77 67 75 37 38 66 65 57 54 60
55 47 55 54 42 36 65 78 76 53 65 28 38 77 87 43 42 60 66 64
77 83 42 29 58 68 89 64 42 48 64 77 87 47 66 29 65 78 69 46
44 60 34 47 86 56 66 58 34 54 89 57 64 30 68 65 89 64 42 60
55 54 87 47 54 36 34
```

Let's suppose that we guess that the period is 7. We divide the ciphertext into seven slices/columns:

37	75	68	77	64	59	38
54	55	53	63	60	37	55
59	75	35	39	44	48	95
65	42	67	56	65	58	83
42	29	47	57	65	56	35
47	56	44	89	75	36	69
66	58	58	67	56	40	66
48	85	43	64	40	34	76
67	65	35	50	56	44	85
55	64	56	64	46	86	65
32	56	65	66	76	65	56
48	34	74	58	74	45	29
65	47	59	55	53	69	56

75	89	64	54	26	68	65
87	45	52	47	65	54	67
53	32	26	37	48	77	67
75	37	38	66	65	57	54
60	55	47	55	54	42	36
65	78	76	53	65	28	38
77	87	43	42	60	66	64
77	83	42	29	58	68	89
64	42	48	64	77	87	47
66	29	65	78	69	46	44
60	34	47	86	56	66	58
34	54	89	57	64	30	68
65	89	64	42	60	55	54
87	47	54	36	34		

Take a look at the first column. It has nine different digits in the one's place; therefore, 7 is the wrong period. Suppose we try period 6:

37	75	68	77	64	59
38	54	55	53	63	60
37	55	59	75	35	39
44	48	95	65	42	67
56	65	58	83	42	29
47	57	65	56	35	47
56	44	89	75	36	69
66	58	58	67	56	40
66	48	85	43	64	40
34	76	67	65	35	50
56	44	85	55	64	56
64	46	86	65	32	56
65	66	76	65	56	48
34	74	58	74	45	29
65	47	59	55	53	69
56	75	89	64	54	26
68	65	87	45	52	47
65	54	67	53	32	26
37	48	77	67	75	37
38	66	65	57	54	60
55	47	55	54	42	36
65	78	76	53	65	28
38	77	87	43	42	60
66	64	77	83	42	29
58	68	89	64	42	48
64	77	87	47	66	29
65	78	69	46	44	60
34	47	86	56	66	58
34	54	89	57	64	30
68	65	89	64	42	60
55	54	87	47	54	36

34

Now if we look at each column, there are five or fewer distinct digits in the one's place and six or fewer in the ten's place (to allow for possible carry digits). For example, the first column has 4, 5, 6, 7, 8 in the one's place and 3, 4, 5, 6 in the ten's place. We can be confident with a ciphertext of this length that this criterion gives us the correct period.

The remainder of the cryptanalysis resembles the two-stage attack we built against the quagmire 1 cipher: we find a subtrahend (something to subtract) for each slice/column, subtract it, put the pieces back together, and solve the remaining monoalphabetic substitution. Each subtrahend must leave a column with only the digits 1, 2, 3, 4, 5. For our example, the only possibility for the first column is 23. For the other columns, 33, 44, 32, 21, and 15. After subtracting, we have

14	42	24	45	43	44
15	21	11	21	42	45
14	22	15	43	14	24
21	15	51	33	21	52
33	32	14	51	21	14
24	24	21	24	14	32
33	11	45	43	15	54
43	25	14	35	35	25
43	15	41	11	43	25
11	43	23	33	14	35
33	11	41	23	43	41
41	13	42	33	11	41
42	33	32	33	35	33
11	41	14	42	24	14
42	14	15	23	32	54
33	42	45	32	33	11
45	32	43	13	31	32
42	21	23	21	11	11
14	15	33	35	54	22
15	33	21	25	33	45
32	14	11	22	21	21
42	45	32	21	44	13
15	44	43	11	21	45
43	31	33	51	21	14
35	35	45	32	21	33
41	44	43	15	45	14
42	45	25	14	23	45
11	14	42	24	45	43
11	21	45	25	43	15
45	32	45	32	21	45
32	21	43	15	33	21
11					

We next replace each number with its corresponding letter in a Polybius square with an unmixed alphabet (without J, of course). We have:

```
DRIUSTEFAFRUDGESDIFEVNFWNMDVFDIIFIDMNAUSEYSKDPPKSEQASKASHND
PNAQHSQQCRNAQRNMNPNAQDRIDRDEHMYNRUMNAUMSCLMRFHFAADENPYGENFK
NUMDAGFFRUMFTCETSAFUSLNVFDPPUMFNQTSEUDRUKDHUADRIUSAFUKSEUMU
MFUMFSENFA
```

If we apply the hill-climbing attack from Unit 36 to this text, we get the plaintext

```
ANDTOPRESENTABROADERVIEWIHAVEADDEDAHISTORYOFALLFORMSOFSOCIA
LISMCOMMUNISMNIHILISMANDANARCHYINTHISTHOUGHNECESSARILYBRIEF
ITHASBEENTHEPURPOSETOGIVEALLTHEIMPORTANTFACTSANDTOSETFORTHT
HETHEORIES
```

(from *Anarchy and Anarchists* by Michael J. Schaack) and the substitution key

DGHIFKLMNJOPQRSTBEAUCVWXYZ

But bear in mind that this is the *inverse* of the mixed alphabet that belongs in the Polybius square, and that J is not allowed. Once we invert this key, we have

SQUAREBCDFGHIKLMNOPTVWXYZ

so the keyword is SQUARE and the square contains

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

From this square and the subtrahends above, we find that the other keyword is CIPHER.

Reading and references

Wikipedia, en.wikipedia.org/wiki/Nihilist_cipher

American Cryptogram Association,
www.cryptogram.org/downloads/aca.info/ciphers/NihilistSubstitution.pdf

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, pages 619-621.

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; pages 164-168.

Merle E. Ohaver, “Solving Cipher Secrets,” *Flynn’s*, March 28 and June 27, 1925,

toebes.com/Flynns/Flynns-19250328.htm and toebes.com/Flynns/Flynns-19250627.htm

Programming tasks

1. Modify your two-stage attack on the quagmire 1 cipher to make an attack on the Nihilist substitution cipher, as explained in the text.

Exercises

1. Break this ciphertext with the two-stage attack.

```
34 80 57 87 47 63 47 25 88 56 78 76 44 58 24 60 65 57 45 34
86 44 58 95 75 63 44 86 25 67 57 57 45 36 57 43 77 86 87 47
34 89 27 56 65 77 66 33 50 24 66 86 58 43 65 50 36 77 65 77
64 65 56 36 60 64 64 77 57 67 55 66 78 75 63 54 69 44 88 64
65 67 36 57 47 67 55 67 63 76 67 47 89 74 75 75 66 66 27 90
68 74 67 35 59 25 88 86 65 44 54 69 66 68 55 75 45 33 67 56
76 65 86 55 35 48 47 89 74 56 73 67 47 53 60 86 56 76 37 60
44 96 56 65 66 56 79 43 58 75 64 73 37 47 56 67 78 87 43 44
59 56 88 65 78 46 66 50 55 58 87 54 47 34 79 43 89 74 56 76
53 48 27 57 75 56 75 37 46 34 79 77 87 63 37 78 25 97 74 58
84 53 48 56 76 56 55 53 37 49 25 57 65 87 45 37 47 24 67 57
75 56 44 69 64 76 56 87 63 35 47 55 77 78 67 45 34 48 46 99
77 65 55 37 47 44 80 68 75 67 66 66 25 77 78 87 45 34 48 55
89 86 58 43 53 80 33 67 78 75 76 76 50 24 68 58 75 75 66 48
24 60 88 86 66 76 78 56 57 75 94 64 57 60 23 60 55 78 47 66
50 24 77 56 87 86 53 57 63 58 56 78 46 35 57 63 60 55 56 46
37 47 53 57 56 87 45 57 49 25 59 87 58 64 43 76 24 60 94 56
77 63 50 47 89 74 56 45 75 67 55 89 75 78 57 37 47 26 58 55
58 43 65 50 36 77 56 87 86 54 79 44 88 65 84 66 44 67 47 80
65 55 44 44 79 44 96 56 95 63 37 78 25 77 78 87 45 34 48 55
89 77 75 45 65 67 47 89 74 56 53 37 56 25 80 87 58 77 65 59
43 67 55 65 56 66 48 24 60 54 87 63 35 46 34 69 87 86 84 53
67 36 76 75 87 44 35 69 34 89 56 86 53 67 59 43 60 54 75 76
54 78 47 60 95 54 47 34 79 43 58 54 75 44 65 79 56 77 64 56
57 54 86 25 80 87 58 76 53 48 53 90 66 77 64 46 67 43 67 94
56 46 34 57 64 80 88 84 47 57 79 43 58 55 56 56 37 47 26 88
58 54 76 53 48 36 67 86 56 53 44 49 25 77 78 67 47 67 47 56
68 88 87 53 37 47 25 58 86 84 45 46 67 34 79 77 97 77 63 50
47 89 74 56 44 35 76 27 57 87 86 53 44 49 25 89 58 64 45 36
80 24 77 78 68 76 53 48 53 57 58 68 44 35 78 55 60 54 87 63
35 67 47 96 56 86 76 54 60 34 89 75 58 67 45 89 56 76 56 64
54 57 89 26 58 87 56 56 66 67 63 58 86 95 63 37 87 25 57 56
95 47 34 68 44 80 68 88 67 36 48 24 66 97 57 64 34 48 36 89
75 58 67
```

Unit 117
Bifid cipher

The *bifid cipher* is one of Félix Delastelle's inventions. It uses a keyword to fill a Polybius square and a period to determine how the plaintext is divided into units that are enciphered together. The only way to adequately explain is through an example. Here is our short message:

```
THIS MESSAGE WAS ENCRYPTED WITH A GRID CIPHER
```

We will use the keyword `POLYBIUS` and a period of seven. In the simplest way, we can fill the square thusly:

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

Next, we divide the plaintext into blocks of length equal to the period seven. The last block is short, but that's OK. We can encipher it in the same way as a full block.

```
THISMES SAGEWAS ENCRYPT EDWITHA GRIDCIP HER
```

Now let's encipher the first block. We write below each letter the row and column labels that address that letter in the square:

```
T H I S M E S
4 2 1 1 3 2 1
0 4 0 2 1 1 2
```

Next, read off the coordinates from the upper row and follow it with the lower row. We divide it into pairs, and remap those pairs back into letters by using the same square.

```
42 11 32 10 40 21 12
 W  U  N  I  T  E  S
```

The full ciphertext is

```
WUNITESUFVSQSNGAPVHXPFVWULPXGSUYTBPFRC
```

If the period is specified as zero, then the convention is that the entire plaintext in enciphered as one block.

The breaking of letters into smaller parts (in this case two base-5 digits) and separating the parts of each letter from each other is called *fractionation*. We will see this again.

Reading and references

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; pages 210-211.

Félix-Marie Delastelle, *Traité Élémentaire de Cryptographie*. Paris: Gauthier-Villars, 1902, archive.org/details/8VSUP3207b

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, page 243.

American Cryptogram Association, www.cryptogram.org/downloads/aca.info/ciphers/Bifid.pdf

Wikipedia, en.wikipedia.org/wiki/Bifid_cipher

Practical Cryptography, practicalcryptography.com/ciphers/bifid-cipher

Programming tasks

1. Implement the bifid cipher. Remember that there are many ways to mix an alphabet from a keyword and to lay it into a Polybius square.

2. Implement a dictionary attack.

Exercises

1. Encipher this text with keyword `SOCIETY` and period five. Fill the grid in the simplest way, as we did in the example above.

   > How self-contradictory, in the first place, is the nature of man! How sociable he is! also how unsociable! We have among animals the gregarious and the solitary. But man is of all animals at once the most gregarious and the most solitary.
   >
   > (from *Modern Society* by Julia Ward Howe)

2. Encipher the text in Exercise 1 with the same keyword but with period zero.

3. Decipher this ciphertext with keyword `ROBERT` and period 6.

```
RRHIKFERHMKSUFBPEVBETUFWEAOALEQERYMKHBTSLCRBANVTTRKNNPVFBEH
ALCBFNVRSDNNTTFNPETERHMKSPPENBMRHBKSLTLGCAWFNBETVODKSTFVOAU
RLIVRVQUHQHEUGLAREWZHHPEARHRCYLASRHYUEHAOAVNORAZETAOOVNZOPC
RBNBRPVNUOPCRBNBRWVNAPBCNLOLKBLWVFDGRMERBBLVPBSEAEAPRBUGVBE
DPRTROFQPTNEASAFCECTSTOISAVRDOHTQCLICRRHNNNPBZRRHIKFERHMKSP
INQDTABLYVFPHNRYSAPICBRPABBNGFVORQDXORNUGHKLGOYBAOSQPFNHTSE
WPCANNHSNWYFEEEA
```

4. Decipher this ciphertext with keyword `SAMUEL` and period zero.

```
FQAVKKIOQSEUWUKSURZICUSQINORBSRLYSILSPYUSSLRUYUSSLRGRSUAVKU
EOGCHOHATFDMQAWFBAERKQUSVRBILMZCVOTSQZUDTIYMVYUTNHYBYUTNHYW
FKZPOZYTOPDNYFN
```

5. Break this ciphertext with a dictionary attack.

```
REKTAXKSIIZVNCOGEOOMEKGUFQUKXMSPYBTBBRVYKGSOMSVHTOLDXPAYODC
NEHVDTCVYAKHPSBWXVKBBOVYICWOLVITUHTWEFTVPUUBDUPPLRMFYUYCKUH
VWWCQNETERWOZANOOPPTLDQTPOFCHSRCCTAHRCIKZVNGWCWNEHUCQXQOWQK
QNFYFDGVSVDBLSEEZHBONAIQWFBYUQYMDRLRZNYHEEIEQXVMHTCEFTVPDWB
FNDHPWVFEWSWLGNQEPAATVHROHCKOQSQHITFRIELFOUAPUTFABREUCOSBQL
ELFZEEKBSRTITEMWZVIACVEVKNSKMPIPXBDCQGONFSHNOVFTVYPMGYUEPMQ
BMXLDCQGONBSHHZVFCGSZAK
```

Unit 118
Attack on the bifid cipher

If we are given a ciphertext and want to break it, the first thing we need to do is find its period. One approach to this question is to graph the index of coincidence as a function of the period in the same way as we did when examining the polyalphabetic substitution cipher in Unit 39. Here are three examples from real ciphertexts that have periods four, twelve, and zero. As you can see, the peaks are not as large as they were in our analysis of the polyalphabetic substitution, and it is not always easy to find the correct period in the graph.

Once the period is known (or guessed), we can apply a hill-climbing attack that strongly resembles the one we built for the Playfair cipher in Unit 106. We need to change the decryptor function, of course, but the rest of the algorithm remains unchanged.

Reading and references

Practical Cryptography, practicalcryptography.com/cryptanalysis/stochastic-searching/cryptanalysis-bifid-cipher

António Machiavelo and Rogério Reis, “Automated ciphertext-only cryptanalysis of the bifid cipher,” Universidade do Porto technical report DCC-2006-1, www.dcc.fc.up.pt/~nam/publica/dcc-2006-01.pdf

Programming tasks

1. Write a function to take a ciphertext and try to find the period. It will be similar to your function for the polyalphabetic substitution, but you will need a new way to detect a peak.

2. Implement the hill-climbing attack by modifying a copy of your attack on the Playfair cipher.

Exercises

1. Find the period of this ciphertext. Then break it with your hill-climbing attack. What is the keyword?

```
GRRAVYGIOLRGTSCYNRTYWYHVYGUMLRRENOUVRVEERIYITRLVOACOTDEYYGN
MTVBECVEERSYWIRFRONCUCPARIYIVDNVCBKXGADXETDRYGOEGTPGYFRRMOQ
CUSQBYDFVTGFMRROMCSDWEAIBGWTHFYAVBYCHTBAEXCUMROOACIDIVIYIVV
OGVRHRQADGGQRYOHAQXMEYNDDYSPQYYEYPVYGUMDGRKYLDGIYIVIYIYIAIU
VRMEOGZQPOEZDDUBYCAROAKOEGIMNGTMSSUIDLEGTYTVEDVUIDIYXVKRFGN
MRAAHTEFBNBEUCHAWCFMWAAQYPAXZOOGCOWVGVUCTGAMGDQGYNDUYYPPRAN
MWAFSHHYVUXOIVRMLYERNGHNVBPCROUTRIPDRYASNYMNUUCGOZUIHIGQZVS
CFKPBEUQYAPOARGGOACPCXFFTBRSOYCDURROTKSLKRBBAZYQRRIABVLYPUY
CAFOOITICFFYOYEYLRRALPORWIHLYGUMNGTMRAAHTKFYAAHXFOHZAOKVOGC
AIYIVADXECRREPFYRDHZGRDEUICIFLFERPXETATSYGEMFPDWZDYYNPBDOVU
CTGIGIEDWERAEDBNTAIYIWSCYYAIUIAVUMNQTYSTNFBDWBNGHNLCREPAEXC
RMRNACVOYCYGNOUIAIURGOZCYRAKYGVUIHICTASQRZOTYTVAGXMRDEYYGNM
RYEDCQAZIDIZTARVM
```

Unit 119
CM bifid cipher

Suppose that we start with a bifid cipher and use a different grid to read out the ciphertext characters from the one used to read in the plaintext characters. This cipher is the *CM bifid cipher* (or *conjugated matrix bifid*). Let's rework our example from Unit 117 by adding a second square to see how this works. Start with the same plaintext:

```
THIS MESSAGE WAS ENCRYPTED WITH A GRID CIPHER
```

Use the keywords `POLYBIUS` and `SQUARE` to fill two grids:

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

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

The period remains as seven. Divide the plaintext into blocks:

```
THISMES SAGEWAS ENCRYPT EDWITHA GRIDCIP HER
```

Now let's reencipher the first block. We write below each letter the row and column labels that address that letter in the first square:

```
T H I S M E S
4 2 1 1 3 2 1
0 4 0 2 1 1 2
```

Next, read off the coordinates from the upper row and follow it with the lower row. Divide it into pairs, and remap those pairs back into letters by using the second square:

```
42 11 32 10 40 21 12
 N  Z  O  Q  R  Y  T
```

The full ciphertext is

```
NZOQRYTZPWTITOKLSWDHSPWNZVSHKTZMRGSPCF
```

An astute reader may notice that this ciphertext differs from the original by a simple monoalphabetic substitution. If we think of the bifid cipher as a compound cipher, consisting of (in this

order) an (inverse) monoalphabetic substitution, a polybius cipher, a columnar transposition (two columns), an inverse polybius cipher, and a final substitution cipher, then the CM bifid differs from the bifid cipher only in the final substitution. Whereas the bifid uses two substitutions that are inverses of each other, the CM bifid uses two different substitutions.

A hill-climbing attack on the CM bifid could use the same technique as used for the bifid. However, rather than maximizing textual fitness, we could maximize the index of coincidence. A large IoC, near 1.7, means that we are only a monoalphabetic substitution away from the solution. The second stage of the attack is to break the remaining substitution with the technique from Unit 36.

Reading and references

American Cryptogram Association, www.cryptogram.org/downloads/aca.info/ciphers/CMBifid.pdf

Programming tasks

1. Implement the CM bifid cipher.

2. Implement the two-stage hill-climbing attack on the CM bifid cipher.

Exercises

1. Break this ciphertext which was encrypted with the CM bifid cipher. Recover both keywords.

```
DQVAGNHSFVAPSSLLEOTUYWVGUAGOVIYAYRXTAZAKFQUPUCUKSDUZGGWVDTU
PLEXANTZHAZPSIDIESTVDZAPDUVDRWOLTCISPVYXVFASGXAFTWPXLYNDQNA
GSXDRBULTOQVKUNYDUMPGRMYSXIYBRLUSVQPUAAVDXCTZMVWVEHTSTSYCNX
WELRLAUPCGYAVTVSGMTDLOSCGWSISYYKBPOTTYOEAYZCBXSWEADEQPUUVWR
RMMDRLALVYOBTTPNIGLRSTUUTXKWIKRZTUTBEDRZSGYZZRVZOOOTWDVMAGS
DSUHICKTXZULTBTYSNLDMSLRLHGKWMFULARVDSUPLYTOISYHTRMREYSMIZY
ZLWKPQTXSAANFZRIURDDQTLLWALOEZYRGRHVLUNSAXLLPQDRWAFRYDUAGGR
UWPOUIITIVZRHOIYXHIYKRZUZTBUGTASOXGELQNRSRDANSCCLSAIQUHPTUZ
UGUFITCCVPVTSRWEGCSXPIWTXLPNWIELZVZUOIGEUBVUVNWGMIAOEDYEGDH
VOZTTBOSQMECICSUGCNVPXYDTPGVDAHTUYQLRSIVYHEWSVLICHZBTVVNAUR
NRAVNRNDSKADBTDANACLURWNRYTEVCAUPYAWZLLIVLWLHIEFWUAZLQYXMSI
KIULUZQGUCAYVQOGUVVEBPLLGFYVPXLZSWOMZUMSQRICXEYWURYIVPNKOYM
UVCLGOAYDEGLNDVXPGFUTGTALKNYPTWIELLAACNHPZTNZUXYNTUUNAUAKVL
UUCLFWHIUTITLNPSOAWALUEZSQTYCNLSLWXCONIYAZIXGDRAVGKGDTWPSLI
FTWTUOVIVUTUKAFZITUUSYNYMMTZTVVHEKDDRRVSSYTTYAWIXLSINIDRTWU
FONVKVAZUCGDTQSGSTZTFLGSRAAMQYDUTDWANLETAZLXGQQTPENIFLOIYYH
ODTDTLAGLACPRGYDMNZLNLIKGGYYPVNTWHNNATVXCLQAATTXCLTFZNIUULE
IOUQINTXHNNUKLQ
```

Unit 120
Trifid cipher

The *trifid cipher* is a generalization of the bifid cipher to three dimensions. Yes, Félix Delastelle invented it, too. Instead of the Polybius square, a mixed alphabet is placed in a 3×3×3 cube. Breaking the plaintext into blocks of equal length is the same, and period zero means the whole text is one block, but the fractionation is done with three base-3 coordinates. Since we have 27 spaces in the cube, we do not need to drop any letters, and need to add one. The new character can be space or some item of punctuation.

Here is an example. Suppose we want to encipher this message with keyword `KEYWORD` and period eleven.

```
THIS MESSAGE WAS ENCRYPTED WITH A GRID CIPHER
```

The mixed alphabet can be `KEYWORDABCFGHIJLMNPQSTUVXZ_`, and we can put it in a cube so:

```
     0         1         2
   0 1 2     0 1 2     0 1 2
0  K E Y     C F G     P Q S
1  W O R     H I J     T U V
2  D A B     L M N     X Z _
```

The coordinates of a letter are the layer number, the row number, and the column number. So `G` has coordinates 1, 0, 2. We divide the plaintext into eleven-letter blocks, and write the coordinates under each letter.

```
THISMESSAGE WASENCRYPTE DWITHAGRIDC IPHER
21121022010 00201100220 00121010101 12100
11102000200 12002010010 21111201120 10101
00121122121 01212022001 00100122100 10012
```

The coordinates under the first block are read out by rows and broken into sets of three. Each triplet is remapped to a letter in the cube.

```
211 210 220 101 110 200 020 000 121 122 121
 U   T   X   F   H   P   D   K   M   N   M
```

The full ciphertext is

```
UTXFHPDKMNMYOYQPQEEVBEETFRILJKCNCMEWHR
```

Reading and references

Félix-Marie Delastelle, *Traité Élémentaire de Cryptographie*. Paris: Gauthier-Villars, 1902, archive.org/details/8VSUP3207b

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; pages 210-211.

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, page 243.

American Cryptogram Association, www.cryptogram.org/downloads/aca.info/ciphers/Trifid.pdf

Wikipedia, en.wikipedia.org/wiki/Trifid_cipher

Practical Cryptography, practicalcryptography.com/ciphers/trifid-cipher

Programming tasks

1. Implement the cipher.

2. Write a function to take a ciphertext and try to find the period. It can be the same as the function you wrote for the bifid cipher.

3. Implement a dictionary attack. There are many ways to place a mixed alphabet into the cube; just use the simplest one.

4. Implement a hill-climbing attack. The modifications to the cube that you need to consider are swapping two element, swapping two planes in any of three directions, and flipping in any of three directions (up-down, right-left, front-back). Feel free to make a modified copy of your attack on the bifid cipher. For the fitness function, you may want to strip out any of the 27th character in plaintext. The margin of error for stepping downward should be variable, and about 5% of the fitness.

Exercises

1. Encipher this text with keyword `FAIRY` and period 8.

   > The children who read fairy books, or have fairy books read to them, do not read prefaces, and the parents, aunts, uncles, and cousins, who give fairy books to their daughters, nieces, and cousines, leave prefaces unread. For whom, then, are prefaces written?
   >
   > (from *The Orange Fairy Book* by Andrew Lang)

2. Encipher the text from Exercise 1 with period zero.

3. Decipher this text with keyword `VIRTUE` and period 5.

```
TPHVXTTZXYSK_FAAGJVRIIDN_VRDVEVZJFPDFBIPIN_AYII_RIKEVWBYKKV
PDPFGDIONZITBSCOFRMQOHXIAFVDHCXSV_IQTBHLSVMEQDQIUWAHXIAFVDH
CXWTPPAAWDSGSBHDSVMEQDVRAKRUCOGTDTUEZBDALKRR
```

4. Decipher this text with keyword `CIRCLE` and period zero.

```
VEDGTIACIGQLVCCCYOAJGCRPCNVRLTDLYOAJPCFGBJVJMGCORGBJLLSDCTB
EWDOAHMZLQCKMGIGMLTSGIJICCIGKNIGMLTUINCEIGEC_QHKSEIHQTJCKEE
GVEOIFPHURNTSAIMPYZMAIHNHMWMJWNMPYZHKPNMMEPKMJIZNMMFOXWPMK
```

5. Find the period (not so easy) of this ciphertext and break it with a dictionary attack.

```
NMINDWATDAOAEAPYHH_IMALEWTJUPNP_QUKFNUPAPGNERKEIK_KPEXGWFLV
_SGTHHN_CRGYGGZPMDPYKGDTBIUIMAGHNNJOFHVEEDIMKPNHNN_CQTABYWQ
UJWMIWHJPTNVTCQGFPGDJCAQADEKLQIYFXLWFHPNLCGBLPJJCENIRJPLVP_
HPADLPGEFHMNBHANZLQUSIPCGIEAHCBKMPTNVTCQGFSGPJCHAUKAGTGPNIE
OVNIZBPIEQDDDVLCGADOOWLCDRSKDZNACEYHQWGNYTDBTZISEOLJIRR_LXG
BQRIZUPPDWASMZXPPEBEBZJAXIHUPXPSHSBABNZFERLIJHCENBFJPLYGTYX
YVOIZBYFPLNDCBTIMTUSKSLGWONVAPTMGEFFUKBICSIHIFVPPRCHUWXBKHI
EOOYQNISHS_JUKJPRDPWVPYHPEDOIWLUUKEXASDUETZJPXINKJIBPBHIFVG
PXCKAFIIPUXDHEYABHNSKADTMPCQAYEJSPAPJ
```

6. Find the period of this ciphertext and break it with a hill-climbing attack. What is the keyword?

```
RUPHNRDTHBYKTTRTQUXCAKYINADHLHT_BXOQRLWZNBZHZOMSVAFYWRSDFYE
IEIRNHSSUGTXNMNAFHMQTETDBTYPMHDIJYKZHBKEQOFMYJ_THYUGQEXUAG_
WYYUPYCANTFTFDF_DSKATHOPDNRKMPRICTOOFHIPAIRUIAYIOYPFOQZ_S_R
GFPEVUCSYAHAXWXPRNGMHQBKVACI_DBDCBJQATTCUTODHIKMGQHQPQXAHA_
TMEUJOIHNKT_HBYKTPRQRYYLJPEKEOUSZHSWDOTIDMMDLAFTUXI_RFFPLCU
TLPMDNNWCARHVMMAERURHMPDVVGHASDZRRVUKHDJDJOJQOVTSBMPFNUYHAZ
FKH_Z_EIASHXSSSAUGSXPSFIERENHZWYYYYXUDSAQXX_X_FEEDQDIQQOBMD
ENPODWJHHNJUTCOANXTAZIVPIT_YDJTCNTPGYAXHZUSQRHVMMAERGBOIPIL
CDQPDPNXEQOQAHK_CKNHRBTZYWPYPKNMCFYHNJM_DEPFSAKSCTCQAPXTHHR
RVP_INRXPRH_SSVFYHDNMDBGHQ_RSKKJHINI_AMQMAPDKLGOAYYWZJYKPFK
_FQADUGIETADUGFBAXHQLXQY_TUEFIH_RLGEBIFQFLBHWNFLOXYSTLHUXIC
FVISOJKITNYYZTTYBHHSUQDRREBH__NQQFVKRPDPYBPQFKTCIJIPHMWUTCY
HDETFJQNOUBIPARJVOOTGACPKPUEBPHSHSRTOQBKXZZXSYKLKDAPEYWYGRT
EISIYMRSIILBAOQSYLUDYIYWMTBCJVDOAHTSXOKUDCCYHY_WDQEOXSRXTVY
IIEH__TDBQHMTDSHDPVKWAAOWVQORJKKOHDIIXPOCQCXTRDYQPFNIKKPARJ
TUOFOCRLUXESZXHADINCGYAZQZSBRHEPLWISCAYYKRQNNPHPNQXEOFSIHFJ
VUOETL
```

Unit 121
ADFGX cipher

The *ADFGX cipher* is a Polybius cipher with row and column labels `A`, `D`, `F`, `G`, `X` followed by a columnar transposition which may be keyed with a keyword. Decipherment must be done in reverse order. The choice of the labels `A`, `D`, `F`, `G`, `X` is due to their low likelihood of being mistaken when transmitted in Morse code.

Some use a variation in which the transposition stage is a permutation cipher rather than a columnar transposition.

One way to attack the ADFGX cipher is to modify the hill-climbing attack on the columnar transposition cipher. However, since we do not know the contents of the Polybius square in advance, we cannot use tetragram fitness. Therefore, we use the index of coincidence as the function that we wish to maximize. For each permutation that we try, we decipher the ciphertext with a grid containing `ABCDEFGHIKLMNOPQRSTUVWXYZ` and evaluate the IoC. If we can maximize the IoC at a value resembling the IoC of English, then what remains is a monoalphabetic substitution, and we can use the attack from Unit 36. For permutations of length greater than five, there is a complication: There may be more than one permutation that gives the same maximum IoC. In that case, we will have to try several candidate permutations until we find the right one.

Reading and references

Practical Cryptography, practicalcryptography.com/ciphers/adfgx-cipher

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, pages 339-344.

Programming tasks

1. Implement the cipher.

2. Implement the attack described above.

Exercises

1. Encipher this text with the keywords `PINK` for the square and `FLOYD` for the transposition.

   It's been three months of lockdown. Please, someone send vegan brownies to Old Pink, in care of the Funny Farm, England.

2. Decipher this text with the keywords `FRANCE` for the square and `PRUSSIA` for the transposition.

```
AFAAGAGGFGADFXFFGFAGFDAAFFFGAGAAXFXAFDXFAAAGFGAFGDXGFFFXFAA
GAGXXGAAXAAAFDXGXFFFGXXDDFGGFFXFDAAGFGDGXXFDADADAADGAGDGADA
DADAGGDDFAGGAAGDGDXDDADDDDFDDDDFXDGFGDAXAFFGGADFXADDDAGFFAA
GAFGDGDAFDGADAAAGXAAAGAGGGFGDDAFAAFDGDFDADFFDAFADAGDFFDFDFD
DXAGFFAAGGXAADFAADDFFAGDFFAFGGXAGGXFGFXDGXGGFGGAADXDDGAFXAD
GAAGXFFFDGGAAXDXXFFAFGXAGDGGFAGDDFFGXAAFFAGXAAAXXDFFGAFXDGX
XXAAAAXFGGDDAGAXFGAXDXAGFGAAFGFGXFFAGADFADDGAAFDFGXXGFAFDFG
GDXGDDFFDAFDADAAAGFDFDFDDFAGADDFGFGAADFXAFGGXXDGGDGGFDD
```

3. Break this ciphertext. What are the keywords?

```
DXGAGFDAAAXAAAADAFADAAGFDADGAAAGFAGADAGFDGAGFGAGAAADAAGDAGA
DADFFADDDDAXDAFFGDDDXADAAGAGFXDADXFDAADAGDAFAAFADAGFFFDADAD
AGXAADDFAAAAGXGFDGDGGDAFDFXXDDGDFFDDDAFFDAGAAGDXADAAXDADFDX
DADGGFDFDAGGXGDGXGDDGXGDDAGDFFXFFDFXGDGDFDXGDFFDAFGAFGFFXGD
GDFFXGXFXFFFFFGXFDGXFFXGDDFFAXAGGXDXDDGGFGGAGGADAXDGDXFDFFF
DDGDAXFDXGXGDFXGGDFDGDXFFXGXXDGGDDXGADFFDAGFXDGFDAXXDXDAGFF
GXADGFGXXDFDGXDGFFDXGDFFFXFFGAFDXXDGGGFADFFXFFDGFGXDGXXXFGD
GGDDFGAGDFFFDDAXDFFAGDAFXADGAGDXADGXFGGXDAXFFFXGAXGGDGGFGXX
XGADGGFAAGAGFDGGGDAXXXDDGXAGDDGDXXAGDFGGFGFDGXXXFDXAFGFFDDX
FFFDGFGGGXADFFXFXXGGGDXXFDGDDGFGFGXXFAAADDADFAXXGDADDAAGDDD
XAAGAFDAAADXGAGDADFDXAAADDFFGGAXDDDAGFDXDXXAGXFFAAGAGDADDAG
ADADDDDDAAGFFADGFAAFAFDDDGFDDDDAAGADAGDDAGFDFADDXADDAAAFDAA
GAADDAAGAGDFGAAFDAADXGDGDAGAFFDAAADDAFXFGAAAAAGFDADDXDXFXXF
GDXDXFXDFAGXDXFFDDXXFGFGXGXGFXADGXGXXXDXDAXFDAGXFFXFXGDFDDA
XAFGADFGDGDFGDXDXDGDFFGDAGDFFGFGGDXFDXFXDADGDFXXFFXDAXFFGGG
FGXDDGDFFFDDFXDDADGDDGXGDXAFXXGFDFGGGDFFXXDGFDDGAFFFDDXGXFF
XADAGADAFXDFDDAAADFDAGADDADXXGAAFXDDADDADFDDDAGDFGFAAADGAAD
DFAADADDDADGFDAAAADADAAGXDGDAADAAFAXDXAADDDGAAAGDDDDGFGDDFA
ADXAAAADAADDXDAGAGADDAGDFAFGAAFFGXAAAAGXAAXGGXXGAAGGXDAAFFG
ADDDDDAAXADXAAA
```

4. Break this ciphertext. What are the keywords?

```
AAAFXAFFGAAXDDGGAAAXDGGFGDAAGADAGGDFDDFGAFGDGADDDGAFXGDAFXD
AXDFADADAFFGGDFGAGFGADFAFFAXAFAXFAXFAGAXDAFFXXAGXGFDFGDDDXA
XXAXAXFAFXFXXGDXXAFFADFFFFAAFAXADXAXGGGAFXDFFDFDDGGGDADDGGA
DAAADFFGGXAAAFGADFDXGDAGDADGFFFGAAGDGDFDDFDXFGAGDADDGDGFDDA
AAAGFFAGAFGDFGAXFGDDAGGFDAFDGDDAFADDGGAXFGDGGDXAADDGFFGDAGA
GGAAGDAFFDGFGADDAGGAFGDDDFGAAGGAGDFXGDFFDAXFDXAGFAXAXAXDAFX
```

XADXDAXFADGXADDADGAGFXDXXXXAXFXAXFFXFAGDXADGAAGFGAXXXGDGADG
FAAGFADXGAXAGFDFAAFAFXAADFGXAXGFXFDADGAAXDXFGXAADGGAGGDXFXA
XFAXFADXFXXXDADGDFXDAGGGDDDXGAGAFGGDDGAXAXAAFGFAGDXADGGDXDF
DXGXDFAGAXFDGFXDADXXFDDAFAXXXDXAXFAGGGGFAFDDXDFGFAGDAFADDGX
XDFAAAADAGDAFGAFXDXGGFGGGFFDDADAAGFGGGAGDGAGFGFDAGFDADDADA

Unit 122
ADFGVX cipher

The *ADFGVX cipher* is the extension of the ADFGX cipher by using a 6×6 Polybius square. Because the square has 36 places, it holds the full English alphabet and all ten digits. The choice of the labels `A`, `D`, `F`, `G`, `V`, `X` is due to their low likelihood of being mistaken when transmitted in Morse code.

Some use a variation in which the transposition stage is a permutation cipher.

An approach to breaking the ADFGVX is to look for a columnar permutation that results in the fewest distinct digrams. We are looking for a decryption with mostly letters and few or none of the digits. After the transposition, the Polybius cipher is broken as usual, if possible.

Another approach to breaking the cipher is to extend the attack on the ADFGX to use 36-character alphabets and a 6×6 Polybius square.

Reading and references

Practical Cryptography, practicalcryptography.com/ciphers/adfgvx-cipher

General Solution for the ADFGVX Cipher System, Washington D.C.: U.S. Government Printing Office, 1934, www.nsa.gov/Portals/70/documents/news-features/declassified-documents/friedman-documents/publications/FOLDER_269/41784769082379.pdf, archive.org/details/41784769082379

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, pages 339-346.

Programming tasks

1. Implement the cipher.

2. Implement the first attack described above. The length of the permutation is an input.

3. Implement the second attack described above. The length of the permutation is an input. As a subproject, implement the attack from Unit 36 for an alphabet of 36 characters. In calculating the fitness, it is acceptable to delete digits from the text.

Exercises

1. Encipher this text with the keywords `WEATHER` for the square and `PICNIC` for the transposition (remove repeated letters in the keywords).

   To give a picnic party a fair chance of success, it must be almost impromptu: projected at twelve o'clock at night at the earliest, executed at twelve o'clock on the following day at the latest; and even then the odds are fearfully against it. The climate of England is not remarkable for knowing its own mind; nor is the weather "so fixed in its resolve" but that a bright August moon, suspended in a clear sky, may be lady-usher to a morn of fog, sleet, and drizzle.

   from "The Picnic Party" by Horace Smith

2. Decipher this text with the keywords `MUNCHKINS` for the square and `CYCLONE` for the transposition (remove repeated letters in the keywords).

```
DGDADVAADDDDDGGDFVDGDVFGDGDDFADDADGDDGDDDAGDADAGDDDDDDGADFF
GADGDDDGADAFFGDAAFAGGDAGAADAGDDFGDFDDADDGDAGFDGFGGADGFFDDAD
GFAAFDAFDAFGFGFFGAFADAAFDDGGGDGDDFDDFDAFDADGADXDFFGDGVVFVGX
FADVFVXGFFXAXVAFGXVFAFVFFVGGXDFFVXVVVXXFVXVGVFXDFVAXVAVDGVV
AFFFAVFVVXDFGDDDVXDXAFFAGXFVAVGVDXFXVVXFDVGFVVFFDXDFVFFVDXF
XDFFAFDXFGXFDXXADVDDDVDDFVVADVDDADGDDDFDADADGADFGAADADFAFDD
AADDAGDGGFADGGADDDGDGGDFADGFGDFDFFDFGAAADDDVGADFDGDAGADDFFA
FDGGFAGAAADAAGGDFFAADFFDDDFAGDDGFDDDDADDDDFADFDFDGFDAGDFDFA
ADDFADDADDGDAGFFDDGDDDADDAFDADGDDFGADFDFAADFDADGGDAGDDADDAA
ADGGAFDAAAFDAFFAADGDDAAFDDDDDFDFVGDADDDAFDDGGFGDDAGDDGADADD
ADFFDFDGDFAFAGADDGAGGGDGDFAGDGDGDGDDVDVDDFADADDVFGFFGDFGGGD
FDADDVFVXVFVDDXFXDVAVFVFGFAVVVGDAVFGFDDVFFAXDFVGFFVVVFVXFFF
VFDGAGVAVDGFVXAFADVFXVFXVFVDFVVFVFFDVVDVDVXAFFVVAVFVAGVDDFA
ADVVFVFXXDFVXVXVVAGADXDAFAFVVVVVVXDVFAXFAXFXFDGDDXVDVFXXAVX
DVFDXFFVVAVFXADFDFVXDVXXDFFDXDVXFVDXVVFDFFFXVVFDAVGFFXFXFFV
FVAGVFDDFVGFFXDVFAAFVAFAFGDDDFFAFXFVVAFGFDAXDDVXXFAFVVVGFFX
XVVFGFDGGGVAXFFVXGFDDXDAGDFGDFVVXDAFVF
```

3. Break this ciphertext. The transposition has length nine. What are the keywords?

```
GFAADFVAGDXFVAAFADGDDFAADAXAVDAADAGAAADAXDAVGAVFDAGFVAXDDDG
DXGDAGAFAVDXAFAAGXDFDADFAVFFADAAGVDADXAXFAAVADAVDXDDAFDDFXF
FADFVADAAGAAAGAFGVFGVDAAAAGDAAXDFAAADAAGVGAGAAADVFXDDFADGDA
AAAGGDDFAAAAFVDXDAGXGDAAAVGAAVADDXDXAAFVDGADDFGAGVFVDXAGDVD
AAGGXAAAAFVDGFDFGFADGADDAFFDGGXAAGFGGFDAAADFDAADGAAFDGGGXAF
DGAAAFFFDGADAGGDFDGXAFAFADAAAGAADXAVFXDGFVAGDDAVAXFAAXDXGDD
VGGGVFXDADDAFDVFDDVDGDDAGAXDVAGAAGVFAAVDVDDAVDAFGDFDXAAAAAA
ADAAAFADAGAFGADADDDXGDDXAAGXADAXAAFAAXGDAVDGGAAFAFFDAAADAXF
GFADXFDDAAGFXGDFDGFFAGGAAGXAAGXAVADDGDXFGGAADDGAGDAGDAXAVDA
GAADXDAAVGVAADDFVAGDADXGXDAAXDAFADAAAAAAXGFFDDXAVAVFADADDAA
ADAAAGAFDVGXFAFXGXFAGXAXADVADXFXAXAVDVGVFGAXGGGFFGAADXAVAAA
FDGGXDGVFDGDXDAAFAAFXFGAAAFAFDAAAAAFAAXAADGGXAXAAGVDAAADADA
DVADAXDFGXAGGFDVFGGDAXAADXAXAAAGAVAGDGFXAXAADFFAAVDADAGADAA
VAXFFDADAAAAXFAFVAFDXFADVDGDGAADXAA
```

Part VI
Matrices

Unit 123
Matrices and vectors

A *vector* is an ordered set (a list) of numbers. The length of that list is its *dimension*. The numbers in the list are the *components* of the vector. In this section of the book, vectors will be written as *column vectors*, i.e., as a column of numbers. The name of the vector is written either in bold face or with a vector arrow over it; for example:

$$\mathbf{A} = \vec{\mathrm{A}} = \begin{pmatrix} 2 \\ 3 \\ 5 \end{pmatrix}$$

The components are written with a subscript. For the above example, $A_0 = 2$, $A_1 = 3$, and $A_2 = 5$. Notice that we start counting with zero; this is not the usual convention, but we are programmers and we use languages that index lists beginning from zero.

The set of all vectors with the same dimension is called a *vector space*; its dimension is the same as the dimension of its vectors. The *origin* of the vector space is the vector all of whose components are zero; This vector is called the *zero vector*, **0**. In a vector space, we can add vectors by adding their components. If **C** = **A** + **B**, then

$$C_i = A_i + B_i$$

for all $i = 0, ..., d-1$, where d is the dimension of the vectors. Since it does matter in which order we add two numbers, it is also true that we can add vectors in either order; we say that addition is *commutative*. Of course, if we add the zero vector to any other vector, then that vector remains unchanged; for any vector **A**,

$$\mathbf{A} + \mathbf{0} = \mathbf{0} + \mathbf{A} = \mathbf{A}$$

A *scalar* is a number that is not part of a vector. We can multiply a vector **A** by a scalar by multiplying each component by that scalar. If **B** = c**A**, then

$$B_i = (cA)_i = c\,A_i$$

Recall from Unit 9 that we saw how to multiply vectors (it's not the only way, but it is the only way that we will need) by defining the inner product (scalar product, dot product) as

$$\mathbf{U}\cdot\mathbf{V} = \sum_{i=0}^{d-1} U_i V_i$$

An $m \times n$ *matrix* is an two-dimensional array of numbers with m rows and n columns. For example:

$$\mathbf{M} = \begin{pmatrix} 3 & 4 & 2 \\ 7 & 9 & 5 \end{pmatrix}$$

The numbers in a matrix **M** are its *elements* (or *entries*) M_{ij}, where i labels the row, and j the column, in which a component exists. For the example above, $M_{01} = 4$. The *dimension* of a matrix is the number of rows by the number of columns. For our example, the dimension is 2×3. The set of all matrices with the same dimension has algebraic properties similar to a vector space. We can add two matrices in the set by adding their elements:

$$(\mathbf{M} + \mathbf{N})_{ij} = M_{ij} + N_{ij}$$

The matrix all of whose entries are zero acts as the additive identity. We can also perform scalar multiplication:

$$(c\mathbf{M})_{ij} = c\, M_{ij}$$

Multiplying matrices is a little more complicated. We can only multiply two matrices **A** and **B** if the number of columns in **A** matches the number of rows in **B**. The rule is that if **C** = **AB**, and **A** has dimension $m \times n$ and **B** has dimension $n \times p$, then **C** has dimension $m \times p$, and

$$C_{ik} = (\mathbf{AB})_{ik} = \sum_{j=0}^{n-1} A_{ij} B_{jk}$$

for all $i = 0, ..., m-1$ and $j = 0, ..., p-1$. Notice that if m and p are different, then we can multiply **AB**, but not **BA**.

The set of all square matrices with the same dimension is special. A matrix is *square* if it has the same number of rows as columns. In this set, it is possible to multiply in either order. In general, however, the result will depend on the order, so matrix multiplication is noncommutative. In this set of square matrices, the additive identity is the matrix all of whose entries are zero. Adding it to any other matrix leaves the matrix unchanged. The multiplicative identity is the matrix **I** such that

$$I_{ij} = \begin{cases} 1 \text{ if } i = j \\ 0 \text{ if } i \neq j \end{cases}$$

All of the entries on its diagonal are one; all other entries are zero. The multiplicative *inverse* of a square matrix **A** is another matrix called $\mathbf{A}^{-1}$ such that

$$\mathbf{A}\,\mathbf{A}^{-1} = \mathbf{A}^{-1}\,\mathbf{A} = \mathbf{I}$$

To find the inverse of a matrix we must go on a short detour in the next two units.

Next we need to discuss the product of a matrix and a vector. We will only be concerned with square matrices here. A n-dimensional vector is the same thing as a $n\times 1$ matrix, so we can handle multiplication in the same way as we multiply matrices. The product of an $n\times n$ matrix **A** and an n-dimensional vector **V** is another n-dimensional vector **U** whose components are

$$U_i = \sum_{j=0}^{n-1} A_{ij} V_j$$

Reading and references

Wikipedia:
en.wikipedia.org/wiki/Matrix_(mathematics)
en.wikipedia.org/wiki/Square_matrix

Programming tasks

1. Go back and find your function that evaluated the dot product of two vectors.

2. Create a Python class for vectors. Override the addition operator to add two vectors. Override the multiplication operator to find the dot product if the two operands are vectors, or to multiply by a scalar if one of them is an integer or floating-point number.

3. Create a Python class for matrices. Override the addition operator to add two matrices only if they have the same numbers of rows and columns. Override the multiplication operator to do one of these things, based on the type of the operands: multiply matrices (check that the numbers of rows and columns allow it), multiply a matrix by a vector and return a vector (again check the sizes), multiply a matrix by a scalar.

Exercises

1. Find these products:

   a. $\begin{pmatrix} 6 & 6 & 7 \\ 4 & 9 & 7 \\ 1 & 5 & 5 \end{pmatrix}\begin{pmatrix} 6 & 6 & 4 \\ 5 & 0 & 8 \\ 9 & 7 & 6 \end{pmatrix}$

   b. $\begin{pmatrix} 4 & 3 & 2 & 6 \\ 7 & 2 & 9 & 1 \end{pmatrix}\begin{pmatrix} 7 & 0 \\ 5 & 4 \\ 3 & 8 \\ 1 & 5 \end{pmatrix}$

   c. $\begin{pmatrix} 5 & 9 \\ 1 & 2 \end{pmatrix}\begin{pmatrix} 4 \\ 7 \end{pmatrix}$

d. $\begin{pmatrix} 5 & 7 & 0 \\ 8 & 4 & 6 \\ 9 & 7 & 4 \end{pmatrix}\begin{pmatrix} 7 \\ 1 \\ 3 \end{pmatrix}$

Unit 124
Inverting a matrix: row operations

A matrix is invertible if and only if its determinant is invertible. When dealing with real numbers, this means that the determinant cannot be zero. When dealing with modular arithmetic, this means that the determinant must have an inverse in that modular system of numbers.

One way to invert a matrix is by using elementary row operations. There are three *elementary row operations*:

- swap two rows
- multiply every element in a row by a scalar
- add one row to another (add elements that are in the same column)

To invert a square matrix, we first extend it with a copy of the identity matrix with the same dimensions. Then we apply row operations until the identity matrix appears on the other side. The matrix that is now where the identity used to be is the inverse matrix. As an example, consider this matrix:

$$\mathbf{M} = \begin{pmatrix} 2 & 1 & 2 \\ 1 & 3 & 3 \\ 5 & 6 & 7 \end{pmatrix}$$

Extend it with the 3×3 identity matrix:

$$\left(\begin{array}{ccc|ccc} 2 & 1 & 2 & 1 & 0 & 0 \\ 1 & 3 & 3 & 0 & 1 & 0 \\ 5 & 6 & 7 & 0 & 0 & 1 \end{array}\right)$$

Swap the top and middle rows (R_0 and R_1, since we count from zero):

$$\left(\begin{array}{ccc|ccc} 1 & 3 & 3 & 0 & 1 & 0 \\ 2 & 1 & 2 & 1 & 0 & 0 \\ 5 & 6 & 7 & 0 & 0 & 1 \end{array}\right)$$

Replace the middle row with $R_1 - 2R_0$, and the bottom row with $R_2 - 5R_0$:

$$\left(\begin{array}{ccc|ccc} 1 & 3 & 3 & 0 & 1 & 0 \\ 0 & -5 & -4 & 1 & -2 & 0 \\ 0 & -9 & -8 & 0 & -5 & 1 \end{array}\right)$$

Replace the bottom row with $R_2 - 2R_1$:

$$\left(\begin{array}{ccc|ccc} 1 & 3 & 3 & 0 & 1 & 0 \\ 0 & -5 & -4 & 1 & -2 & 0 \\ 0 & 1 & 0 & -2 & -1 & 1 \end{array}\right)$$

Swap the middle and bottom rows:

$$\left(\begin{array}{ccc|ccc} 1 & 3 & 3 & 0 & 1 & 0 \\ 0 & 1 & 0 & -2 & -1 & 1 \\ 0 & -5 & -4 & 1 & -2 & 0 \end{array}\right)$$

Replace the bottom row with $-(R_2 + 5R_1)/4$:

$$\left(\begin{array}{ccc|ccc} 1 & 3 & 3 & 0 & 1 & 0 \\ 0 & 1 & 0 & -2 & -1 & 1 \\ 0 & 0 & 1 & 9/4 & 7/4 & -5/4 \end{array}\right)$$

Now that the leading entries of each row are all 1, we need to put zeroes in the off-diagonal entries on the left side. Replace the top row with $R_0 - 3R_1 - 3R_2$:

$$\left(\begin{array}{ccc|ccc} 1 & 0 & 0 & -3/4 & -5/4 & 3/4 \\ 0 & 1 & 0 & -2 & -1 & 1 \\ 0 & 0 & 1 & 9/4 & 7/4 & -5/4 \end{array}\right)$$

Now $\mathbf{M}^{-1}$ is the square matrix on the right-hand side.

Reading and references

Wikipedia:
en.wikipedia.org/wiki/Matrix_(mathematics)
en.wikipedia.org/wiki/Square_matrix
en.wikipedia.org/wiki/Determinant
en.wikipedia.org/wiki/Elementary_matrix

Programming tasks

1. Write a function to find the determinant of a square matrix. Add it as a method to your class of matrices.

2. Write a function to determine whether a square matrix is invertible. Add it as a method to your class of matrices.

3. Write a function to invert a matrix by using elementary row operations. It should first check whether the matrix is invertible and have some way to alert the main program if it is not.

Exercises

1. Find the inverses of these matrices:

   a. $\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$

   b. $\begin{pmatrix} 2 & 2 \\ 1 & 3 \end{pmatrix}$

   c. $\begin{pmatrix} 17 & 19 & 15 \\ 10 & 12 & 13 \\ 16 & 18 & 17 \end{pmatrix}$

   d. $\begin{pmatrix} 3 & 9 & 5 & 8 \\ 7 & 0 & 6 & 2 \\ 2 & 3 & 2 & 6 \\ 4 & 7 & 5 & 1 \end{pmatrix}$

Unit 125
Inverting a matrix: adjugate matrix

In this unit we present another way to find the inverse of a square matrix.

Given a row number and a column number, the *minor matrix* of a matrix is formed by taking the matrix and removing that row and that column. For example, the (1, 2) minor matrix of this matrix

$$\mathbf{A}=\begin{pmatrix} 4 & 3 & 9 & 4 & 3 & 2 \\ 7 & 6 & 8 & 5 & 7 & 0 \\ 8 & 2 & 1 & 3 & 2 & 5 \end{pmatrix}$$

is found by deleting the second row (because the first is row 0) and the third column (because the first is column 0):

$$\mathbf{M}_{1,2}=\begin{pmatrix} 4 & 3 & 4 & 3 & 2 \\ 8 & 2 & 3 & 2 & 5 \end{pmatrix}$$

The *determinant* of a square matrix is a number derived from the matrix. We will define it in a recursive manner. The determinant of a 1×1 matrix is the value of the only entry. The determinant of a 2×2 matrix **A** is

$$\det \mathbf{A} = A_{00}\, A_{11} - A_{01}\, A_{10}$$

The determinant of a square matrix with arbitrary dimension is found from the determinants of its minor matrices:

$$\det \mathbf{A}=\sum_{i=0}^{n-1} (-1)^i A_{0i} \det \mathbf{M}_{0,i}$$

where $\mathbf{M}_{0,i}$ is the (0, i) minor matrix of **A**. With this definition, we can find the determinant of any square matrix by working our way down until we are working with determinants of 2×2 matrices. An important thing to know is that a matrix is invertible in the set of matrices if and only if its determinant is invertible in the set of numbers. In the realm of real numbers, all numbers are invertible except zero. However, in modular arithmetic we know that not all numbers have inverses.

The combination of a power of −1 and the determinant of a minor matrix that we see in the formula above is called a cofactor. In general, the *cofactor* of an element A_{ij} is

$$C_{i,j} = (-1)^{i+j} \det \mathbf{M}_{i,j}$$

The matrix **C** whose *i*, *j* entry is the cofactor of A_{ij} is called the *cofactor matrix* of **A**. The *transpose* of a matrix **A** is another matrix $\mathbf{A}^{\mathrm{T}}$ that is obtained by exchanging the rows with the columns of **A**:

$$(A^{\mathrm{T}})_{ij} = A_{ji}$$

If we take the transpose of the cofactor matrix of **A**, what get is called the *adjugate matrix* of **A**. Finally, we are able to say that the inverse of a square matrix **A** is the multiplicative inverse of the determinant of **A** multiplied by the adjugate matrix of **A**.

Reading and references

Wikipedia:
en.wikipedia.org/wiki/Matrix_(mathematics)
en.wikipedia.org/wiki/Square_matrix
en.wikipedia.org/wiki/Minor_(linear_algebra)
en.wikipedia.org/wiki/Determinant
en.wikipedia.org/wiki/Adjugate_matrix

Programming tasks

1. Write a function that finds the minor matrix of a square matrix given a row number and a column number.

2. Write a function to find a cofactor of an element in a square matrix.

3. Write a function to find the cofactor matrix of a square matrix.

4. Write a function to find the transpose of a matrix. It is OK if you only consider square matrices.

5. Write a function to find the adjugate of a square matrix.

6. Write a function to invert a matrix by finding the adjugate matrix. It should first check whether the matrix is invertible and have some way to alert the main program if it is not.

Exercises

1. Find the inverses of these matrices:

   a. $\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$

   b. $\begin{pmatrix} 2 & 2 \\ 1 & 3 \end{pmatrix}$

c. $\begin{pmatrix} 17 & 19 & 15 \\ 10 & 12 & 13 \\ 16 & 18 & 17 \end{pmatrix}$

d. $\begin{pmatrix} 3 & 9 & 5 & 8 \\ 7 & 0 & 6 & 2 \\ 2 & 3 & 2 & 6 \\ 4 & 7 & 5 & 1 \end{pmatrix}$

Unit 126
Matrices over the set of residues

For use in ciphers, we need to work with numbers from the set of residues. Review Units 19 and 27 if you have forgotten what we mean. For the rest of this part of the book, we will be working with matrices and vectors whose elements are taken from the set of residues, usually modulo 26. This means that every number, whether it appears in a vector, in a matrix, or alone, is a member of $\mathbb{Z}_{26}$. All multiplications and additions (and subtractions) must be done modulo 26. The inverse of any number must be found by the algorithm in Unit 27.

In the last unit, we saw how to find the inverse of a square matrix as the inverse of the determinant times the adjugate matrix. We now have to be careful that we handle our arithmetic correctly, and that when we find the inverse of the determinant we find its inverse modulo 26. If that inverse does not exist, then the matrix is not invertible.

Programming tasks

1. Find your function from Unit 27 to get the multiplicative inverse of a number in modular arithmetic.

2. Make revisions of your functions from the previous unit so that you now have a function that multiplies a square matrix by a vector or another matrix. It should return a vector or matrix all of whose components are in $\mathbb{Z}_{26}$. Allow for the option to change the modulus.

3. Make revisions of your functions from the previous unit so that you now have a function that finds the determinant of a square matrix whose elements are in $\mathbb{Z}_{26}$. Allow for the option to change the modulus.

4. Write a function to determine whether a square matrix over $\mathbb{Z}_{26}$ is invertible. Allow for the option to change the modulus.

5. Make revisions of your functions from the previous unit so that you now have a function that finds the inverse of a square matrix over $\mathbb{Z}_{26}$. Allow for the option to change the modulus.

Exercises

1. Find these products modulo 26:

a. $\begin{pmatrix} 9 & 6 & 10 \end{pmatrix}\begin{pmatrix} 10 \\ 18 \\ 13 \end{pmatrix}$

b. $\begin{pmatrix} 15 & 9 \\ 2 & 2 \end{pmatrix}\begin{pmatrix} 8 & 4 \\ 24 & 25 \end{pmatrix}$

c. $\begin{pmatrix} 11 & 9 & 18 \\ 5 & 19 & 7 \\ 6 & 10 & 10 \end{pmatrix}\begin{pmatrix} 4 \\ 11 \\ 20 \end{pmatrix}$

d. $\begin{pmatrix} 25 & 18 & 19 & 18 \\ 14 & 19 & 24 & 15 \\ 8 & 4 & 1 & 7 \\ 23 & 8 & 24 & 6 \end{pmatrix}\begin{pmatrix} 23 \\ 13 \\ 5 \\ 21 \end{pmatrix}$

e. $\begin{pmatrix} 16 & 22 & 1 & 8 & 1 \\ 24 & 9 & 16 & 1 & 25 \\ 2 & 4 & 2 & 19 & 11 \\ 11 & 17 & 0 & 1 & 12 \\ 18 & 9 & 25 & 1 & 5 \end{pmatrix}\begin{pmatrix} 13 \\ 4 \\ 16 \\ 7 \\ 6 \end{pmatrix}$

2. Find the inverses of these matrices over $\mathbb{Z}_{26}$:

a. $\begin{pmatrix} 8 & 19 \\ 21 & 18 \end{pmatrix}$

b. $\begin{pmatrix} 15 & 14 & 8 \\ 0 & 15 & 11 \\ 6 & 7 & 0 \end{pmatrix}$

c. $\begin{pmatrix} 21 & 3 & 0 \\ 8 & 8 & 19 \\ 13 & 2 & 4 \end{pmatrix}$

d. $\begin{pmatrix} 16 & 19 & 2 & 14 \\ 20 & 8 & 0 & 13 \\ 0 & 5 & 19 & 0 \\ 13 & 8 & 8 & 11 \end{pmatrix}$

e. $\begin{pmatrix} 4 & 17 & 4 & 14 & 7 \\ 11 & 14 & 15 & 6 & 8 \\ 4 & 4 & 7 & 17 & 2 \\ 2 & 13 & 0 & 0 & 0 \\ 19 & 2 & 11 & 15 & 11 \end{pmatrix}$

Unit 127
Hill cipher

The $n \times n$ *Hill cipher* is a block cipher that uses matrix multiplication to encipher each block of n letters. (A *block cipher* is a cipher that acts on blocks of texts, rather than individual letters.) Each block is written as a vector whose components are the numerical equivalents of its letters, where `A` = 0, `B` = 1, ..., `Z` = 25. Encipherment is done by multiplying this vector by the key, which is a (square) $n \times n$ matrix. Decipherment is done with the inverse matrix. All operations are done modulo 26 (or the length of the alphabet, if we are using a different one).

Let's work an example. Suppose we have this short message and that we want to encipher it with the matrix that follows:

```
THIS MESSAGE WAS ENCRYPTED WITH A HILL CIPHER
```

$$\mathbf{M} = \begin{pmatrix} 7 & 8 & 11 \\ 11 & 2 & 8 \\ 15 & 7 & 4 \end{pmatrix}$$

We first divide the plaintext into blocks of three letters, and pad with nulls if necessary:

```
THI SME SSA GEW ASE NCR YPT EDW ITH AHI LLC IPH ERX
```

The first block, `THI`, is expressed as this column vector:

$$\mathbf{V} = \begin{pmatrix} 19 \\ 7 \\ 8 \end{pmatrix}$$

This block in enciphered to a new vector:

$$\mathbf{U} = \mathbf{M}\mathbf{V} = \begin{pmatrix} 7 & 8 & 11 \\ 11 & 2 & 8 \\ 15 & 7 & 4 \end{pmatrix} \begin{pmatrix} 19 \\ 7 \\ 8 \end{pmatrix} = \begin{pmatrix} 17 \\ 1 \\ 2 \end{pmatrix}$$

This vector becomes the block `RBC` in the ciphertext. The full ciphertext is

```
RBC GUG KAG EQY GQM IXR DEV ISN ZAV OAD FDQ TST BCL
```

Decipherment uses the inverse matrix:

$$\mathbf{M}^{-1} = \begin{pmatrix} 12 & 5 & 22 \\ 20 & 5 & 13 \\ 11 & 5 & 12 \end{pmatrix}$$

For the first block of the ciphertext:

$$\mathbf{V} = \mathbf{M}^{-1}\mathbf{U} = \begin{pmatrix} 12 & 5 & 22 \\ 20 & 5 & 13 \\ 11 & 5 & 12 \end{pmatrix} \begin{pmatrix} 17 \\ 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 19 \\ 7 \\ 8 \end{pmatrix}$$

The matrix used in a Hill cipher can often be expressed as a keyword by using letters that are equivalent to the entries of the matrix. For the example above, the keyword is `HILLCIPHE`(`r`).

Given a ciphertext whose blocksize is unknown, it is often possible to find it using the index of coincidence. The same technique we used in Unit 39 to find the period of polyalphabetic substitution cipher can help here. However, the height of the peaks in the graph of IoC versus blocksize is not always above a predetermined threshold. Often it is even difficult to discern the peaks. These examples are for 2×2, 3×3, and 4×4 Hill ciphers. As you can see, it is sometimes difficult to determine the block size.

A more reliable way to find the block size is with the index of coincidence for digrams, trigrams, etc., provided that it is not larger than five. In the exercises from Unit 10, you found ranges of the values for IoC2, IoC3, ... for typical English text. A Hill cipher, since it enciphers each block always in the same manner, will have the same IoC*n* as English when the block size is *n*.

Reading and references

Lester S. Hill, “Cryptography in the Algebraic Alphabet,” *The American Mathematical Monthly* 36:6 (1929) 306-312, DOI: 10.2307/2298294, www.jstor.org/stable/2298294, web.archive.org/web/20110719235517/http://w08.middlebury.edu/INTD1065A/Lectures/Hill Cipher Folder/Hill1.pdf

Lester S. Hill, “Concerning Certain Linear Transformation Apparatus of Cryptography,” *The American Mathematical Monthly* 38:3 (1931) 135-154, DOI: 10.1080/00029890.1931.11987161, www.jstor.org/stable/2300969, www.cs.jhu.edu/~cgarman/files/Hill2.pdf

Abraham Sinkov, *Elementary Cryptanalysis: A Mathematical Approach*, 2nd edition, revised by Todd Feil, published by Mathematical Association of America, 2009; www.jstor.org/stable/10.4169/j.ctt19b9krf; chapter 4.

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, pages 404-409.

Wikipedia, en.wikipedia.org/wiki/Hill_cipher

Crypto Corner, crypto.interactive-maths.com/hill-cipher.html

Chris Christensen, “Lester Hill Revisited,” *Cryptologia* 38:4 (2014) 293-332, DOI: 10.1080/01611194.2014.915260

Programming tasks

1. Implement the cipher. Allow for any block size. Verify that the matrix is invertible before enciphering.

2. Write a function to try to determine the block size. Perhaps it could return a value of zero or `False` if it is unable to clearly determine the size.

3. Implement a brute-force attack. For an $n \times n$ matrix, there are n^2 values to vary. You should discard matrices that are not invertible.

4. Compile a list of four-letter words that can fill a square matrix so that it is invertible. Do the same for nine- and sixteen-letter words.

5. Implement a dictionary attack. You should discard matrices that are not invertible.

Exercises

1. Encipher this text with keyword `HILL`.

   > This, and enough, premised, I go souse into my personal history. My maiden name was Frances Hill. I was born at a small village near Liverpool, in Lancashire, of parents extremely poor, and, I piously believe, extremely honest.
   >
   > (from *Memoirs of Fanny Hill* by John Cleland)

2. Decipher this text with keyword `PROFESSED`.

```
SNSPTIRUZMCUZBWJZIJVGYIVSGJUUZYWDDWMDLPVWKTVYJPUGLPAGQLLYBV
HLIPLXXIKKGKAOHSDCBDSKSKXTTTMERLNKQFBNDBPULJUSHQRTWNSNSLWDP
WGBAKKMZMOEPKGDGEABNTIARIAVQHMAZEKPGYKRXXXTTEKNSNSYMOISTGYX
SSRJAKUAYVKKKPCXGYRHKCARPVPOIAQNYGGLVRMXJBUYLXVHSSRKIIRPAZV
KVYJBKJFGSDYIGSCBDPBVRHCMTLIJXHJMKRPOCNWRZMELXHWBRWVZWXTJFH
TKDWMHVYPHHXLDEYYDJZREVWQDAZEJOGHUXREVVGEDIZOTGELXOSJFUMVXC
CPCZVJWMMKITKYZHIIXJMXRDXSBDISSHTZOGYNOZOGBLZUSFSNSWPCQAFPG
FRAIMKLHABVGGUZNTJSHHRXSUVUUBZTYWDDCJVBTBKJEQDCQIHZNPPGSNSZ
DODCJPFICIAWHEPFKHRWOTGXJMDIZFXPDQQISQZHERGXTSSZEWNGEQKHTYI
XJMTOGDKNLUN
```

3. Can you determine the block size for this ciphertext?

```
IPKGRGNLJUQQGFTQMMYKABOCFYRBGEWEOMYICYODZYLOMJGFXXZDYNTOBIB
ZLTITEMDSQITHEDUUWUQJOJVAWIFKVXFUDCGFUTECOTWBSQLOCVCSEBSSNK
MBJFRGECMIERCGEVAXSBLVBZEAQNIHGHJZGBJMIJJQHNMAKHJCBIOZRDTAZ
MJDSWORRBHXOOCFDBMWNYAPXZASTVSXIOFITERXWVQMLAFSWXSTWKUVWSEF
OIBKUQIHZAELVRPVUQQSGBPCZRKTVAKBLKZXZMAWMNUHBMOLUWNIMUMICAS
OLWHJVLEKONNGOFCDKKTQSGEEZELIBCGDCBCIJLEOHEJODPHRISLZVAWZVJ
JZXJFHESLRYWDITGQUMERAFOXEJUWGCSQYGGJYVAZPJVHMPGYBMKERBHCFJ
UUUUJGPGICASFJHWWCNTEMAYXBUQZVDYDGHQQRMGBWBZKMRAIXDUYMTQXDR
MQZVMTDZUYTRVKWIMPVAREZPRLJIURMIDPAMSDRQZOZRIARCPJPLHWWICHD
RQCWVILQKAIMXMYWVEJRQZOZDUXOOGYXNNOMZXRMYJYHGRLFWJWPJHNADQC
TXFREHIJDNXKZONJTWWCRYESVCGWGVUWALSSPQACCGEVAXSBLVIERGK
```

4. Use a brute-force attack to break this ciphertext. Is there a keyword?

```
TOPGRHYADTVKXXSQYQXZUBWLGFALFCMGEZUBEBHGQKWODTGFYUBIMOWJFVK
XEVFOVTRYCNXWOJYMSVZWPLANLUCCTOKEUIPVWVSOJWURXAFVCUMPCUQJDE
WJFVSNEVJQEFIWJHQTVHYASVFVSQZWFVSAMROUTOHBIGWSFVCUQJHECURMN
QEZFVIGPCMYURTOLQLBXWOYOFZSSAGBITRYAJSACRFIHEANLIANGMGPZWAH
QWFCDIVGQWDTZWPYDTXWQAYWRMSSZWXWIKKRZGGDJK
```

5. Use a dictionary attack to break this ciphertext.

```
RMYKUKRLHLXETRSMCYXFOUYMHHNOJAMEERFEDXQRXOWOSIEAPXFDORZBCZZ
HLNBRMYRRGVTCNOZNYSOHYDORWPKRHPCEJEKOPGAJTDRPERMYVZUCHCDORS
WGAYJMJOVBULXRVXTGKOZNHAQDYZHKYRMNXKMCMIREMZITKDDLYRYWGDJZU
ZNXCEJKALYRYCEEDMTBMAQYUHHNWJWCEEDMTHMKREIZXYOCQEUHSMWJLYTT
QDKSEUHZNHPXFWGDOVAVJSBROHYBYUCFOCIKHXIKPSLUJYJWWAWUHQKSAGX
XMHHNMJOOINZNHROZUMKMCYRBEMLKVTCAJUJZUBRODKSFEBJZWPBUMJOYGN
GGCNTZZBSJZUHYYVEBOWNZULTPRRLHJWWAWURYPKRKQSEAJUPUKITIYFYHY
BQBEQBQQYUUUYUFCRLHWFGVXCYKCBTRHRSAUEZNXKKYBTRRMYXISLBNSQJV
NZPWGJLYKXMGVUDPQRHPQAFXISRBXWOSIEAIHUHYBDGAZMZSMYBHXLGUSTS
IKHCVOKKOPYDDORZCVOLXYWBWGDRLUXLRYWNLWBZNHOSTZNXGWKZREYMTND
KOMAFSIHRADORGHSOAFZWQMRMLSSWXWOFQZGCECLPAPROXSEIMLL
```

Unit 127½
Brute-forcing Hill ciphers with less brute force

Suppose we want to break a Hill cipher that uses $n \times n$ matrices. To use brute force, we could try all possible $26^{n \cdot n}$ matrices (and maybe rule out those that are not invertible), generate a candidate plaintex for each, and choose one with the highest tetragram fitness (Unit 11). For 2×2 matrices, this is not too difficult, since $26^4 = 456{,}976$. But the work involved in the attack grows like an exponential of a square as the size of the matrices embiggens. For 3×3 matrices, the number is 5,429,503,678,976, which already makes the attack infeasible. For 4×4 matrices, the number is 43,608,742,899,428,874,059,776, which would require that we check over 100,000 matrices every second from the beginning of the universe until last Tuesday. We need a better way.

The trick to improving the brute-force attack is to notice that from each block of ciphertext, one letter has been enciphered with the same *vector*. So, for example, if we pick out the first letter of each block, we know that it was enciphered with the vector made up of the entries on the top row of the matrix. We need to work with the inverse matrix (the deciphering matrix), because we do not know the contents of each block, yet. So rewrite that previous sentence to say that one letter in each block of the plaintext is found by deciphering with the same row of the inverse matrix. This means that if we try all possible values for the n entries on just that row alone, we may be able to recover every n^{th} letter. But those letters are not contiguous in the text, so we need to use monogram fitness (Unit 10) to choose the best set of deciphered letters. This reduces the attack complexity from an exponential of a square to the exponential of a linear function. There is a catch, however: when we find a good vector, we do not a priori know on which row of the inverse matrix it belongs. Therefore, the second stage of the attack is to sort the rows; this is equivalent to breaking a columnar transposition cipher (Unit 93). Also, because we are relying on monogram statistics from a set of n letters for each row, this attack requires a ciphertext that is at least about $100n$ letters in length.

As usual, we look at an example to see how it works. Consider this plaintext from *The Old Man's Guide to Health and Longer Life* by John Hill:

> Old men's diseases are hard to cure; but they are easy to prevent. It must be a good natural fabric which has preserved itself so long; and the same strength may keep it much longer well, under good regulation. Moderate diet, and due exercise, are the best guardians of health in all: but in the advanced period here considered there are two great preservatives besides; these are Ease, and Cheerfulness: both are the natural offspring of health; and they will continue the blessing to which they owe their birth.

In blocks we have

```
OLD MEN SDI SEA SES ARE HAR DTO CUR EBU TTH EYA REE ASY TOP REV
ENT ITM UST BEA GOO DNA TUR ALF ABR ICW HIC HHA SPR ESE RVE DIT
SEL FSO LON GAN DTH ESA MES TRE NGT HMA YKE EPI TMU CHL ONG ERW
ELL UND ERG OOD REG ULA TIO NMO DER ATE DIE TAN DDU EEX ERC ISE
```

```
ARE THE BES TGU ARD IAN SOF HEA LTH INA LLB UTI NTH EAD VAN CED
PER IOD HER ECO NSI DER EDT HER EAR ETW OGR EAT PRE SER VAT IVE
SBE SID EST HES EAR EEA SEA NDC HEE RFU LNE SSB OTH ARE THE NAT
URA LOF FSP RIN GOF HEA LTH AND THE YWI LLC ONT INU ETH EBL ESS
ING TOW HIC HTH EYO WET HEI RBI RTH
```

First, let us look at the encryption. Take this 3×3 matrix:

$$\mathbf{M} = \begin{pmatrix} 22 & 6 & 7 \\ 7 & 4 & 6 \\ 12 & 1 & 24 \end{pmatrix}$$

When we encipher it, the first letter of each block in the ciphertext is found by finding the product of the vector on the top row of **M** with the column vector of plaintext letters. For the first block,

$$c_1 = \begin{pmatrix} 22 & 6 & 7 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} = \begin{pmatrix} 22 & 6 & 7 \end{pmatrix} \begin{pmatrix} 14 \\ 11 \\ 3 \end{pmatrix} = 5$$

where the arithmetic is done modulo 26, and so the letter is `F`. The first letter of the second block of the ciphertext is calculated with the same row vector from **M** but the next column vector from the plaintext and found to be `P`.

$$c_4 = \begin{pmatrix} 22 & 6 & 7 \end{pmatrix} \begin{pmatrix} p_4 \\ p_5 \\ p_6 \end{pmatrix} = \begin{pmatrix} 22 & 6 & 7 \end{pmatrix} \begin{pmatrix} 12 \\ 4 \\ 13 \end{pmatrix} = 15$$

What we are doing is enciphering the first slice of the ciphertext.

```
F__ P__ C__ E__ A__ A__ N__ S__ X__ A__ J__ Y__ K__ ...
```

The second row of the matrix is used to encipher the second slice, and the third row for the third slice. The resulting ciphertext is

```
FER PWS CEV EMM AQC AOJ NVY SZB XOK AWJ JRZ YUU KDS QIW JTE ZBK
NMX KWN FOM UXQ CAG OVX HDG XWB VCT EOC IPK OZN HCP QUG ITJ NLG
DAQ EJY BDQ PQU VJP OWO YAI CRD NVU STS SYE AGV GPS HEJ MEN GUV
XIL TCN YCB XQU YPO MCR SPA OPK BJG MWL MZK PDU QXZ NAG WEJ AWC
AOJ UDT QBG WRM TIL HES VEM WNK PNH UEF DXL MEJ HBF FUQ HRS LWW
FPU VAA LLC QQW IDC BJG FYN LLC ZAO SCX VQK NMK SPH TKE XBG SIF
OYB XUK RGC SRA ZAO ISA EMM GLZ YLC YZN KXH RWY DIR AOJ UDT DXO
WAX XHG LPW TVE RYY WNK PNH VSH UDT OSI KDJ ZEN EUR RQB PUB KAE
KOT GJQ IPK FLL SAS RYW AJU UPH RDB
```

But we are here to break ciphers, so let us look at the decipherment using the same row-by-row method, and then a bit later we will attack it as if we did not know the matrix. The inverse matrix is

$$\mathbf{M}^{-1} = \begin{pmatrix} 14 & 7 & 18 \\ 18 & 24 & 5 \\ 15 & 2 & 6 \end{pmatrix}$$

The first letter of the plaintext is calculated from the product of the first row of the inverse matrix and the first column (the first block) of ciphertext letters:

$$p_1 = \begin{pmatrix} 14 & 7 & 18 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 14 & 7 & 18 \end{pmatrix} \begin{pmatrix} 5 \\ 4 \\ 17 \end{pmatrix} = 14$$

As usual, all arithmetic is done modulo 26. The first letter is `O`, as we already knew. In the following equation, as i runs over the block number, we calculate the first letter of each plaintext block:

$$p_{3i+1} = \begin{pmatrix} 14 & 7 & 18 \end{pmatrix} \begin{pmatrix} c_{3i+1} \\ c_{3i+2} \\ c_{3i+3} \end{pmatrix}$$

This gives us one slice of the plaintext:

```
O__ M__ S__ S__ S__ A__ H__ D__ C__ E__ T__ E__ R__ ...
```

If we collect all of the letters from this slice, we have

```
OMSSSAHDCETERATREIUBGDTAAIHHSERDSFLGDEMTNHYETCOEEUEORUTNDADTDEEI
ATBTAISHLILUNEVCPIHENDEHEEOEPSVISSEHEESNHRLSOATNULFRGHLATYLOIEEE
ITHHEWHRR
```

The monogram fitness using the cosine of the angle between 26-dimensional vectors for this collection of letters is 0.957, which is very close to 1. This fact is worth mentioning, because this is the kind of statistics that we can use to make sure that we have used a correct decryption vector. The other two slices of the plaintext are found in the same way, using the second and third rows of the inverse matrix.

Now let's back up to the ciphertext and forget that we know the matrices. All we know is that we have a 3×3 Hill cipher. We will try all possible 3-vectors and use them to decipher a slice of the plaintext. Once we find three slices that have good fitness, we can be confident that we have found the three rows of the inverse matrix. For the 3×3 case, we set up three nested loops in our algorithm for this first stage of the attack:

```
set up storage for three 3-vectors and three values of fitness
for v1 = 0, ..., 25 do:
    for v2 = 0, ..., 25 do:
        for v3 = 0, ..., 25 do:
            use row vector (v1 v2 v3) to decrypt a slice of plaintext
            calculate the fitness of that slice
            if that value is higher than the lowest of the stored values:
                replace the corresponding stored 3-vector with (v1 v2 v3)
```

```
                    replace the lowest stored value of fitness with the new value
            if the three stored values of fitness are good enough:
                    exit loops
    output the three stored 3-vectors
```

To generate the following histogram, we let the algorithm run through *all* possible 3-vectors, and did not exit when we had found the three we wanted. The fitness function is the monogram fitness based on the cosine of the angle between the frequency vector of the text and the vector of English frequencies. Notice that at the high end of the graph, three data points stand out. These are the values of the statistic that correspond to the three best 3-vectors, which are (14, 7, 18), (15, 2, 6), and (18, 24, 5).

What do you mean you can't see them? They're right there. Fine, here is a close-up of the upper tail:

At this point, we have the three best 3-vectors, listed above, and their corresponding slices:

```
OMSSSAHDCETERATREIUBGDTAAIHHSERDSFLGDEMTNHYETCOEEUEORUTND...
DNIASERORUHAEYPVTMTAOARFRWCAREETLONNHASETAEIULGWLDGDGAOOR...
LEDEERATUBTYESOENTSEONULBCIHPSVIESOATSERGMKPMHNRLNROELIME...
```

We have to interleave them to make a sensible plaintext. We can do this by anagramming, and for small matrices, this is easy. Or, we can concatenate them and treat the whole like a columnar transposition ciphertext and attack it with the methods of Unit 93, which were adaptations of the methods from Units 89 and 90. Once we know the correct order of the slices, we can put the 3-vectors in the same order; the result is the decryption matrix.

Programming tasks

1. Implement the attack. For the second stage, in which the transposition is solved, use a brute-force method; for small matrices, which means a small number of columns, this adds very little time to the attack.

Exercises

1. Break this ciphertext, which was Exercise 5 in Unit 127. The block size is 3.

```
RMYKUKRLHLXETRSMCYXFOUYMHHNOJAMEERFEDXQRXOWOSIEAPXFDORZBCZZHLNBR
MYRRGVTCNOZNYSOHYDORWPKRHPCEJEKOPGAJTDRPERMYVZUCHCDORSWGAYJMJOVB
ULXRVXTGKOZNHAQDYZHKYRMNXKMCMIREMZITKDDLYRYWGDJZUZNXCEJKALYRYCEE
DMTBMAQYUHHNWJWCEEDMTHMKREIZXYOCQEUHSMWJLYTTQDKSEUHZNHPXFWGDOVAV
JSBROHYBYUCFOCIKHXIKPSLUJYJWWAWUHQKSAGXXMHHNMJOOINZNHROZUMKMCYRB
EMLKVTCAJUJZUBRODKSFEBJZWPBUMJOYGNGGCNTZZBSJZUHYYVEBOWNZULTPRRLH
JWWAWURYPKRKQSEAJUPUKITIYFYHYBQBEQBQQYUUUYUFCRLHWFGVXCYKCBTRHRSA
UEZNXKKYBTRRMYXISLBNSQJVNZPWGJLYKXMGVUDPQRHPQAFXISRBXWOSIEAIHUHY
BDGAZMZSMYBHXLGUSTSIKHCVOKKOPYDDORZCVOLXYWBWGDRLUXLRYWNLWBZNHOST
ZNXGWKZREYMTNDKOMAFSIHRADORGHSOAFZWQMRMLSSWXWOFQZGCECLPAPROXSEIM
LL
```

2. Break this ciphertext, which has block size 4.

```
MXAKUULBUQAECAWPTVHJRYKLMMQJOYMKVMUGMBVHOEPJHLSDLZLNUNIWYWCSWOIK
WKHOYCYSBXISFXWQTMBSXDSCKSDEKBXNCHDXINSNGQKBECOFYSPSKIZUIYTNJSKB
QLTKUXOHMJQNLUVBSDMOMUUBJKEZBRUYZTISGUASJTCGNYEAQRZHVRQPEFFHEZAM
ISHDNUJRWCJMXQKDXRIQYMIZSBERSLWQXKBGPOCPKXYFVMUGIFZCRWPWLKPRABWO
VTYDZGZKDPQGIMGBFGVJHNYBRQVOXQFDPNEJQNZEOFPJCGSUIONFMWHCJVVPMQGQ
UVJEAJUWMDKFUYOGEAPWJVVPOZAZGNGNCOYJCULIQTJFHZSAPPOBAUUMMOMRSOAX
TQDTIOAFLHAIHAMXSRTKSGMGDKZUXXICJXSMDSKDZXAKYTJOTNUBQOBYLRTN
```

Unit 128
Attacking the Hill cipher with cribs: linear equations

Before we begin an attack, we need to know the size of the matrices. Checking the *n*-gram index of coincidence (see Unit 13) can be very helpful, up to block size 5.

An $n \times n$ Hill cipher has n^2 parameters, and therefore we need n^2 constraints to completely determine the key matrix. These constraints take the form of equations that we get by matching a crib to a piece of ciphertext. If the crib is too short, or some parts of it repeat, then there may be too few constraints; in that case, we have to brute-force the remaining parameters.

When we are matching a crib to a piece of ciphertext, we must remember that we can only use parts of the crib that cover complete blocks. Let's return to the example from the previous unit. If our crib is `MESSAGE`, then we move the crib over the plaintext and try each position until we can find a key that gives a good plaintext.

```
MES SAG E
RBC|GUG|KAG|EQY|GQM|IXR|DEV|ISN|ZAV|OAD|FDQ|TST|BCL
```

In the above position, the constraints from the first block are (remember that the key is a matrix **M**, and `A`=0, `B`=1, ...)

$$\begin{aligned} 17 &= 12\,M_{00} + 4\,M_{01} + 18\,M_{02} \\ 1 &= 12\,M_{10} + 4\,M_{11} + 18\,M_{12} \\ 2 &= 12\,M_{20} + 4\,M_{21} + 18\,M_{22} \end{aligned}$$

From the second block we get

$$\begin{aligned} 6 &= 18\,M_{00} + \phantom{18\,M_{01} +} 6\,M_{02} \\ 20 &= 18\,M_{10} + \phantom{18\,M_{11} +} 6\,M_{12} \\ 6 &= 18\,M_{20} + \phantom{18\,M_{21} +} 6\,M_{22} \end{aligned}$$

From the third block we do not get any constraints, unless we brute-force the next two plaintext characters. The equations we get, where x and y are the missing characters from the third block, are:

$$\begin{aligned} 10 &= 4\,M_{00} + x\,M_{01} + y\,M_{02} \\ 0 &= 4\,M_{10} + x\,M_{11} + y\,M_{12} \\ 6 &= 4\,M_{20} + x\,M_{21} + y\,M_{22} \end{aligned}$$

Collect the equations for the top row:

$$\begin{aligned} 17 &= 12\,M_{00} + 4\,M_{01} + 18\,M_{02} \\ 6 &= 18\,M_{00} + \phantom{4\,M_{01} +} 6\,M_{02} \end{aligned}$$

$$10 = 4\,M_{00} + x\,M_{01} + y\,M_{02}$$

From these three equations, we can solve for M_{00}, M_{01}, and M_{02}, for each choice of x and y. We then collect the three equations for the middle row of the matrix, and the bottom row. For each of the $26^2 = 676$ choices for x and y, we get a complete key matrix (unless some of the equations are not independent and so cannot be solved). We invert each matrix and decipher the ciphertext. Unfortunately, for this position of the crib, none of the plaintexts that we get are acceptable.

Next, we move the crib over by one position:

```
 ME SSA GE
RBC|GUG|KAG|EQY|GQM|IXR|DEV|ISN|ZAV|OAD|FDQ|TST|BCL
```

Now we must brute-force the first character of the first block and the last character of the third block. The process is similar to the above, and again we do not find an acceptable plaintext. We can shift the crib one space at a time. When we do find a good plaintext, it is in this position:

```
     ME SSA GE
RBC|GUG|KAG|EQY|GQM|IXR|DEV|ISN|ZAV|OAD|FDQ|TST|BCL
```

We need to check all possibilities for the first plaintext character in the second block and the last plaintext character in the fourth block. For each of the $26^2 = 676$ choices, we set up nine equations. The equations form three sets of three. Each set allows us to solve for one row of the key matrix.

Programming tasks

1. Implement the attack. Build an attack for 2×2 and 3×3 Hill ciphers, at least. If you are very clever, perhaps you can build an attack for any block size. Be careful in handling cribs of various lengths. Use tetragram fitness to determine when you have found an acceptable plaintext.

Exercises

1. Finish the example above using linear equations by hand to find the key. Check that it matches the one we used in the previous unit to encipher the message.

2. Break this ciphertext with the crib `FOLLY`. What might the keyword be?

```
GHTSYUESFCYMNNBGCGESCNGHAXDULTNVXOYRWPWTKPGUNKAWNPDVESBBRFN
TRNBGOWXLWTYNDPNSDLEYGUBBEHYQLAGOGJCNGIYUWTEULBBBREYVZVBGCG
ESAEDNCQLHDLZZGRXOYKGUNKAWNNNSCNYBYCOLYRYWLIAWNLXOGULXGBDNL
GYTAEXWGJDLEXYQFBWDDLJHBFFXNNNSDLERDDLHQGXREYWTKBWQDGGPGULN
GJDNBBREEXWKMTOIZBTRMUDPYSWXYQNSUWWLYDMOUUGSYJAMMLAYMOUUGHT
CWTKVDNPDOLYUXRCILNGJESYVGVKEFWDNEACILWLZNXTCGYWMEQNQWKZBCI
LULJUVDAOCWGEOGUNKAWNSGYYNGIXWYOXLNSXRCXANFWGGYBGHTCWTKMYJL
JUSGIDVUU
```

3. Break this ciphertext with the crib `PITILESS`.

GQTIRDBVCMCVUCJCPZFCSPEOFOPRBDTOQGQAHWBXNBWLEWJCNUBRMRETIEV
CJSKZLCUDWHWBGESEPEVBRGXWCHDQSFAHWNRUJMHTRRPAAZGYGEZTDGDLEL
CSEMNLOCGESEPMNRUAFKGQTUHGHSFCPEIYGHCRDCSIHWRTIOJVKAXSRIGSF
ZMKNAZWMVIRDVUHYDBPEUPEVRBDOFSJVCHSFLCSLWFRBDREQCRUKPMFDHAH
OEMNZRJQLYYGHGFZSQHLAQBPQTQHGFSLQBWKHOFNMACHRFGXEKUYYDBFYZV
JGATEDYGGXERWCPPKGISLAALBOCSTCIYIVFMKPUSKCEGFYWUWSQMAYALQCE
KMWWFSNSWMCVUCJBQDUWCKLSYDBTUNDKPXMKEIZAFKIBRDXAGKE

Unit 129
Attacking the Hill cipher with cribs: matrices

There is another way to handle cribs that does not involve solving so many equations, given that we have a computer routine to find inverses of matrices (a task from Units 123 and 124). If we know the block size n, and if we have a crib that is n^2 in length, and if we know that the crib starts on a block boundary, then we can perform the attack by manipulating matrices instead of equations. Consider the example ciphertext above, and the crib `A HILL CIPHER`. We assume that it begins on a block boundary; if not, then we can try dropping the initial letter until it does (we will know that it does when the attack succeeds). Pack the first nine letters by columns into this plaintext matrix:

$$\mathbf{P} = \begin{pmatrix} \mathrm{A} & \mathrm{L} & \mathrm{I} \\ \mathrm{H} & \mathrm{L} & \mathrm{P} \\ \mathrm{I} & \mathrm{C} & \mathrm{H} \end{pmatrix} = \begin{pmatrix} 0 & 11 & 8 \\ 7 & 11 & 15 \\ 8 & 2 & 7 \end{pmatrix}$$

It is imperative that we check that this matrix is invertible before continuing. Its determinant is 7 modulo 26, which is invertible (it shares no factors with the modulus), and so we can proceed. At each block boundary, we construct a ciphertext matrix by packing nine letters by columns. For the first three blocks of the ciphertext, the matrix we get is

$$\mathbf{C} = \begin{pmatrix} \mathrm{R} & \mathrm{G} & \mathrm{K} \\ \mathrm{B} & \mathrm{U} & \mathrm{A} \\ \mathrm{C} & \mathrm{G} & \mathrm{G} \end{pmatrix} = \begin{pmatrix} 17 & 6 & 10 \\ 1 & 20 & 0 \\ 2 & 6 & 6 \end{pmatrix}$$

Its determinant modulo 26 is zero, therefore we cannot use it. Since the crib matrix **P** is invertible, and we know that the key matrix **K** must be invertible, then it follows that the corresponding ciphertext matrix must also be invertible. This is true because

$$\det(\mathbf{AB}) = \det(\mathbf{A}) \cdot \det(\mathbf{B})$$

for any matrices **A** and **B**, a matrix is invertible if and only if its determinant is invertible, and the product of two invertible numbers is also invertible. Since **C** = **K P** and **K** and **P** are invertible, **C** must also be so. We move on and try the ciphertext matrix at the next block boundary:

$$\mathbf{C} = \begin{pmatrix} \mathrm{G} & \mathrm{K} & \mathrm{E} \\ \mathrm{U} & \mathrm{A} & \mathrm{Q} \\ \mathrm{G} & \mathrm{G} & \mathrm{Y} \end{pmatrix} = \begin{pmatrix} 6 & 10 & 4 \\ 20 & 0 & 16 \\ 6 & 6 & 24 \end{pmatrix}$$

It, too, in not invertible. A bit later in the ciphertext we can make this matrix:

$$\mathbf{C} = \begin{pmatrix} \mathrm{I} & \mathrm{D} & \mathrm{I} \\ \mathrm{X} & \mathrm{E} & \mathrm{S} \\ \mathrm{R} & \mathrm{V} & \mathrm{N} \end{pmatrix} = \begin{pmatrix} 8 & 3 & 8 \\ 23 & 4 & 18 \\ 17 & 21 & 13 \end{pmatrix}$$

Its determinant is 5 modulo 26, and so it is invertible. Multiplying both sides of the equation **C** = **K P** by $\mathbf{P}^{-1}$ on the right we have

$$\mathbf{K} = \mathbf{C}\,\mathbf{P}^{-1} = \begin{pmatrix} 8 & 3 & 8 \\ 23 & 4 & 18 \\ 17 & 21 & 13 \end{pmatrix}\begin{pmatrix} 0 & 11 & 8 \\ 7 & 11 & 15 \\ 8 & 2 & 7 \end{pmatrix}^{-1} = \begin{pmatrix} 8 & 3 & 8 \\ 23 & 4 & 18 \\ 17 & 21 & 13 \end{pmatrix}\begin{pmatrix} 3 & 21 & 11 \\ 25 & 2 & 8 \\ 8 & 20 & 15 \end{pmatrix} = \begin{pmatrix} 7 & 22 & 24 \\ 1 & 19 & 9 \\ 4 & 9 & 4 \end{pmatrix}$$

If we use this matrix to decrypt the ciphertext, we get this plaintext:

`TGUCIOEGKKIEEEQAHILLCIPHTPYELLRGGFTWNNG`

Notice that the only intelligible part is the appearance of the crib (and not even in the right place, as we shall see). We continue and eventually get to the point in the ciphertext where we construct this matrix:

$$\mathbf{C} = \begin{pmatrix} \mathrm{O} & \mathrm{F} & \mathrm{T} \\ \mathrm{A} & \mathrm{D} & \mathrm{S} \\ \mathrm{D} & \mathrm{Q} & \mathrm{T} \end{pmatrix} = \begin{pmatrix} 14 & 5 & 19 \\ 0 & 3 & 18 \\ 3 & 16 & 19 \end{pmatrix}$$

(determinant 11 mod 26; invertible). From this we get the key matrix

$$\mathbf{K} = \mathbf{C}\,\mathbf{P}^{-1} = \begin{pmatrix} 14 & 5 & 19 \\ 0 & 3 & 18 \\ 3 & 16 & 19 \end{pmatrix}\begin{pmatrix} 3 & 21 & 11 \\ 25 & 2 & 8 \\ 8 & 20 & 15 \end{pmatrix} = \begin{pmatrix} 7 & 8 & 11 \\ 11 & 2 & 8 \\ 15 & 7 & 4 \end{pmatrix}$$

With this key, the plaintext is the correct one:

`THISMESSAGEWASENCRYPTEDWITHAHILLCIPHERX`

Programming tasks

1. Implement the attack.

Exercises

1. Break the ciphertext from Exercise 2 of Unit 126 with the crib `FOLLY`:

```
GHTSYUESFCYMNNBGCGESCNGHAXDULTNVXOYRWPWTKPGUNKAWNPDVESBBRFN
TRNBGOWXLWTYNDPNSDLEYGUBBEHYQLAGOGJCNGIYUWTEULBBBREYVZVBGCG
ESAEDNCQLHDLZZGRXOYKGUNKAWNNNSCNYBYCOLYRYWLIAWNLXOGULXGBDNL
GYTAEXWGJDLEXYQFBWDDLJHBFFXNNNSDLERDDLHQGXREYWTKBWQDGGPGULN
GJDNBBREEXWKMTOIZBTRMUDPYSWXYQNSUWWLYDMOUUGSYJAMMLAYMOUUGHT
CWTKVDNPDOLYUXRCILNGJESYVGVKEFWDNEACILWLZNXTCGYWMEQNQWKZBCI
LULJUVDAOCWGEOGUNKAWNSGYYNGIXWYOXLNSXRCXANFWGGYBGHTCWTKMYJL
JUSGIDVUU
```

2. Break the ciphertext from Exercise 3 of Unit 126 with the crib `PITILESS`:

```
GQTIRDBVCMCVUCJCPZFCSPEOFOPRBDTOQGQAHWBXNBWLEWJCNUBRMRETIEV
CJSKZLCUDWHWBGESEPEVBRGXWCHDQSFAHWNRUJMHTRRPAAZGYGEZTDGDLEL
CSEMNLOCGESEPMNRUAFKGQTUHGHSFCPEIYGHCRDCSIHWRTIOJVKAXSRIGSF
ZMKNAZWMVIRDVUHYDBPEUPEVRBDOFSJVCHSFLCSLWFRBDREQCRUKPMFDHAH
OEMNZRJQLYYGHGFZSQHLAQBPQTQHGFSLQBWKHOFNMACHRFGXEKUYYDBFYZV
JGATEDYGGXERWCPPKGISLAALBOCSTCIYIVFMKPUSKCEGFYWUWSQMAYALQCE
KMWWFSNSWMCVUCJBQDUWCKLSYDBTUNDKPXMKEIZAFKIBRDXAGKE
```

Unit 130
Affine Hill cipher

The *affine Hill cipher* is an extension of the Hill and affine ciphers. We take the Hill cipher and add a constant term to the equation. Of course, since we are dealing with actions on vectors, the constant term is also a vector. When the key is a matrix **A** and vector **B**, a block of plaintext in a vector **P** is enciphered to a block of ciphertext in vector **C** with this equation:

$$\mathbf{C} = \mathbf{AP} + \mathbf{B}$$

Decipherment is done by solving for the plaintext vector:

$$\mathbf{P} = \mathbf{A}^{-1} (\mathbf{C} - \mathbf{B})$$

This cipher can be factored into a Hill cipher followed by a Vigenère cipher. This should be obvious from the first equation above. Furthermore, if in that equation **A** = **I** (the identity matrix), then the affine Hill degenerates to a simple Vigenère cipher.

Programming tasks

1. Implement the cipher. Remember to check that the matrix is invertible.

2. Modify your brute-force attack on the Hill cipher to accommodate the affine Hill cipher.

3. Modify your dictionary attack on the Hill cipher to accommodate the affine Hill. Use one keyword for the matrix and one for the added vector.

4. Modify your attack using linear equations on the Hill cipher with cribs to accommodate the affine Hill cipher. Each equation will now have a constant term on the right-hand side. There are n^2+n parameters for which to solve.

Exercises

1. Encipher this text with the given matrix and vector.

$$\mathbf{A} = \begin{pmatrix} 21 & 3 & 0 \\ 8 & 8 & 19 \\ 13 & 2 & 4 \end{pmatrix}, \mathbf{B} = \begin{pmatrix} 5 \\ 7 \\ 9 \end{pmatrix}$$

After marriage arrives a reaction, sometimes a big, sometimes a little one; but it comes sooner or later, and must be tided over by both parties if they desire the rest of their lives to go with the current.

(from “Three and—an Extra” in *Plain Tales from the Hills* by Rudyard Kipling)

2. Decipher this ciphertext that was enciphered with the given matrix and vector.

$$\mathbf{A} = \begin{pmatrix} 8 & 19 \\ 21 & 18 \end{pmatrix}, \mathbf{B} = \begin{pmatrix} 23 \\ 11 \end{pmatrix}$$

```
VSWEZMIFVTXDROVWQYJVVKQRZMVSWFWQXZJFFESTHETMQRQYXQHCIDBCTUN
NOPNPKLNBJIIFRUQDVPVAGBEYDFUPWQWJNVWXTMHCWFXQXJIFBLZSLZNUIS
VTXDROVWQYJVVKQRPINCFJHDXQGPBLDWONQNTUUCKLOOFBWXHDMPRVDRCAG
WBXDRCABRTMNBHICAUBQRMFOLZWBCLZZMNZHUMCSLOLQNBVCYOLCLJWRJKL
ZMJGXUPONJGNPBKLVVMJDQHEGANJNAIDNAHEKLOOMFOLKRWVBEWXUPQRTQA
INTDWWDKGBXXQST
```

3. Break this ciphertext with a brute-force attack. What are the keywords?

```
HFPQQDGRZVKFKUEMZTGEUZSIUVNGRFBCWNHNGBGWUOZAGEGIWZGCPKNZBHZ
RBKEJTZZFXXWNSDZCUKSQUKGUTXZRGOGWWIIHGEGIPQPXWZKMOEJRPQWNXP
GKZZHBUUBDHRZJSYMGPNTZKMZIGEAZCQBFZVTZKBZIGEGMSVHCHNWTHPDAF
PBRNPGTZGJRLZAZZYSYYAHNSVEKHPPQPXWZKMSCHPCVZVZVTJJGKGZDGESI
GTPZHBUZJOTHZVCNAFGVGSSMAMCBZGHNPDWZGOJOHNGBWAHPSMDEFBDPHNU
HUCNSGQHNKMIAUFNSKGSUUZGOOMQDUXNGTJWG
```

4. Break this ciphertext with a dictionary attack.

```
QJCSKYUYIUASSRXELHDERHZYKTHNQMJDIHYIAKLZBCKYQXYMVNKILMCFIRY
OZZILKMCTRNDAONWVAUPAPJKDBYLWKQOLYHNSKYOUGDALEKJTJNMQJJTFGJ
KDOWNXOAWLOGTXTEDSLLUNTQDZNNSGWLDQGQQTENQTTUQCOVQULXROLPWCO
UGDEJWZASZTOQYQOESIIWYXPPOAIWHQLZQHSLQRCVYJDGMTMOGPIBRDYJFA
VGWDTOAWJRSDATSAIEIHSEQDIXBHTDMTDAIYKTLSVKAQGFYYWRNAYWYKTJQ
BSCQFJYGRATUGWLKLAUQ
```

5. Break this ciphertext with the crib `AFTERNOON`.

```
KGMJEIVFJJGJHCKOGBRDZRHBUETXPIWNHUVFEWPTHHDXODTKUCEWUOCITNO
LEYCEUOUNHFRFKIXNPXIGEYKOCLHUVFPLWMETRIVFXZDGUHCVHYWKTGCOUR
TTPHEDHYDZPBVHWNNPELPUWFHYKOGJTLDXTQDLMHHYSXXNNNSXHHTGCUWFH
YMEKGCBDPICETHUVFEYWFDICUETRIVFVRDGEMPTXZHHTGCTPXVTNDMLUVPL
EFTGCKKNDIEDHYNDMWTGCNPVEFQYWXNSVTTQUTCXTGCIUZCLRFTKCHTZVFH
NTEUREPEIVSWFGGHFPAVCWWSTPM
```

6. Break this ciphertext from the 2013 British National Cipher Challenge with the crib `TELEGRAM`.

```
HCHKK MEYXE YMEZW RLFMS RPJSU BPWHR PMDEY XWHRM IQKEC JNBCM
CJFHY HSELD JXTSH IGEXH BMBIB GJRCN VAQJX FJYHL LQPAF DJHAO
```

NUCJF LRXTS XKTNQ JATUU VBGSH OHKOC VXHRZ WWHFY EEZIU DSJFJ
IBGQK XSUIN IFDPI HRXVZ BGGYT SXTPV NETDB QMZAT SXRDW HLXKT
EIWAZ WRTSX JWHSH VSIGE YRKBZ WJQYI UVPTU VFMPR SOTWI DTEXL
NMKNB YYMTV JIJWR JQFLZ WJDHM LCVXH RZWWH FYEQH XMIQK ECUZK
PJSBF KXVAY NZSUY AJRRJ QRDWG DHENG CMRNR RBOVU FDUAE PRGAU
QTBWN PLOJV RLSVP TNBPI KHOQL PFXOT FMCWM KIRFG YNVBZ AXRS

Unit 130½
Brute-forcing affine Hill ciphers with less brute force

In this unit we extend the attack from Unit 127½ to the case of the affine Hill cipher. Recall from the previous unit that for each block of text, the ciphertext vector **C** is found by adding a shift vector **B** to the result of a Hill cipher with matrix **A** as follows:

$$\mathbf{C} = \mathbf{AP} + \mathbf{B}$$

Here **P** is the plaintext vector. Decipherment is the inverse:

$$\mathbf{P} = \mathbf{A}^{-1}(\mathbf{C} - \mathbf{B})$$

Now it looks as though we need to already know all the shifts in the vector **B** before we can apply the attack. But let us rewrite the equation by distributing $\mathbf{A}^{-1}$ and moving the constant term to the other side.

$$\mathbf{P} + \mathbf{A}^{-1}\mathbf{B} = \mathbf{A}^{-1}\mathbf{C}$$

The product $\mathbf{A}^{-1}\mathbf{B}$ is simply a new constant vector, which we can call $\mathbf{B}'$. Its entries do not depend on either the plaintext or the ciphertext. Our equation now looks like this:

$$\mathbf{P}' = \mathbf{P} + \mathbf{B}' = \mathbf{A}^{-1}\mathbf{C}$$

where $\mathbf{P}'$ is a "plaintext" vector that actually resembles a text enciphered with the Vigenère cipher, which you recall is a collection of Caesar shift ciphers. For each row of the matrix **A** there is one shift that can be thought of as having been applied before the Hill cipher or after it.

So our strategy is this: Apply the row-by-row attack from Unit 127½ using a method to recognize text that has letter frequencies that resemble English, but which has undergone a Caesar shift. What we do is to try each possible shift and measure the monogram fitness for each. This adds to the running time of the algorithm because because it adds another loop nested inside the others. We need to keep track of the best row vectors *and* the best values for their shifts that we find. Note that the index of coincidence will not work as a way to recognize shifted text without having to try all possible shifts; this is because some row vectors will give nonsense that has a high IoC, and we are led away from the true solution. To be clear, here is the algorithm for the first stage of the attack:

```
set up storage for three 3-vectors and three values of fitness and three shifts
for v1 = 0, ..., 25 do:
        for v2 = 0, ..., 25 do:
                for v3 = 0, ..., 25 do:
                        use row vector (v1 v2 v3) to decrypt a slice of plaintext
                        for shift = 0, ..., 25:
```

```
                    apply a Caesar shift to the slice
                    calculate the fitness of that slice
                    if that fitness is higher than the lowest of the stored values:
                        replace the corresponding stored 3-vector with (v1 v2 v3)
                        replace the lowest stored value of fitness with the new one
                        replace the corresponding stored shift with the new one
                    if the three stored values of fitness are good enough:
                        exit loops
output the three stored 3-vectors and their corresponding shifts
```

We take the best 3-vectors and use them to generate the three slices of "plaintext," then apply the three shifts. The three resulting slices can now be used as we did in Unit 127½ as the ciphertext of a columnar transposition cipher, which we can break by brute-forcing its permutation.

Programming tasks

1. Implement the attack. For the second stage, in which the transposition is solved, use a brute-force method; for small matrices, which means a small number of columns, this adds very little time to the attack.

Exercises

1. Break this ciphertext:

```
LJSYNIHFXXZKNEWGRXTQCTHMLGJPVPLFRKPSDRDKUXFDRWCKMVDXVGEOERVRGPTD
BYKKHDEBTJJWUFMADKCUGJDYATXUKMLPOZFRBGOUMYIMJLOMUASMDJIMLJJUSXZJ
EVZYKHLOWHHIUBRULTNOVRXEJTJWQWSFHTTVFCBTJQCADLOPTPPETTEFNQNSTSDF
MLPIAOXVCENFZFXVCZKWSDOBMACHZGZZWAPYJOKZMVENZITNQFHHZWHXSKYWHZIP
XUPPECVRLRRWQJQUPLVQYRFFMWJXODOBSPJFTSVJQCKDNCNMACPWHHDCQMIDMZFM
WOUZZIDHAROPTDBYWKTVVGHFOKLJACWACLSTVVOKHTGOSFIEXGZNIABGJWZLENNU
MACGQFABSXRGCWYHQCJFPHIUCCLWTXFCXXUKQKRFKHHPYBNYEYQMCBCYCMACBFFG
WWYXFBHMRYNTTIZGYXAXDFWQKYFHTOWGDOBAUUAWEPMNDPQWTXFBOBLKFFSBQGMA
CHXEDJACVDRTYLGM
```

Part VII
Stream ciphers

Unit 131
Stream ciphers

A *stream cipher* enciphers characters of the plaintext one at a time. It generates a *key stream*, which is a pseudorandom stream of characters. Each character of the key stream is combined with one character of the plaintext to give one character of the ciphertext. To decipher, the same key stream is used to recover each character of the plaintext from a character of the ciphertext. If the key stream is generated independently from the text, then the cipher is called *synchronous*. If the key stream depends on characters from the text, then it is called *asynchronous*. If that dependence is on the last few ciphertext characters, then the cipher is *self-synchronizing* (or *ciphertext-autokey*), because the receiver of the ciphertext can resynchronize after reading a few ciphertext characters.

There are three variations on how key-stream characters are combined with plaintext characters. Each corresponds to one of the polyalphabetic ciphers that use unmixed key alphabets.

- Key-stream characters are added to plaintext characters, as in the Vigenère cipher. This is the standard variation.
- Key-stream characters are subtracted from plaintext characters, as in the variant Beaufort.
- Plaintext characters are subtracted from key-stream characters. This is similar to the Beaufort cipher.

Whatever information the encryptor of a stream cipher stores as is goes from one plaintext letter to the next is its *internal state*. Typically, the key is used to generate the initial contents of the internal state. The state changes as the encryptor advances through the plaintext. The key stream is generated from the state.

In contrast to stream ciphers, ciphers which act on a block of text at a time are called *block ciphers*. For example, periodic polyalphabetic substitution ciphers and permutation ciphers take blocks of a fixed number of plaintext characters and encipher each block in the same way.

Reading and references

Wikipedia, en.wikipedia.org/wiki/Stream_cipher

Exercises

1. Decrypt this ciphertext which was encrypted with a synchronous stream cipher, and the key stream is based on the Fibonacci sequence. (The Fibonacci sequence in modular arithmetic is periodic, by the way.)

```
ZPWFFVUVDHXCQRRTYGDYHFWECMWAHGIUWVZBCDLXHTGZNLIVNSVQWMUVGCT
SFPSZOXPHMYKAAFVLJSYBTSIAUMGDUANILUFBRBHDCDCJPJGLADGFFSBVLJ
QWOKGOFSSNWLPSYQLETFJZSXWYQZTCZMLUQEHZLMWSQBZPVRUAZTUWNTYVV
GWSSFKNFNJYHNSQHSCVUOIKGPFIIGEBLERBYKWWCQSVTPXVHIBEADNBRQEF
DNSQGBZDTSRBKSJSBTHUJAUVALQBNBZDSKYULFDOSVWGIWIEWABZZ
```

Unit 132
Trithemius cipher

The simplest stream cipher is the *Trithemius cipher*. It is synchronous. We can look at this cipher in three ways. First, we can say that it starts with a shift of zero and adds one to the shift with each character. Shifts are applied to plaintext characters modulo 26. Second, we can say that the cipher generates the key stream `ABC...XYZABC...XYZABC...` which is added to the plaintext to get the ciphertext. Third, we can view it as a Vigenère cipher with key `ABCDEFGHIJKLMNOPQRSTUVWXYZ`.

As explained above for general stream ciphers, the Trithemius cipher has three variations on how key-stream characters are combined with plaintext characters. In addition, there is another variation in which the initial shift is taken to be nonzero.

Like the atbash cipher, the Trithemius cipher has almost no security. Once the adversary knows the cipher, s/he can decrypt any ciphertext.

The internal state S of the encryptor is a simple counter. As we have done before, we can think of the plaintext $P = \{p_i\}$ and ciphertext $C = \{c_i\}$ as each a series of integers. In these terms, the action of the encryptor for the standard version of the Trithemius cipher is

1. Set $S = 0$
2. For each p_i in P
   a. $c_i = p_i + S$ modulo 26
   b. $S = S + 1$

Reading and references

Wikipedia, en.wikipedia.org/wiki/Tabula_recta#Trithemius_cipher

Johannes Trithemius, *Polygraphiae libri sex*, Reichenau: Joannis Haselberg de Aia, 1518, www.loc.gov/item/32017914, book 5.

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, pages 135-136.

Fletcher Pratt, *Secret and Urgent: The Story of Codes and Ciphers*, New York: Bobbs-Merrill, 1939, chapter XI, section III.

Programming tasks

1. Implement the Trithemius cipher. Give your functions some optional arguments to allow for the three variations on the method of combining key-stream characters with plaintext characters, and for the possibility of a nonzero initial shift.

2. Write a function or script to break a ciphertext that was encrypted with some variation of the Trithemius cipher. Use tetragram fitness of the plaintext to determine when you have found the right variation.

Exercises

1. Break this ciphertext:

```
PVPJXYQFHGYKZXGDQPUZSDCXKMZPVJCYAJGIEIEHHCEPUZYIQPIFTKBXYVD
RTJWVAGUWAKCCIDVLRJGIJNISQFHPOVVMBZNBYMDUCXKMZAPNWZZXGWENLH
JVNQWCOXPDRIMLBFSETGVTDLFLCICOYSIUWGIKUYPRCSQYDLYKZFIMWICSH
FAHPJIAMMRTYNPQEKLHIHQZFOASVBMSMSMJNEAUWHTVSMBVJAFDWBUEYWOL
LJDETROQRVQ
```

2. Break this ciphertext:

```
HSUFOWQFNYJLEJOTVJGDYNRPJITVKKLXSFFZIMKWMSCVVLZBRBLXAGQKQVC
PQXZLKAKUMUPZSSMFKIWNTJFRXTGCFLVGIZWWUHVQXDBXHRWFCUCAKYWWGZ
ZJCUBDPYELCGWIAMWEOPINSXZNKHJPPGCDEJKXMUFQSEQNYGQYGUOXFFVBV
XMWJXDGQEODQTATXBNWILAIEPNQNDAISXUPECGGOWMYUVILISKSNRYMCMTR
WFYDBPGXDWGCESDJQGCBRZFBORLTTVHQZNYWKKLNRCEFSTSJPTHBKSDRHLZ
YQFKVOHKSQOSVRPGTVZWPHOQCKZYLMTVHKWFDLBRKVXNGRTFRMBAYIEDUFH
VVQPYEWSUICGRXDVNLPOMCSJMBXFQPYSWKGTPOXTTNCREWMYKKEYRCECMMK
MYBNWHTJSQCKYICWJKRTF
```

Challenge

```
TAEVZRVZLPSLZAERFZRVDQMMUNVIOPNGNRCKDWHLOCQHZNFMQMQDWXXZJCQ
JQUNNOZDKKVAHVKSFLDKOAIOVOTBORRESLSWNXBZNAIBPGGZZMKBBUYYZNB
LZEVPAEWBPGYFZSLTGMBFTITXRGRZJNGKVZRTHWASLSASLTFJBIMRFZMPPA
IBUFJKYRKVZNAIBXEMOVXBSLLTGRPLLDQEXEIZDEINBSWWJMDRLYBUBFSWY
IWBUBFGUNGRVRRJJYYNRITTUYJFNGNGNRNBSVNQHPCMYCDHHBUCHBMXVKBJ
WPTWKXFYGTEEVBFQUNGKBT
```

Unit 133
Autokey cipher

The *autokey cipher* (or *autoclave cipher*) uses a key stream that begins with a keyword which is followed by the plaintext itself. Hence the name of the cipher. It is asynchronous. In the standard version of this cipher, key-stream characters are added to plaintext characters, modulo 26.

An example will help to elucidate. Let us encipher this short text with the key `AUTOKEY`.

```
THIS TEXT WAS ENCRYPTED WITH AN AUTOKEY CIPHER
```

The key stream is written under the text, and the ciphertext under that is the sum of the two, modulo 26:

```
plaintext:    THISTEXTWASENCRYPTEDWITHANAUTOKEYCIPHER
key stream:   AUTOKEYTHISTEXTWASENCRYPTEDWITHANAUTOKE...
ciphertext:   TBBGDIVMDIKXRZKUPLIQYZRWTRDQBHRELCCIVOV
```

If the length of the key is L, then the internal state $S = (s_0, s_1, ..., s_{L-1})$ of the autokey cipher is the last L characters encountered by the encryption routine. The state is initialized by filling it with the letters of the key. For each letter in the plaintext, the character of the key stream is found by shifting the state vector to the left. The leftmost character is popped off the left end of the state vector to become the key-stream character. The current plaintext character is pushed into the vacancy at the right end of the state vector.

Let's rework our example in the language of state vectors, just to belabor the point. The state is initialized by filling it with the key:

$$S = (\mathsf{A}, \mathsf{U}, \mathsf{T}, \mathsf{O}, \mathsf{K}, \mathsf{E}, \mathsf{Y})$$

The first plaintext character is `T`. We push it onto the right end of S and pop off the first key-stream character, `A`.

$$S = (\mathsf{U}, \mathsf{T}, \mathsf{O}, \mathsf{K}, \mathsf{E}, \mathsf{Y}, \mathsf{T})$$

$$k_0 = \mathsf{A}$$

The next plaintext character is H, which we push onto the right, and pop off a U.

$$S = (\mathsf{T}, \mathsf{O}, \mathsf{K}, \mathsf{E}, \mathsf{Y}, \mathsf{T}, \mathsf{H})$$

$$k_1 = \mathsf{U}$$

You get the idea. This continues until the full keystream is generated.

There are three variations of the cipher that correspond to the three variations of combining keystream characters with plaintext characters: Vigenère (standard), Beaufort, and variant Beaufort.

Reading and references

Blaise de Vigenère, *Traicté des chiffres ou secrètes manières d'escrire*, Paris: Abel l'Angelier, 1586, HDL: 2027/ien.35552000251008, gallica.bnf.fr/ark:/12148/bpt6k1040608n, gallica.bnf.fr/ark:/12148/bpt6k94009991

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; chapter XVI.

Fletcher Pratt, *Secret and Urgent: The Story of Codes and Ciphers*, New York: Bobbs-Merrill, 1939, chapter XI, section II.

Wikipedia, en.wikipedia.org/wiki/Autokey_cipher

Practical Cryptography, practicalcryptography.com/ciphers/autokey-cipher

American Cryptogram Association, www.cryptogram.org/downloads/aca.info/ciphers/Autokey.pdf

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, pages 147-148.

Programming tasks

1. Implement the autokey cipher. Allow for the possibility of choosing any of the three variations of the cipher.

2. Implement a dictionary attack on the autokey cipher.

Exercises

1. Encipher the following text with the keyword `STREAM`,

    a. with the standard autokey cipher (the Vigenère-like version),
    b. with the Beaufort-like variation,
    c. with the variant-Beaufort-like variation.

    The deep hollows which separate the hills are thickly covered with fern and heather, over which blocks of granite are scattered in all directions; and, as in all similar districts, each valley has its own clear mountain stream, which receives the innumerable waterfalls descending from the hill-sides.

(from *Devon: Its Moorlands, Streams and Coasts* by Rosalind Northcote)

2. Decipher this ciphertext with the keyword `RIVERBED` and the standard autokey cipher.

```
KPZWYFIUKVGCZEGIFGUDZEUKFWSFJYLHWJFDQGMVJQNXVFPLVPOXLRWRUK
XZQPIFLTXCBZVTIAQOEWACAAUNDKNSDFDASIOFUKTISHSBFUOMCPVGCWQS
SXKASGPVWRYARQFGEOIXGLWEXOYEUSIFSSDRJYLHZTQKIOSSAGZXCPYGQC
RFNSFVMCNOPWKYSIRFLSVTKVHLOEDPKQLGHXIPBANQELWZNHNRRSBOGTDS
GHRYGLGIRHWOOYJPGFWESFQXMQEYGZGXC
```

3. Decipher this ciphertext with the keyword `FUTURE` and the Beaufort-like variation of the cipher.

```
MNPWKEQCQEUXKYKMWTPJLUYDAMVWADNOKYAQMFGIPNLAPFADSFZWBLOVBRM
AGATHOUBPJJHTPRDUSCZZVDASTVPLXYYMXWEGYHPWAJQWVWLAINXQDAYLTZ
WMERQPBDRW
```

4. Decipher this ciphertext with the keyword `ELECTRIC` and the variant-Beaufort-like variation of the cipher.

```
EPIGUCJSLHHZRZUGCXVYPPTMFOFZYDAMAVKWVVKTCRATCJNNWPELOALICUW
OMROBYBKMICOVUIVOITZVJVSDAEALTPHTNWFUTSHGAOEFPCEKOAVGSNIPZM
WKLBOSDAEATQZDNGQQQPBGIBBA
```

5. Break this ciphertext with a dictionary attack.

```
IVIDVWYYSWAFNTVFDLJQVDAELSGSMSXEEMPWGRGWPQTPOFEACINUPHQYEPX
MZSVVHPYXBROOHKVWOXNYIXWWLASKMEYEWXLCUPBKAWSHTALFWBMJGJXMDB
ZWYWDEMZMPAJDVTBQJTHNXQJTHVORWGUIYGKLHSPXKPIMUFPNWNLPBYWUXX
KXQPRGLGVWATPKGSOKEEROOXAAQQNRWSUZTRWSENRMVPASQHBDJSAFRJYHI
EJCHDEEXTBOXFJQRKTGJQQHLWSAPPTWIWV
```

6. Break this ciphertext with a dictionary attack.

```
PVALSAXRIUTGBRNFNMPKKRATZLWEMOACVIFCJDVZNABCNIMLXYIAHSOGPTE
AGKLSKMWGSVBKYKTCSDPIRSSWKPJTFHCEXGSVQTBCHQJVZGEWCVCICDXYBQ
LKRWLXJMKDWRKZVSKXOPVXQQCWKFAWDKHJTDNOULXXNMGPVRQUZFBZRLMVG
DNPBHTYWQNNAABKFLATGRIGQHTCHGDHDETMWSEPSSAJZERDTHXWTMRUQ
```

Unit 134
Hill-climbing attack on the autokey cipher

This is a modification of the hill-climbing attack that we used against the Vigenère cipher. Given a key length, we vary each character of a key in turn until we cannot increase the fitness of the decrypted plaintext any further.

The algorithm follows. The purpose of the flag is to exit the loop if it has run through all characters of the key but not improved the fitness.

1. set the best fitness equal to the fitness of the unaltered ciphertext
2. choose a random parent key with the given key length
3. set flag to FALSE
4. while the flag is not TRUE
    b. set flag to TRUE
    a. for *i* running from 0 to key length minus 1
        i. copy the parent key into a child key
        ii. for *x* running over A, B, ..., Z
            - set the ith character of the child key to *x*
            - decrypt the ciphertext with the child key to get a plaintext
            - calculate a new fitness of the plaintext
            - if the new fitness is greater than the best fitness
                · set the best fitness equal to the new fitness
                · replace the parent key with the child key
                · set the flag to FALSE
5. output the parent key

Reading and references

Practical Cryptography, practicalcryptography.com/cryptanalysis/stochastic-searching/cryptanalysis-autokey-cipher

Programming tasks

1. Implement the attack. Allow for the possibility that the cipher is one of the three variations. Feel free to copy and modify your code from the analogous attack on the Vigenère cipher.

Exercises

1. Break this ciphertext:

```
KPDWZTEEHVSSJGUUHVOTIWHZLALTAIEKBHSLRAEHFSEDRRRDIWTFWGSNPKA
AGSWGFZEFDQLTNSXZEGKIYRLALTAAJOETUDPJEOSLRLJKPKSTAWAWLVUNUC
HOMBMYFSGLAWYVOEETZQSCYEDBRIBBPXSMSQSBLWALMQAFLRNRRSHFJZULB
IPTSPINDUGWALMQAFQRRVYRZYWVMGYGKHLTODPCKZDKCSTURCPLYOWWLMIV
JHZTBOPMTRPGAHFWKZRYNFYWRONRJVREAJKERAGHRFYHBPKFEPRUYEKFEGK
IYRLPRBAIYSGAGILWPXZSLTZACVYOOXHGFYGVAVTJLWFWEEWALMQAFYROLI
ZPWJHCMRBXYYMPAJKMAHXGHWHWELMLIHHVEVMNFZEFWSYIXVDNPTUVOPDKQ
OAGHDFKPHSTRUMACMGUEVHNUTOCCTONWKAVOECAEDTUHRHHPCMOMBZZXYTA
MIKJRZEGWDFCCHIEKWOHECXJLCQKAAWLASBZOIHTSPRSSAMSSJILSUTWDFD
PWKPMJAVPEVXMETUHPHXFVAFWLALBXCJTOEPNMJVMQOIAWALQVLRQVMOEUP
FSXKVMLACCJKCELHBRMGHZXGGVMNF
```

2. Break this ciphertext:

```
OTDBPBEBOCNBUEMOEGMSNQNPNYUMXWKPPMDSDEHZMDMTRCNAYAMGUJUHBTN
LMKPQIWJVEYNRMGWMUETBJPVLEQBHPZARJJYUXWZQQYXSAOTPRZPVEYOAUA
XXYDIBZNMDXEPPTJGFLGOFZSCCMETIPZZTTDHHICGDNQFSOCYNIWKEHWSTD
VMKZMYWGNMLNYLCFSQLCCKDCWRGYKHOHZEJBTQJRWUKBODTYIYDTSMNUSHS
NLPKFNQCCUVBQORDZSTEVJXZAWOALSILAIUEDF
```

Unit 135
Attacking the autokey cipher with monogram frequencies

This attack is a modification of the attack on the Vigenère as collection of Caesar shift ciphers. For the autokey cipher, we are unable to find the key length from the index of coincidence, so we will have to try various lengths until we find an acceptable plaintext. We will partition the ciphertext into slices, each of which (if we have the correct key length) has been encrypted as an autokey cipher with a key comprised of a single character. For each slice, we try the 26 letters of the alphabet as the key and choose the one that results in the closest fit between the monogram frequencies of the decrypted slice and the monogram frequencies of English.

Programming tasks

1. Implement the attack. Allow for the possibility that the cipher is one of the three variations. Use the cosine of the angle between the vectors of monogram frequencies. Feel free to copy and modify your code from the analogous attack on the Vigenère cipher.

Exercises

1. Use this new attack to break the ciphertexts in the exercises of the previous unit.

2. Break this ciphertext:

```
PLEKLARLUSGSOSAOWAWLQWGBLSBLUFPAVSGNAPDLIPLHTGBCARDAQEIVVJG
VQHVMQOVGKIPHISELWDWHWFFTWJFRHIPYIGNGOSGFZABUKNIKFQAIFXBFVR
TXKMPBPUPPRVSCZEVQAEUXZBXNHWXZHBOAVHWNKALQDCGEJWOFRMSRMNFRP
VNMGJOIMAKMYAMNHXMOEKTOKWVSISRXUQWJITEJPWOSQTGSHDSKNSXMPSDC
ASLOZNLKFAZTNHJJZEQXIYPFYOGEOVGRYLENSSPDNSPKHTWJMNHAMCYMOSL
HIRORSMOKSMHBUDMVQGKMAGNLQAQQAZOSDEEGPPPZGDMSQEPTPILMWMVVVT
VVFVQZSVK
```

3. Break this ciphertext:

```
MZHPWWXIFYIIOPPVGLADUZNZRQWWVJDBJRRNIWAYAKRXRULWRCZLUYCKLLO
WLMLAWNUOECNAKGZMONJAQZEJAQEOOGFZEHEDZPIRGLLACCYZKVZLUYXUAI
KLPFXEKKOHZVBDRJUVIPVWEZDCLGOULRRYQIUAWARYMEHWKLWOTAYFOQWJP
GYTLQIEFRTOSSWGUVNNGODRTGGJMKPMASCXAWYIBAQEJXJCBGORUWRDUBPW
CGWNKBNPXNEKKRRVRJRDWXRUPLLOWLYQEUQXVRWFSABAMQFKGBACPMDIVTC
HGRXWHMPXASWDLWIZXAXOQZMZARMSAIXFXQUPHJOTKUFKGVKEXKWMQUWYBD
DYXPW
```

4. Break this ciphertext:

```
MEKXVYDNJQTQFAFWUXIUNNLJTADQMAGWBDTNVGONJNOAHNMRDZCEUSSCSEL
QTZAETCPALNSBNPLZMYMKXDHZYNBYCOXFBOLFIUZPVWMWNAAPPKHLOCSFPN
HRHIEWAVPJUYEDNIHXMJUQKDTK
```

Unit 136
Running-key cipher

In the *running-key cipher*, the key stream is another piece of text, usually taken from literature. Sometimes, it is a plaintext that you have seen before.

Reading and references

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, pages 236-238.

Auguste Kerckhoffs, “La cryptographie militaire,” *Journal des sciences militaires* IX (1883) 5-39 and 161-191, www.petitcolas.net/kerckhoffs/crypto_militaire_1_b.pdf, www.petitcolas.net/kerckhoffs/crypto_militaire_2.pdf, part III.

William F. Friedman, Methods for the Solution of Running-Key Ciphers, Riverbank Laboratories Department of Ciphers Publication 16, Geneva, Illinois, 1918, www.marshallfoundation.org/library/methods-solution-ciphers

Programming tasks

1. We would ask you to implement the cipher, but you can just use your Vigenère functions with a really long keyword.

Exercises

1. This ciphertext was enciphered with a key that is also meaningful English text. Recover the plaintext and the key.

```
SULTFZHWTFUQRFTVMXCZOAAUARHGPACVPRPOXZEVBNUHANEHMWGKVUDMCST
UHYVVEVFMNWZMJVJKLMLPALXPRLOFLNWGGYLHGLRXGGNAVTCUNEMDAEPZRB
ZAAIBXIPSBUZHRJKYHSXBFWPQPXKAVTCTTBDHTQYUKKMKOIWXXUHAZPGXYR
MJKARNNWPDFJDIIPXBEVMDOWVEWGZBUXNPJZPUNCMACRDNILMCANTHIAWAL
GFRBEVRRUZLGTJGOXBCSMLXYEPQRRCEUCAINXSEWXSWTRCJQEZAHXZRWHPO
KXLXMEIVVZMBGZJPLWLAWSIIPHKZAGIMGGLRXKUTGKICNIEBXYSIMML
```

Unit 137
Progressive Vigenère cipher

The *progressive Vigenère cipher* (also known as *progressive-key cipher*) is a synchronous stream cipher. It is a modification of the Vigenère cipher in which the keyword is shifted by an amount each time it is used, as with a Caesar cipher. The shift is called the *progression index*.

An example will clarify things. Start with our usual plaintext and encipher it with keyword `KEYWORD` and progression index 3. Shifted by 3, the keyword becomes `NHBZRUG`. Shifted by another 3 it becomes `QKECUXJ`. You get the idea.

```
plaintext:    THISMES SAGEWAS ENCRYPT EDWITHA STREAMC IPHER
keyword:      KEYWORD KEYWORD KEYWORD KEYWORD KEYWORD KEYWO
shifted key:  KEYWORD NHBZRUG QKECUXJ TNHFXAM WQKIADP ZTNLD
ciphertext:   DLGOAVV FHHDNUY UXGTSMC XQDNQHM OJBMAPR HIUPU
```

We can attack the progressive Vigenère if we know the length of the keyword and the progression index (or we can try values until we find them). With these two numbers, we can remove the progression. What remains is a Vigenère cipher, which we can break using the techniques of Units 51 and 52. For the example above, once we know the key length is seven and the progression index is three, we can subtract the progression as follows:

```
ciphertext:     DLGOAVV FHHDNUY UXGTSMC XQDNQHM OJBMAPR HIUPU
progression:    AAAAAAA DDDDDDD GGGGGGG JJJJJJJ MMMMMMM PPPPP
new ciphertext: DLGOAVV CEEAKRV ORANMGW OHUEHYD CXPAODF STFAF
```

Some special cases:

- progression index = 0: Vigenère cipher
- key length = 1, progression index = 1: Trithemius cipher

Reading and references

American Cryptogram Association,
www.cryptogram.org/downloads/aca.info/ciphers/ProgressiveKey.pdf

Programming tasks

1. Implement the cipher.

2. Implement the attack described above. You will need to try all possible key lengths and progression indices, until you find a good plaintext.

Exercises

1. Encipher this text with keyword `TARGET` and progression index 25.

   > We now take it so much for granted, we are so conscious of constantly progressing in knowledge, arts, organising capacity, utilities of all sorts, that it is easy to look upon Progress as an aim, like liberty or a world-federation, which it only depends on our own efforts and good-will to achieve.
   >
   > (from *The Idea of Progress* by J. B. Bury)

2. Decipher this text with keyword `REASON` and progression index 2.

```
ZRHMANGVTIWGZAWPZVOOLMLXQQISKIJGXQRXFQEYAKNFGZTVTYDBLDEEFQO
YDQIXMCNLQSSWXVQIQCMMCWHBLKPMUHPKLWTVQDKPGXMFQNOZOHSGLZHMAY
LCSGHHGQYIWZIVBQFVKXDGNIMYRALFYPGGQIUYIIAKCATTFNXTDVGNWRPST
EJEDMAUNZSSFGYIHTESMDTTXKEUABQBGRDHVZRJLQYRWYJIPVYDICNYOELV
EGTGAKUCJJVBMKFWOYSUBBBTIHADHVDGLBGLUXAKNLMSKHONVTSZCDWSSUC
MPQULATYCEUSDCDQTKLEJFRFKOVUGZLRKJDBKYSLXCZWRZZTWBGB
```

3. Break this ciphertext.

```
HKOTUMFYIJMSACEODTFEMRQHXNHOCXUARPBHNCZFIHLFDGCFHFAXBJWYIAE
JNQSFGDGUWKZNMXPMOQPPAITGTPJKJSGAYZZIQREBSTEFKIMWSWNSCNTRRK
XKYORQWGXRVGJUNYYOIMAGFZTFIWSUUOWXGTQFHZMJYQWCQEWACGGASQGHO
XMBXCQGVGFVOHRISHXQYYJCDMBQASRAIRQZZZWMMAYFHJDBMDGUTCHOETWC
KIOPUZAOAFUOOBZNDGZJXQWBVKYQQTLBEQUPUJIUWVGRTCAZQBDZSVDSLVX
QKJIXWLZYFMCPXELOYEOWYENKVASNAKMOEGPVCGTKFEKBBUHVQRDARSDEJH
LVRVMRBMSQQJWCSRXKAEHSUFIOUJGMPOSTXCZCEPTEKFXHNALWZVMMXEXNQ
SUVCXAUIILZUJTXAABIXOMHNONIVUZONHYUJECRLJPQJUYTFPVEMDLRVAXA
CNXVITBOWNENNVGREKCDYCSFZPZMCIUDMCKHUUYMBRKIQOESQWUNTVSIVDY
GWEDCYORQWUJSVLEKHSJFAVDAGERCJYEJVYXCYWUOWAXDYQSPVTGHXGJNZY
LHAOOUQPSJIPHKULHDQALOVVMVORRDQMGVCZHYIIKMFPLEDJQQGFKYORKUW
ZUVOGLKJYLEXZMDFQFXUGZEBOKOANKNRTLTXCBOGYQUKMRRIZWQOZTLGXAR
DHYJ
```

Unit 138
Solitaire cipher

The *solitaire cipher* is a synchronous stream cipher in which the key stream is generated with a deck of playing cards. It was invented by Bruce Schneier for the spy novel *Cryptonomicon* by Neal Stephenson.

Solitaire uses a standard deck of poker cards with two jokers. Usually, one of them is black-and-white and the other is colored. We will call one of them “joker A” and the other “joker B.” We also assign a number to each card in the deck. The clubs (♣) are numbered 1 (ace) to 13 (king), the diamonds (♦) 14 to 26, the hearts (♥) 27 to 39, and the spades (♠) 40 to 52. Both jokers are numbered 53.

We begin with the deck in numberical order, clubs followed by diamonds followed by hearts followed by spades followed by joker A and joker B. To key the deck from a keyword or phrase, we perform these steps for each letter in the keyword:

1. If joker A is on the bottom (the last card) of the deck, put it just after the top card. Otherwise, swap joker A with the card below it.
2. If joker B is on the bottom of the deck, put it just after the second card. If joker B is the second-to-last card, put it just after the top card. If neither of these is the case, move joker B down by two cards.
3. Do a triple cut by swapping the stack of cards above the first joker with the stack of cards below the second joker. The first joker is whichever is closer to the top of the deck. The second is closer to the bottom.
4. Look at the bottom card. If it is a joker, do nothing for this step. Otherwise, take the number corresponding to that card and do a count cut by taking a stack of that many card off the top of the deck and putting that stack just above the bottom card.
5. Convert the letter of the keyword to a number, where `A` = 1, `B` = 2, `C` = 3, ..., `Z` = 26, and do another count cut by taking a stack of that many card off the top of the deck and putting that stack just above the bottom card.

The keyed deck is the initial internal state of the cipher, and from it we generate the key stream. We repeat the following process until we have enough key-stream letters to encipher our plaintext:

1. Same as step 1 above.
2. Same as step 2 above.
3. Same as step 3 above.
4. Same as step 4 above.

5. Look at the top card and take its corresponding number. Take a stack of that many cards off the top of the deck. Look at the new top card and record its number. Put the stack back onto the top of the deck.
6. If the number you recorded in step 5 is 53, throw it away and go back to step 1. Otherwise, if the number is greater than 26, then subtract 26 from it. Convert the number to a letter, where `A` = 1, `B` = 2, `C` = 3, ..., `Z` = 26.

The key stream is combined with the plaintext in a Vigenère-like manner, i.e., with addition modulo 26. However, it is not exactly the same; there is an offset of one. The cipher uses `A` = 1, ..., `Z` = 26 for this process as well. To combine a plaintext letter with a keystream letter, convert both to numbers in this way. Add them. If the sum exceeds 26, then subtract 26. Convert back to a letter.

The solitaire cipher is difficult to break unless we know more than half of the keyed deck.

The official instructions say that the plaintext should be padded to a multiple of five letters, but we are not strict about that.

Reading and references

Bruce Schneier, "The Solitaire Encryption Algorithm," Schneier on Security, www.schneier.com/academic/solitaire

Wikipedia, en.wikipedia.org/wiki/Solitaire_(cipher)

Programming tasks

1. Write a function to generate a keyed deck of cards from a keyword or key phrase. Do not use the optional step in Schneier's description of the keying method on his web page.

2. Write a function to generate a key stream given a keyed deck of cards.

3. Implement the cipher.

4. Study what happens to the deck as it is keyed with a keyword or key phrase. Can you devise a method of recovering the keyword from the state of the deck? If so, implement your idea.

Exercises

1. Encipher this text with the key phrase `LUCKY YOU DO NOT HAVE TO DO IT BY HAND.`

   The best poker hand is the royal flush, followed by a nonroyal straight flush. Then comes four of a kind, followed by a full house. Then comes a flush, followed by a straight, then three of a kind, then two pairs, then one pair, then nothing. You don't want to have nothing.

2. Decipher this ciphertext from the 2010 British National Cipher Challenge. It was encrypted with the key phrase `DIE ALCHEMISTEN RISE AGAIN`.

```
AGXJE SSFKM MJMHX ZGJWB CPCVX EBNDK UQOCE DUTIC NPARQ PEDIX
ZAVYM WZSVT BBVMT HJIGW XZAPJ HJMYN MXRGO RXOWE ULMJS AAENC
WVUYI FQUTR XEDEJ BLWAA DFPBW ZAXJD DZOTM GSEZG NQWJY MFNWL
SLTQD URZVQ RKOTS VDNHY EIITY RRWGC CSLKS UKHDR LDBZE DXSGV
UGMTB NZQJT CBZTT IBWKQ PXUTQ MZDIH HKWZK SJEEH FBFYP GYSIH
OKKOB JFJSN XSIKS NBMTN IADVT CXYZZ AOKQY WXNIZ JWOFZ VSPQQ
GASYU MJDEL MDDHV ZTNFH MOOLN XAFPE VBHGS TJFMC IFNHZ YCYGG
WAQYB UNNHD WHSLP IBFAP PDNQN DOCNU RXEAI RZNLR XAKVX XAMPU
ZOPOK TJVQK IZDSC NC
```

3. Decipher this text with the deck 5♦, 6♦, 7♦, 8♦, 9♦, 10♦, J♦, Q♦, K♦, 9♣, joker A, 7♣, 5♣, J♣, Q♣, J♥, Q♥, joker B, A♥, 2♥, 3♥, 4♥, 5♥, 6♥, 7♥, 8♥, 9♥, 10♥, 10♣, K♥, A♠, 2♠, 3♠, 4♠, 5♠, 6♠, 7♠, 8♠, 9♠, 10♠, J♠, Q♠, 3♣, 4♣, A♣, 8♣, 2♣, K♠, K♣, A♦, 2♦, 3♦, 4♦, 6♣. Can you also reconstruct the keyword?

```
SCYWLWXCACDWWIFZSXIMHVMBRIWHCLNLMZIHHWWCHVOZNJCKPPALVNMGFNJ
RLCQFHDKNHZHKRAGIFXGKQQSLNEDGKOTOFRFNZAJOWWVZAPGGMLURKZGQDM
HEHKYEBLRUPMRPKFHFFMCIDKGYLOFLQQLSOMYAEGCPDYRWWJLTXRKQLOLXG
CHCSCQMDUKZWMLMKNTHMJQNVORTTIDRHQZBEIJCJNTMRTHYNVGAID
```

Unit 139
Hill-climbing attack on the solitaire cipher with partially known key

The approach of this attack is to fix the known keys and shuffle the remainder until the fitness of the first few (between fifty and one hundred) deciphered characters exceeds a threshold; then we begin again with the key we have at that point and try to maximize the fitness of a longer part of the deciphered text. For the maximization, we use a hill-climbing technique with the parent/child key paradigm. To generate a child key from the parent, we randomly swap two of the unfixed cards, or do a three-way swap of three unfixed cards. This is a difficult attack, and it may take a long time or need to be restarted many times. It is

Programming tasks

1. Implement the attack. Use tetragram fitness to rate plaintexts. Experiment with the fitness threshold and the length of text to use in the first stage.

Exercises

1. Break the ciphertext from Exercise 2 of the previous unit. The deck is 8♦, 9♦, 3♥, K♦, A♥, J♣, 5♥, 6♥, 7♥, 8♥, 9♥, 10♥, 6♣, K♥, 4♦, Q♦, 2♠, 10♣, A♦, 5♠, 10♠, joker A, K♣, 4♠, 7♠, K♠, Q♣, 9♠, J♥, Q♥, 2♥, 7♣, 3♠, 6♠, 8♣, ?, ?, ?, ?, ?, ?, ?, ?, joker B, ?, ?, ?, ?, ?, ?, J♦, ?, 7♦, ?.

2. Can you break this ciphertext? The deck is 9♦, 3♥, 4♥, 5♥, 6♥, 7♥, 8♥, 9♥, 10♥, 2♣, K♥, A♠, 2♠, 3♠, 4♠, 8♦, 7♠, 8♠, 9♠, Q♥, Q♠, 6♠, 2♥, 2♦, 4♣, joker A, 3♣, 6♣, 3♦, 9♣, 10♣, 7♦, A♣, 10♠, J♠, K♣, A♦, A♥, ?, joker B, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ? Can you also recover the keyword?

```
BHSJKOZKSQZBKVJLAICTFVGPJQJKMJCWBKTGMKUXWWIAPSCHXWUIPGHSPRD
SVSADSBHIAMWWRQCGWPSNIMKZXOXYKGCRHKWCSMGZICJYCJCWXTBKLTAMPC
FAQIIJTIWDLCPXFZKIRSVJSCKQATFLPZYCWJECXZYKIDBINPIFQJHWRECYO
DKTTYKHDYYULKEQFCGPGVGPQZHQAWFRMYUESBIDPSCIIGUJXXOKRSCEOLCH
IDXEQCGKBVNRLNNGIXIYKYGFYUITKKHFVC
```

Part VIII
Codes and ciphers

Unit 140
Codes

Historically, a code is a way of hiding the meaning of a message by replacing whole words and phrases with code words. But a newer meaning of *code* is a way of converting information from one format to another. Codes can be used as ciphers. The symbols of the plaintext are *encoded* in the ciphertext symbols. *Decoding* is the inverse operation. For example, the ciphertext symbols of Morse code are the dot, dash, and space. The encoding of a single plaintext symbol is a *code word*. We have seen another code already: the Polybius square, which can also be called the tap code or knock code (Unit 103). Its symbols were the digits 1, 2, 3, 4, 5, or the tap and the pause.

Two main categories of codes are fixed-width codes and variable-length codes. In a *fixed-width code*, all code words have the same length. In a *variable-length code*, this is not true. For example, Morse code is a variable-length code, and the Polybius cipher is a fixed-width code (in base 5).

Breaking a ciphertext that is encoded with a fixed-width code is easy for us. Once we know the length of each code word, we can build a table of them and assign a plaintext symbol to each. Then what we have is a monoalphabetic substitution cipher.

Variable-length codes have two subcategories: prefix-free codes and non-prefix-free codes. In a *prefix-free code* (also called simply a *prefix code*), no code word is a prefix of another. That means that no code word looks like the beginning of another code word. Therefore, if one knows all of the possible code words, decoding is unambiguous. In a *non-prefix-free code* (or *non-prefix code*), this is not true, and decoding is difficult, even with knowledge of all code words.

Reading and references

Wikipedia, en.wikipedia.org/wiki/Code and en.wikipedia.org/wiki/Prefix_code

Programming tasks

1. Implement the attack described above for fixed-width codes. The length of code words may have to be an input.

Exercises

1. Break this ciphertext that was encoded with a fixed-width code.

```
ADGJBDGJAEGJBEGJADHJBDHLADGJBDGLBDHJADHJADGKAEGJADHKADHKAEG
JADHKADGKAEGLBDHKADGKADHJAEGLADHLADHJADGLBEHLAEGLAEGJADGLAD
```

HJAEHJBDHKBEHJBDGLAEGJAEGLAEGLAEGJADHKADGKBDHJBEHLBDGKADHJA
DGLBDGLAEGJBDGLAEGLADHJADGLBDHKADHKAEGLBDGKADHJBDHJADGJADHK
AEGKADGJADHKAEHJBDHKBEHJBDGKADGJADHLAEGJADHKADGKADHKBDHKAEG
LBDGKAEGJADHKADGKAEGLBDHKAEHJBDHKBDHKADHKBEGJADHJBDHKADGLAE
GLBDHLAEGJBEGJADHJBDGLBDGKADHJBDGKADGJAEHJAEHKADHJADHJAEHKA
DHJAEHJAEGJADHKAEGLBDHKAEGLBDGKADHJBDHJBDHKBDHKAEGKBDGKADHJ
ADGLBDGLAEGJBDGLAEGLADHJADGLBDHLADGJBDGLADGLADHJADGJAEHJAEG
JADHKADGKBDHJBEGLAEGLAEGJAEGLBDGKADGJAEHJADHKBDHKAEHKAEGJBE
GJAEGLBEGLADGLADHJBDGLBDHKADGLBEGJBDHKADHKADHLADHJADGLBDGLA
DGJAEGLAEGJBDHKADHKBDGLAEGJADHKAEGJAEGLADGJADHKAEHJBDHLBDGK
ADGJAEGLAEGJBDGLAEGLBDGKADHJBEGLBDGLADHJBDHKBEHJADGJBDHJBDH
KBDHKAEGKAEGLBDGKBDHKBEGLADGKBDGKAEGLADGJBDGJAEGJBEGJADHJBD
HLAEGJAEGLBDGKBDHKBEGLAEGLAEHKAEGJBEGJAEGLBEGLADGLADHJBDGLB
DHKADGLBEGJBDHKADHKADHLADHJADGLBDGLADGJAEGLAEGJBDHKADHKBDGL
BDGLBDHKBDGLBDGKADHJBDHLADGJBDGLBEGJBDHKADHKBDGLAEGJAEHJADH
JADGLAEGJADHKADGKAEGJADHKBDGKADHJADGLBDHKBDHLADHKBEGKAEGJAD
HKAEHJADGJBDGLBDHLADHJBDGJBDGJADGJBDGLBDGLBDGKADHJBEGJBDHKB
EGLBDGJAEHJBEHJBDHKADGLAEGLBDGKADHJBDGKBDHKAEGLAEHJADGJBEHL
BEGKADGJAEHJADHJBDGKADHJADGLBEHJADHJADHJBDGJADHLADHJADGLBEH
LBDGLBDGJADHJADHJAEHKBEHLADGJADHKAEHJBDGLAEGLBEGLAEHKAEGJAE
HJBDHLBDGKADHJAEGLBDGKADHJADGLAEGLBDGKADHJAEHKBDGJADHJADGJB
DGLBEGLADGLADHJBDHKBEHJBEGKADGJAEGKAEGJADHKADGKADGJAEHJADGJ
AEGJBDGLBEHLBEGJBDGKADGJAEGJADHKBDHLBDHKBEGLBDGJAEHJBDHJADH
JBDHLBDHKADGLAEGLBDGKAEGLBDGKADHJAEGLADGLBDHKBEGLBDHJBDGJAD
HJBDHKBEHJADGKADHJAEGLAEGLAEGJADHKADGKBEGLAEHKADGJADHKAEHJA
EHKAEGJBEGJAEGKAEGJADHKADGKAEGLBDGKADHJAEHJADGJAEGJBDGLAEGJ
ADHJBDGLBDHLBDGKADHJADHKBDGLBEGLAEHJAEHJADHJADHKBDGJBEHLADG
JBDHLBDGKAEGJAEGLADHJADGLADGJBDHJBDHJAEGJAEGLBDHLAEGJAEGLBD
GKAEHKAEGJADHKAEGKADHJBEHLADHJBDGLADGLADGJADHKBEGJBDGJBDHKB
DGLADHJBDHJBEHLBDGKADHJADGLAEGLBDGKADHJADGLADHJBDHLADGJBDGL
ADHKBDHKAEGLBDGKAEGJADHKADGKBDGLBDHKADHLADHJADGLBEHLADGLADH
JBEGKADGJADGLAEGKADGJBDHJBDGJADHJAEGJADHKAEGLBDGKADGJAEGLAD
HKBDHKADGLAEHJAEGJAEHJADGJBDGJAEGJBEGJADHJAEGLBDGKAEGJADHKA
EGKAEGJAEGLBDGLBDHKADHLADHJADGLBEHLBEGKBEGLBEGJBDGKBDHKBEGL
AEGLBDHKBEHJAEGLBDGKADHJBDHLADGJBEHLAEGLBDHKBDGKADHJADGJADG
LAEGLBDGKADHJADGLADGJBDHJBDHJAEGJAEGLBDGLADGJBEHLAEGLBDHKAE
GJAEGLBDGLADHJBDGJBEHJBDHKBDGKAEHJADHJADGJADGLBDHKBDGKAEHJA
DHJADGJADGLAEGJBDGLBDGKADGJBDGJBDGJBDHJADHJAEGLBDHKBDHKBDGJ
ADGJAEGLADHJBDHLBDGKADHJADHKBDGLBDGKADHJAEGLBDGKBDHKBEGLADG
KBDGKAEGLAEGJAEGLBDHKADHLADHJADGLADGJBEHJAEGLADHJADGLBDHLAD
GJADGLAEHJBDGLAEGJAEGLBDHKBEGJBEGJBEGLADGLADGLADHJAEHJAEGLB
DHKBDGKADHJADGLAEGLBDGKADGJAEGLBDGLBDGKADHJBDHKBEGLADGKBDGK
AEGLAEGLBDHKBDGKADGJADHLADHJBDHLBDHKADHKAEHJADHJADGLADHJAEH
JADGJAEGLAEGLBDGKAEGJBDGLBDHJBEGLAEGLADGJAEGLAEGLBDGKADHJAE
GLAEGJBEGKADHJAEGJAEGLADGJBDGJBDGJBDGLADHJADHJBEGKADHJAEHJB
EHKBEGLAEGJAEGLADHJADHKADGJAEGLBEGLADGLADGJBDGJBDHJBEGLAEGL
BDHLBDGKADHJADHKAEGLBDGKADHJADGLADGJBDHJBDHJAEGJAEGLADGJBEG
JAEGLBEGLADGJBDGJBDGJBEHLAEGLBDHKBDHKAEGKADGJBDHLADGJAEGLBE
GJBDGKBDHKBEGLAEGLBDHKBEHJAEGJAEGLBDGLBDHLADGJAEGJBDGLAEGLB

```
EGJBDHKADGJAEGLAEHKBDHKBEGJAEGKADHJAEGLADGJADHKAEHJBDGJBDHK
BDHKAEGKADHJAEHJADGJAEGLAEGJAEGLADGJADHKAEHJAEGLBDGKADHJADH
KBDGKBEGLADGLADGLAEGJADHJAEHJBDHKADHKADGJBDGJAEGJBEGJADHJBD
GLAEGLADGJADGLAEGLADHJAEHJAEGLBDHKBDGKADHJADGLBEHJADHJADHJA
EGLBEHJBDHKADGLAEGJAEGLBEHJBDGJADGJBDGLBDGKADHJAEHJADGJBEGJ
ADGLBDHKBDGLBDGLBDGKADHJADGLBEGKAEGJADHKAEHJAEGLBDGKADGJAEG
LBDGLBDGKADHJBDGKADGJAEHJADHKADHJADHLADHJADGLBDHJADHJBEHJBD
HKADGLADHJBDGLADHJADHJADHKADGJADGLADGJBDHJBDHJAEGJAEGLBDHLA
EGJAEGLBDGKADHJAEGJAEGLBDGKADHJADGLADGJBDHLADGJAEGJBDGLAEGL
BEGJBDHKADGJAEGLAEHKBDHKBEGJAEGKADHJAEGLBDHKADGLADGJBDHLADG
JAEGLBEGJBDGKAEGLBDHKAEGLADGJAEGKADHJBDHKBEGLAEGLBDHKBEHJAE
GJAEGLADGJADHKAEHJBDHJBEGLADGLADHKAEGJADHKADGKBDHLAEGJAEGLB
DGKBEGJBEGLADGLAEGJBDHKBDGLAEGJAEGLBEHLBDGLBDGKADHJADGLADGJ
ADHKADGJBEGJADGLBDHKBDGLBDGLAEGLBDGKADHJBEHJAEGJADHJBDGJAEH
JADGJBEHJAEGLADHJADGLAEGJAEGLADGJADHKAEHJBDHLADGJBDGLAEGJBE
GLBDGLAEGLAEGJADHKAEGLAEGJBEGKADHJAEGLBDHKBDGLADHJADHJAEGJA
EGLAEHKBDHKAEHKAEHJBDHKBDHLADHKADGJBDGJADGJADGLADGKADHJADGL
ADGJBDHJBDHJAEGJAEGLBDGKBDHKBDGJADHJBEGLADHKAEHJADHJADGLAEG
LBDGKADHJBDGKADHJAEHJADGKADHJAEGJADHKADGJADHKBDHKAEGLBDGKAD
HJADGLBEGKBDHKBEGKADHJADHKAEGLAEHJBDHKBDHLADHKBDHLADHJADHKA
EGLADGJBDGJAEGJBEGJADHJADGJBEHJAEGLADHJADGLAEGJAEGLADHKADHJ
ADHLADHJADGLBDHKADHKBEGJADHJBEGJBDHKADHKBDGLAEGJAEHJADHJADG
LAEGJADHKADGKBDGKBDHKBDHLAEGJADHKAEGLBDGKADHJBDHLBDHKADGLBD
GJAEHJBDGLBDGKADHJBDHLADGJBDGLAEGLBDHKADGKADHJAEGLBDHKBEGLA
EGLADGJADGKADGJAEGJADHK
```

2. Break the following ciphertext. Although it is a variable-length code and not prefix-free, you should be able to find a weakness and decrypt it.

```
624435638696115072679850656809026349262963492606798506332249
6963492658679146244356441633224963421679146809026263492658
6210864416791465868090263492679146386961150726798506568090
2633224963421679146441644163322496067914680902626332249680
9026349265679146115072633224968090267914644163322496809026
8090262679146296791461067914629634926763421679146332249638
6244356386809026263322496809026244356441685721963492611507
2658644163492611507265626115072633224964416791464416332249
6115072679146963492638610679146115072680902679146586244356
3868090263492644163492634216791468090262624435638606791462
9644167914633224969634926586791463422624435606268090268090
2611507263322496386441676349261150726342169626332249638606
7914624435638090263492696106065864197634926115072696349269
6419768090263322496244356296296349262108638606791468090262
6791462108644163863322496809026244356349263863322496296441
6791469621086115072624435680902679850633224960679146386967
9850658679146762443563867914658633224969634926586791463322
4964416210864164416809026244356809026210868090262443563492
6386961150726798506568090263492644167985064416809026791463
42162443563868572196262443569626809026267914656296332249624
43
```

5638680902679146558045680902679146296791463421679146386809026441633224961150726724435629679850685721963492611507265864416562611507263322496441679146441634926115072644167914638680902679146386968914646416332249629629679850696349262586791467914636210862446356106332249629679146386809026441696332249629629679146586963492658679146791463621086244356106332249629679146386809026441696332249629629679146586963492658679146791463621086244356106332249629679146386809026441696332249629629679146586963492658679146791463621086244

Unit 141
Baconian code

The *Baconian cipher*, also known as *Bacon's cipher* or *biliteral cipher* or *biliterarie cipher*, was invented by Francis Bacon a long time ago. It is a fixed-width code in which the ciphertext symbols are A and B. The code words are in the following table:

letter	codeword	letter	codeword
A	AAAAA	N	ABBAB
B	AAAAB	O	ABBBA
C	AAABA	P	ABBBB
D	AAABB	Q	BAAAA
E	AABAA	R	BAAAB
F	AABAB	S	BAABA
G	AABBA	T	BAABB
H	AABBB	U	BABAA
I	ABAAA	V	BABAB
J	ABAAB	W	BABBA
K	ABABA	X	BABBB
L	ABABB	Y	BBAAA
M	ABBAA	Z	BBAAB

Table 141.1: Letters and their codewords in the Baconian code.

If we replace the ciphertext symbols A and B with '0' and '1,' we obtain a five-bit binary encoding.

Reading and references

Francis Bacon, *Of the proficience and advancement of Learning, divine and humane*, London: Henrie Tomes, 1605.

Wikipedia, en.wikipedia.org/wiki/Bacon's_cipher

American Cryptogram Association, www.cryptogram.org/downloads/aca.info/ciphers/Baconian.pdf

Fletcher Pratt, *Secret and Urgent: The Story of Codes and Ciphers*, New York: Bobbs-Merrill, 1939, chapter V, section I.

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, pages 882-884.

Blaise de Vigenère, *Traicté des chiffres ou secrètes manières d'escrire*, Paris: Abel l'Angelier, 1586, HDL: 2027/ien.35552000251008, gallica.bnf.fr/ark:/12148/bpt6k1040608n, gallica.bnf.fr/ark:/12148/bpt6k94009991, pages 200b and 241-243b.

Programming tasks

1. Write a function that takes a letter of the alphabet and returns a five-bit binary representation. You may store the five-bit number as a string of 0s and 1s or as an array.

2. Write a function that takes a five-bit binary number and returns the corresponding letter.

3. Implement the Baconian cipher. Allow a switch so that it can use the symbols `A` and `B` or '0' and '1.'

3. Write a function or script to take a ciphertext that contains two symbols (*any* two symbols) and decodes it as a Baconian cipher. Be careful that if you interpret the ciphertext symbols incorrectly, then you will find code words that are not in the table above. In that case, you should automatically correct the mistake.

Exercises

1. Break this ciphertext:

```
B88B88BBB88BB8888B888888B88B888B8BB8B88888B88B8B8B88B8
8B88BB88BBB88888B88BB88B8BB888B888888BB8B888B88B888B88
B88888B88888888B88BBB88BB8B8B888B88B8B88BB88BBB88B88B8
8BBB888BB8B8888B8888888B8B88B88BB88BBB8BBB8B888B8BBB88
8B8BB88B888BBB888888B8B888B88B88B88BBBB88B8888888B888B
88B88B88B8B8BB88BBB8B888B8B8B8B88B888B8BB8B88B888BB88B
B88BBB88B88B888B8BB888BBB8B888B88B88B88B88BBB88BB8888B
88888B88B8BB888888B8888BB88B88BB88BBB88888B88BB88BBB88
B8888B888BB8B888B88BBB8888BB88B88888BBB88B888B88888B8B
888B88B88B88BB8BB8888B88B88B8B88B88888888BB888B88B88B8
8B8888BB8BB88BB8BBB8B88BB88BBB88B8888B8B8B888B888BB88B
8B88BB88B8B8BBB88B8BB8B8888BBB88BBB888B8BB88B888BBB888
888B8B888B88B88B88BBBB88B8888888B888B88B88B88B8B8BB8B8
88B8B888B88BB8B8888BB8B88BB8B88B8
```

Unit 142
Triliteral code

The *triliteral cipher* (or *triliterarie cipher*) is a fixed-width code that uses the ciphertext symbols A, B, and C, and these code words:

letter	codeword	letter	codeword	letter	codeword
A	AAA	J	BAA	S	CAA
B	AAB	K	BAB	T	CAB
C	AAC	L	BAC	U	CAC
D	ABA	M	BBA	V	CBA
E	ABB	N	BBB	W	CBB
F	ABC	O	BBC	X	CBC
G	ACA	P	BCA	Y	CCA
H	ACB	Q	BCB	Z	CCB
I	ACC	R	BCC	*space*	CCC

Table 142.1: Letters and their codewords in the triliteral code. The use of the space character is optional.

Some add `CCC` to represent a space, or put the space at the beginning and shift all the letters down one.

If we replace `A`, `B`, and `C` with ‘0,’ ‘1,’ and ‘2,’ then we have a three-digit base-3 (ternary) encoding.

Programming tasks

1. Write a function to take a letter and return a three-digit ternary number. Allow for the possibility that `000` might represent a space. You may want to represent the ternary number as a string or an array.

2. Write a function to take a three-digit ternary number and return a character. Allow for the possibilities that the alphabet starts with `A` or with a space.

3. Put the pieces together to implement the code.

4. Write a function or script that takes a ciphertext that only has three symbols and finds the best decoding as a triliteral cipher. There are six ways to assign the three symbols, and two ways to decide if `000` is a space or not. Use tetragram fitness to choose the best decoding.

Exercises

1. Break this ciphertext:

```
V4V4VVV444AV4AV444A44V4A44444A444AAA44A4VAA444A4444AAVVA4AA
A44V4AVV44444VVA4AVV4A444A44V4V444A44V4A444A44V4VV4A44444A4
44AAA44A4VAA444A4444AAVVA4AAA44V4AVV44444V4AAV4AAA4AVVAV4VV
4AVV444AV444A44AAV444V4V4VV44AV4V44444AAAV4A4A444AVAAVV4V44
4AAAVVAV4AV4VA444A4444V4VV44A44AAVA44V4VVVA444VVAAV4VA44444
VA4AVV4V444V4V4VVV444AV4AV44444AV4A44444A444AAA44A4VAA444A4
444AAVVA4AAA44V4AVV44444V4V4VV4AV444AVA44AV4AV4V44444AAAV4A
A444V4V4VV4AV444AVAV444AVV4A4AVAAVV4V44444AAAV4AA444V4V4VV4
AV4444V4VA4V4VVA4V444AV4444V444AA44V4V4VV44444AAAV4AA4444VV
AV4AVA4AV44444AAAV4AA4444A44VV44AV44A44V4VVVA444V4V4VV4AV44
44VV4AV44AV44V4V44444AAAV4AA444V4V4VV4AV44444VV4444AA44AAV4
4444AAAV4AA444V4V4VV4AV44444VAV44AAVVA4444VAA44VAA4AV444VVA
AV4VA4444V4V4VVV444AV4AV44444AV4A44444A444AAA44A4VAA444A444
4AAVVA4AAA44V4AVV44
```

Challenge

```
IECIEAEAHEAHDGBGCDCIDGADIAEAEHEAHGCDIADBHFEGBGDCAIDFIBEBGDA
GGDCFBIEAIICEHFCCDGHDAFBGCIEAIDAGDBEHCHFGDCGCEFCHCDGEGCBGFB
IFGDCICEFIAEGCGBDBFGFCHBHEHFAADHHADIADCDIADGEHBEIAFHCAHFIBF
BGFIEAEAIDICBFHAHFFIBFHABGEEAIDHAGBEAFHEHCAIEBFHEAIHFAEBIGC
DAFIFHACIDGBDCGDAHFDBGHBEIBDGDCHADFAHDCIAEHDGCIBFBEGFHAIDCF
CHDGCGECDAHAEICDGFBIDGAEBHAIEAGDIAEHAFHFBHEAGDCCGDFBGAFHIAF
IFACDGDAGGADIDAIBDFHCDGCEGCHADEIAGDAEHBIEAGDBDGCFHABGDBEHHF
CGCDECGHFBCEGDCIGADGBFBIFDGCICEHFAHEAHAEADGEHBEIAEAGHAEHFAH
CFAEIHADICDDAIBGDEIAAHDDGAHFAADIBIFHEACHFHDBCDGCEIAEIAHEHEA
CHFDGCGCEFBGFBIDCGECIDAICHEAEIIBFIAECHEAIEDAHHBFIEADAGAGDHE
BEAIEGBGCEHFCCDIICDGDCFCHCDGCGEAHEHEAEHBAFHCHEIAEDAGGCDAGDI
EAAHEHEABFHIAEDAGEHBAIEAIDDHAFBIAHFFHBAEIDCIHFAIFBBDHAGDEBH
EIABIFADIHEAAHEFBGBIFDGCECICEIBEHDCGDCIEGAEHAFAHHFCDIAIFBEG
BGDCFBIGDAEHBIAEDAGAEIHAFBHFCDGEHBIADHAEEAHAGDAEIHEAHEAFCHD
GCEGCADGHEBIAEIDAHADIFBAFHHFBIEADCIIFAEGCGADHCFCGDCGEBGFIFB
DCGCEIIDADAGIDCAEIEAIFBHCDIAGDDGCHBFAEIAGDBHEIEACHFEGBAIDCH
EEAIDGABEHEAICDIEAIIFAFAHEHAAHEEAGEHAAHFFCHAEIHDACIDFBIGDCC
IEFAHDHBAHFFCHDCIHECEAIHADHCFGEAEIAGBDCGEEHADAIAFHDAHIFBHFA
BFHIEACIDIECBEHGCDIDCDGCFIBDIBIADDAHIDCADGAEIGFCHFABDGAGDHA
EFHCEICHBEHFAAGDFBIGDCEICEHBAHFGADCDICGDIFBDICEIABGDCDGIBFB
DHADIBGEAEIAGDAIFIEAEHBIADIBFHDBAGDHBEAEIEAGHEAHAFAGDIEAAGD
CGDBDHCGDDICGDCBHFAIEBDIAFHIFBBGDCHFBGDFHAGADDBGHEBDIABFIBE
GAGDCGDBHFCGDDHAADHGCDIECICDEAGAIDGDAGBDHEBIDAIFBEGBCGDFBIF
```

HBFCHADGEIAHAFHFBHFABFIBHDAFHEHBAIEAFHCHECHFHBEIDAADGGDAEAI
DAHGEBAEIDGAIDCCEGEAGIBFGDCEICADGHEBEIAHBEAEIAFHCHEHFCEBHAD
IAGDADGEAIHDAFIAECGFIBGDACIDAGDHEBAIEFIAHAFHEAAEHIECBHEIAEI
BFBEHAIEIAFGCEIBFAGDICDGDAEBHAIEIFAHFAHAEAHEHBFAEIIAFAEIDAI
FBIEBGAHFGEBCGDCDGHBDBDGHFAGDAGDBHBEAEIHADADIBHFBEGGCDFBIFB
IAHFDAGBHEHDADCGCIEADGHBEEAIEGBCGEHFCGCDEGCDGAFAHDAGBDIADIH
DAIDCDGAIFAFAHIDCIEACDIDCGDIAGAEDIADBGFGBGCEGAEGDABHEIEAAFI
AHFAEHHAEFAHBIFHDBGADHEBDAHGCDIECDIAGDAAGDDCGCIEHBEDGCIFBDC
GICEICEHBECGDCDIGEBDGCADGDIAAGDFBIFAHDGACGEHDAFHAAEHAEHFHCA
EHDGCDGCFBGDIADBIIADAGDEBHAHDDCGICEGDABHEEIAIAFFHAHAEHEAGAD
DCGBDIADIHADDICDAGIAFAFHCDIIEACDIGCDHBFEIAAIFDGCBDHCFHDICEB
GGDCADGADGFHAEGBAIEAGDDAIADGBFIGDCICEEICBHECDGBEHHFADCIAIDG
DAFIBHFAGADEGCAHDFAHAHEHEAHCFIDCDCGADIAEGIADGDBFBGGECEAGADG
HEBIAEAIFAHFHEAAHEFHABIFHBDDAIADGHEBADHDCGIECIDAADGDAGDCGCE
IEBHGCDIECBEHDGCHBDAHDGCDAEGICDGDAEHBAIEIAFHAFEAHHAEHAFBFID
BHDAGBHEEIAGBEGECCFHDAHGCEIBFICDDGACDGCIDEIABDGCGDIBFBDHECI
EBHCGDEAGDAIDBGBFGDICCGEEAGGADHEBIEAFAIAFHHEAEAHFAHBFIHBDDA
GEHBAHDCDGCIECDIDAIADGDAGGDCCIEHEBHAFDAGCEIBHEHAFGDAGDABEHA
HDDCGIECCIDDIAGADGADGDCDAIBHDDGCIFBADGGFBBIFGDCIECDAIBDHDGC
BIFAGDBGFIFBGCDEICADGEBHDHADCGCIEDCIAIDDGAAIFHFAGBDBFGDGAGD
CADGCDGFBHDCGHDADAHDGCIECDAGAHDDAIGAEHAEIAEGEAAEHAFHCFH

Unit 143
Morse code

Morse code is a variable-length code. The ciphertext symbols are the dot · the dash – and space. Sometimes the space is written with a slash ('/'); it is placed between code words. In addition to the English alphabet, there are code words for other European characters, digits, and some punctuation. In the table below, you might notice that some code words are not unique.

A	·–	0	–––––	Ä	·–·–	.	·–·–·–
B	–···	1	·––––	Á	·––·–	,	––··––
C	–·–·	2	··–––	Å	·––·–	:	–––···
D	–··	3	···––	CH	––––	;	–·–·–·
E	·	4	····–	É	··–··	-	–····–
F	··–·	5	·····	Ñ	––·––	!	–·–·––
G	––·	6	–····	Ö	–––·	?	··––··
H	····	7	––···	Ü	··––	'	·––––·
I	··	8	–––··			"	·–··–·
J	·–––	9	––––·			(	–·––·
K	–·–					)	–·––·–
L	·–··					&	·–···
M	––					%	·––––·
N	–·					$	···–··–
O	–––					@	·––·–·
P	·––·					/	–··–·
Q	––·–					+	·–·–·
R	·–·					=	–···–
S	···					#	···–·–
T	–					~	·–···
U	··–					_	··––·–
V	···–						
W	·––						
X	–··–						
Y	–·––						
Z	––··						

Table 143.1: International Morse code.

Reading and references

Wikipedia, en.wikipedia.org/wiki/Morse_code

Morse Code World, morsecode.world/international/morse2.html

Programming tasks

1. Implement Morse code.

2. Write a function or script that takes a ciphertext containing three symbols and finds the best interpretation of those symbols as dot, dash, and space, and decodes appropriately.

Exercises

1. Encode this text in Morse code:

```
IN MORSE CODE THE LENGTH OF A DASH IS THREE TIMES THE LENGTH OF
A DOT. THE SPACE BETWEEN DOTS AND DASHES IN THE SAME CODEWORD IS
THE LENGTH OF ONE DOT. THE SPACE BETWEEN LETTERS IS THE LENGTH
OF THREE DOTS. THE SPACE BETWEEN WORDS IS THE LENGTH OF SEVEN
DOTS.
```

2. Decode this text:

```
_   ....   ..   ...     _   .   _..._   _     .__   ._   ...     .
_.   _._.   ___   _..   .   _..     ..   _.     __   ___   ._.
...   .     _._.   ___   _..   .   ._._._     _   ....   ..   ...
_   .   _..._   _     ..   ...     ._   ._..   ...   ___     _.
___   _     ..._   .   ._.   _.__     ..   _.   _   .   ._.   .
...   _   ..   _.   __.   ._._._
```

3. Decode this text:

```
::'';':;:'::':'::;'';'::;''::;'';':;':;'::';:;';:;'';:;:
:';'::;'';'::'::;'';'::;'';'::;'';'::;'';'::;'';'::;'';:
::;':;::';::'';::';;';'';::;':'::;;':;;';;::';:';':'
:;:':'::;';::';:';::';:::;;;'';:;';:::;;:::';:;;';:::;
';:;''::;:';:';:;';:::';:;'';:::;';:;;';:;:'';:';:;';:;'';:::;'::'
::';':';:'::';:';:;';:::;;;';:';:;;';:;:'';:;:';:;;';:;'';:::;'
:;';:::;':'';:;;';:::;'';:;';:::;';:::;';:;:';:;:';:::;';:;:';:;;':
:;';:;;';:::;'';:;';:';:;;';:::;':';:;:';:;:';:;:';:;:';:;:';:;:
';:;;';:;:';:;';:';:;';:';:;:;:;
```

Challenge

```
011100010101010001010001010100000001110101110100010100010 11
101110100010101010001000101110100011100010001110101011100 01
110000000101010001010001110111000101011100010111010100010 11
100011100010001010100000001110001010101000100000000101010 111
000111011101110001011101010001110001011100011101110100010 00
000011101110111000111010000000101110000000111000100010111 01
010001000111011101000101110100010111000101110111010001010 10
100000001011101010001010001110100010000000010111000101010 000
000101110000000101011101000101011100011101000111010111010 00
111000101000111011101110001110100000001110111011100010101 11
010000000111000101000111011110001000101110101110101110000 000
101000111000000010101110100011101110111000101110101000101 11
010100011101110111000101110111000101010000000111000101010 10
001000000001110101000100010101000111010111010001011101000 101
000101110111010001110001010001110111011110001110100000001 110
111011100010101110100000001110001010101000100000001110111 00
011101110111000101110100010101000100000001010100011100010 11
100011101000111010100010111000101110100011101010000000111 01
110100010100010101011100010001110100000001010001110100000 000
100011101010111000100010111010001110101110100010100010101 00
010000000101110111011101110001011101011110101110000000111 010
001110111011100010111011110000000111000101110100011101011 101
110000000111000111011101110000000010100011101110001011100 011
101110100010100011101000100000001010101000111011101110001 01
110111000000010101010001011100010111010001110101000000011 10
001010101000101000101010000000101110111000111011101110001 01
011100010111010100011101010000000111010101000100000001010 00
101011101000000011101011101110001110111011110001010111000 000
010111011100010001011101000100000001011101010001010001010 10
001110001000111010001010001110100011101110100000001110001 11
011101110000000101110000000111011110001110111011100011101 010
001000111011110001011101011110101110
```

Unit 144
Fractionated Morse cipher

The *fractionated Morse cipher* first encodes a text with Morse code using dots, dashes, and × between letters, and ×× between words. The stream of symbols is taken three at a time and replaced with letters according to an alphabet key that may be mixed by use of a keyword. Since ××× never occurs, there are $3^3-1 = 26$ combinations, which fits the number of letters that we can use. If necessary, we pad with one or two × symbols. The fractionation occurs when symbols from the same plaintext letter are separated and become part of different ciphertext letters.

As an example, let's encipher a short message with the keyword `FRACTION`. If our message is

`HIDE THE CANDIES UNDER THE PILLOW`

we first encode it in Morse code, with × as the separator:

```
H    I  D   E  T H    E  C    A  N  D   I  E S
····×··×—··×·××—×····×·××—·—·×·—×—·×—··×··×·×···××

U   N  D   E R    T H    E  P    I  L    L    O   W
··—×—·×—··×·×·—·××—×····×·××·——·×··×·—··×·—··×———×·——
```

The resulting symbol stream is divided into groups of three. We need to pad the last group with two × symbols.

```
···  ·×·  ·×—  ··×  ·××  —×·  ···  ×·×  ×—·  —·×  ·—×  —·×  —··  ×··
×·×   ···  ××·  ·—×  —·×  —··  ×·×  ·—·  ××—  ×··  ··×  ·××  ·——  ·×·
·×·  —··  ×·—  ··×  ———  ×·—  —××
```

Now let's set up our key. Each letter in the mixed alphabet is associated with one triplet of symbols.

F	R	A	C	T	I	O	N	B	D	E	G	H	J	K	L	M	P	Q	S	U	V	W	X	Y	Z
·	·	·	·	·	·	·	·	·	—	—	—	—	—	—	—	—	—	×	×	×	×	×	×	×	×
·	·	·	—	—	—	×	×	×	·	·	·	—	—	—	×	×	×	·	·	·	—	—	—	×	×
·	—	×	·	—	×	·	—	×	·	—	×	·	—	×	·	—	×	·	—	×	·	—	×	·	—

Note that we do not alter the Morse symbols in this table; we only modify the scrambled alphabet. By replacing each group of three symbols with the letter that appears above it (where the groups are written vertically), we get the ciphertext:

`FONABLFUVGIGDQUFYIGDUCZQABTOODSAJSP`

We can attack the fractionated Morse cipher with the same hill-climbing attack in Unit 36. However, we will not be seeking the best plaintext, but rather the best match to English text that was enciphered with the key `ABCDEFGHIJKLMNOPQRSTUVWXYZ`. So, we must encipher a corpus of text with this key and from it compile a table of tetragram frequencies for the attack to use. We must also augment the attack with a margin of error and allow downward steps about 5% of the time, as we did for the attacks on grid-based ciphers, in order to avoid becoming trapped in a local maximum. A good margin to use is 0.5. Once we have found a key that gives the best match to English text that was enciphered with the key `ABCDEFGHIJKLMNOPQRSTUVWXYZ`, we can use that key to simply decipher the ciphertext. To say this another way: we are factoring the cipher into a fractionated Morse cipher that has the standard alphabet as its key, followed by a monoalphabetic substitution; first we break the substitution, then we decipher the fractionated Morse.

Let's take our example to see what is really going on by breaking the cipher up into its smallest components. The first step is to encode with Morse, where spaces are handled as explained above; we can label this operation *M*.

```
HIDE THE CANDIES UNDER THE PILLOW
                ↓
····×··×—··×··××—×····×··××—·—··×·—×—·×—···×··×·×···
××··—×—·×—···×·×·—·××—×····×··××·——·×··×·—···×·—···×———
×·——
```

Next is the fractionation into blocks of three characters; call this *F*. Don't forget to pad the end.

```
                ↓
···  ·×·  ·×—  ··×  ·××  —×·  ···  ×·×  ×—·  —·×  ·—×  —·×  —··
×··  ×·×  ···  ××·  ·—×  —·×  —··  ×·×  ·—·  ××—  ×··  ··×  ·××
·——  ·×·  ·×·  —··  ×·—  ··×  ———  ×·—  —××
```

Then is a transliteration step (*X*) in which we change from {·, —, ×} to {A, B, C}. You might think of this step as an inverse symbolic substitution.

```
                ↓
AAA ACA ACB AAC ACC BCA AAA CAC CBA BAC ABC BAC BAA CAA CAC
AAA CCA ABC BAC BAA CAC ABA CCB CAA AAC ACC ABB ACA ACA BAA
CAB AAC BBB CAB BCC
```

Now comes a triliteral decoding (T^{-1}) from blocks of three letters from the set {A, B, C} to letters of the usual English alphabet.

```
                ↓
AGHCIPAUVLFLJSUAYFLJUDZSCIEGGJTCNTR
```

The final step is a monoalphabetic substitution $S(k)$, where k is the key both of this step and of the full cipher.

↓

FONABLFUVGIGDQUFYIGDUCZQABTOODSAJSP

So if we write the full cipher as $Z(k)$, for lack of a better choice of letter and since we have already used F, we have this equation (in operator notation):

$$\begin{aligned} c &= Z(k)\, p \\ &= S(k)\, T^{-1}\, X\, F\, M\, p \\ &= S(k)\, S(e)\, T^{-1}\, X\, F\, M\, p \\ &= S(k)\, [S(e)\, T^{-1}\, X\, F\, M]\, p \\ &= S(k)\, Z(e)\, p \end{aligned}$$

Remember that e is the name we gave to the unmixed alphabet. We have inserted another substitution $S(e)$ which does not actually do anything, but it completes the sequence of operators inside the brackets to be a full fractionated Morse cipher, albeit one without a key. What we have shown is that the fractionated Morse cipher can be factored into an unkeyed fractionated Morse followed by a monoalphabetic substitution. Since the substitution occurs last during encryption, it comes first in decryption, and we attack it first.

The attack is simple to understand. In section 36 we developed a hill-climbing attack on the monoalphabetic substitution based on n-gram fitness. The difference now is that we are not dealing with typical English text t, but with $Z(e)\, t$, i.e., with typical text that has undergone a keyless fractionated Morse. This means that we can carry over our attack from section 36 if we make only two changes: our tetragram-frequency table must be replaced with a new one generated from $Z(e)\, t$, where t now represents our textual corpus, which includes letters and spaces. The table, however, only contains frequencies of blocks of four letters, since spaces do not survive the $Z(e)$ operation. The second change is to allow for the possibility of occasional steps to lower fitness. We've done this before— one way to do it is to allow a 5% reduction every few out of a hundred steps; this removes the need for random restarting.

Reading and references

Practical Cryptography, practicalcryptography.com/ciphers/fractionated-morse-cipher

American Cryptogram Association, www.cryptogram.org/downloads/aca.info/ciphers/FractionatedMorse.pdf

Programming tasks

1. Implement the fractionated Morse cipher.

2. Implement a dictionary attack on the fractionated Morse cipher.

3. Encipher your textual corpus with a keyless fractionated Morse, i.e., one with key equal to the unmixed alphabet. From the resulting text, compile a table of tetragram frequencies and save them

into a file that your attack can use. Remember that you need to use letters *and* spaces from your corpus. The tetragrams whose frequencies you tabulate will only contain letters, however.

4. Put together the attack from section 36 and your frequency table from Task 3 to implement the hill-climbing attack on the fractionated Morse cipher.

Exercises

1. Encipher this text with the keyword `LINGER`. Use the same method for constructing the mixed alphabet as in the example above.

   Two hours. One hundred and twenty minutes. Anything might be done in that time. Anything. Nothing. Oh, he had had hundreds of hours, and what had he done with them? Wasted them, spilt the precious minutes as though his reservoir were inexhaustible.

   (from *Crome Yellow* by Aldous Huxley)

2. Decipher this ciphertext with the keyword `ODIN`. Use the same method for constructing the mixed alphabet as in the example above.

```
LVVJESICJCMBOTVWQSICJOSQCQEIBJLXOCTEQMLVTVCCNSJCJEESCBJBJGT
VCCNSJCJEEWWQMJDCJLWXOCWQCSCMCICVWPMOWPGTPPIPOUJBMEMDESIBPI
DVTVVEPQCCMBOXOCVTWULSJPOTXMLWQEOSQTTIATVMOVICOSVVCILTIBJGW
TVSTQMNCDUOTPVXBAWPGSJGCSIIJGUBHCOXOCTQMNCDUOVJBDWCOXLVXOCW
TXNCPIPQEIPOUHQICDWPBNUGSEMNCQEIMBJIBJGXOCVXOCTQMNTIGTPPIIJ
GTTEBJCIOTVVENWPGVF
```

3. Break this ciphertext with a dictionary attack.

```
FHRNJLMKCRAKPSYMEWYMRNJYUKRMAAFKPSGRMSFLEJYSPWHUWPMVMFMUVSI
CVKJPRUUMEEQAMAJAYMEEFTEJHRAFHUSDMNEYVNKGRRMRKEQATMNACUJKPS
CMRQIMFFHWMLSFPMMADEJLQMEMHSYRETVQKADRMNQAITMGUJTKNYUKAGKIA
YRMRAIFCQERMMGSCUFPMMGVQNEIPWHRMQKGQFHVNALEJYWPMRAHUVEYVKCU
ENJFKPQMJPMUREJLVMPUAYVNQAMFFHREQATQUUMNEATKNYUKYREJPWYQKCS
IFCRJMFYUJJLQEIYRWCMIAYEAPRQFAFMPVKIPSTQAMFMEEVNIMRFTRREKNE
ADEJJLEJYWICMCSCMIKKCRAKPQQMMAIPUVNQAAFGWHWPMRNJYUNJLMGRUEA
DNQAEPSIPKRB
```

4. Break this ciphertext with a hill-climbing attack. What is the keyword?

```
NTQQMKATXTMQHLGQBSMQMIWOMLLDMDURBKTQXLJVQQLADWOGVHESHLBLLHN
TUNSLBHGSSHTMLMHTNBGUAHDQSNNLTWNOBEQWNATXJVLMOQLAHDQVVHATUL
TQLJQNDVBTUHMVMVBQLAOKHBMBHTBQAMHFNDSXMMLANBBQLBKEMGQVHBLES
MBBHATXJSBOBKNKLTQLLHIXJVHEXJQHEBQLBGWOBMVHNBKHLTNVXESMVBWO
KHMNHNTUFUFNOBLEBKGWVKDBKGBLHMLTSNNLFVWOGVNVBKEMGWVKTXJWNDS
QHGMNLTBKEMDQWMTXHLHIQAMHFOMSSMDVKBBLBLKFOGSSMUGSQHNSBQLANM
KATVWOUDQVXTMWNGQHLFMQHJSBGSQOQWOBLESMBBHGMBHTBVQVWMLOKGWQC
```

Unit 145
Morbit cipher

If instead of three Morse symbols per ciphertext symbol, as in the fractionated Morse cipher, we use two, then the cipher is called the *morbit cipher*. The name may be a shortened version of “Morse bit.” This cipher is keyed with a nine-letter keyword, which is then converted to digits in the same way as we found a permutation from a keyword when we studied transposition ciphers. So, for example, if the keyword is `FRACTIONS`, the substitution uses this table (note that most people use the digits 1-9 rather than 0-8):

F	R	A	C	T	I	O	N	S
3	7	1	2	9	4	6	5	8
·	·	·	–	–	–	×	×	×
·	–	×	·	–	×	·	–	×

The ordering of the dots, dashes, and crosses in the lower two rows is standard and fixed; we should not change it. Now, a plaintext such as

```
THIS IS MORBIT
```

is enciphered with that key in this way:

```
T  H     I   S     I   S     M   O    R    B     I   T
–×····×··×···××··×···××––×–––×·–·×–···×··×-×
4  3  3  6  1  3  1  6  1  3  1  5  4  9  4  7  1  2  3  6  1  4
```

The ciphertext is

```
4336131613154947123614
```

We can factor the morbit cipher in the same way that we factored the fractionated Morse cipher. The effect of the full cipher (m) is then seen to be the action of a keyless morbit, followed by a substitution that acts on the set of nine digits. The factors are Morse (M), fractionating into groups of two (F), transcription to base-3 digits (X), converting from base 3 to decimal while adding one (Y), and finally a substitution (S) that acts on the set of nine non-zero digits. The morbit (Z) is then a keyless morbit (one whose key is the identity permutation e on the nine digits) followed by the substitution:

$$c \;=\; m(k)\,p \;=\; S(k)\,Y\,X\,F\,M\,p \;=\; S(k)\,[S(e)\,Y\,X\,F\,M]\,p \;=\; S(k)\,m(e)\,p$$

It follows that to break a ciphertext encrypted with morbit, we could use a hill-climbing attack similar to the one we built for fractionated Morse. In this case, however, we would have to use *n*-grams comprised of digits, and those *n*-grams would need to have length six or more. An alternative approach to the attack is to break the cipher between *F* and *M*:

$$c \;=\; m(k)\; p \;=\; [S(k)\; Y\; X\; F]\; M\; p$$

We then attack the part of the cipher in the brackets by hill-climbing a fitness based on dekagrams (10-grams) of symbols from a Morse encoding of our corpus with spaces (and punctuation, if wanted). Once the best key is found, we can fully decrypt.

Reading and references

American Cryptogram Association, www.cryptogram.org/downloads/aca.info/ciphers/Morbit.pdf

Programming tasks

1. Implement the morbit cipher.

2. Implement a dictionary attack on the morbit cipher. Remember to use only nine-letter words.

3. Encode your textual corpus that contains spaces (and maybe also punctuation) with Morse. From the result, compile a table of dekagram frequencies and save them into a file that your attack can use.

4. Build a hill-climbing attack that uses your table of Morse dekagram frequencies.

Exercises

1. Encipher this text with the morbit cipher with the keyword `BACKWARDS`.

   > A snip of the scissors, and six inches of white beard fell to the floor. For the first time in thirty years Mr. Simpson felt a razor on his face. Then his hair was cut and shampooed; and an hour later he sat gazing at a dark-haired, clean-shaven man in the glass who gazed back at him with wondering eyes.
   >
   > (from *Stepping Backwards* by W. W. Jacobs)

2. Decipher this text with the morbit cipher and keyword `EMPLOYING`.

```
911353121621139543788436129113993731884291132513918967358633131312187915334338784312113948618884218543689313188373918356854346731916136125916318676318858186672956532911338583785139443259113851327942565844673568967358631883133918794218113214673568483893846585885313846434847248135814384435348966727889369543487947698 46
```

3. Break this ciphertext:

```
457534743498616878454984372675677647313764498437267376374754772236464775861646873474647243625494349443713433274294232364627468435763625492743434716233459644366313982364329471631332784961617764433 2
```

Unit 146
Monome-dinome cipher

The *monome-dinome cipher* is a prefix-free variable-length code. The key is a keyword and a permutation of the ten digits. The cipher uses a 24-letter alphabet, so we have to boot out two letters. Usually, we merge `J` with `I`, and sometimes `Z` with `Y` or `S`. The keyword is used to generate a mixed alphabet from the remaining 24 letters. This alphabet is put into a 3×8 grid. The first two digits label the second and third lines, while the remaining eight digits label the columns. The code word for a letter is the row label, if any, plus the column label. Code words that contain one digit are called *monomes*; those with two, *dinomes*.

Let's work through an example, using the keyword `KEYWORD` and the list of digits 5, 9; 3, 4, 1, 7, 6, 8, 2, 0. One way to generate the mixed alphabet (remember that there are many ways) gives us this grid:

	3	4	1	7	6	8	2	0
	K	E	Y	W	O	R	D	A
5	B	C	F	G	H	I	L	M
8	N	P	Q	S	T	U	V	X

If we encipher the message

```
THIS MESSAGE WAS ENCRYPTED WITH A CODE
```

we get

```
96 56 58 97 50 4 97 97 0 57 4 7 0 97 4 93
54 8 1 94 96 4 2 7 58 96 56 0 54 6 2 4
```

Of course, we don't want the spaces to betray our encryption method, so they are removed. The ciphertext is

```
965658975049797057470974935481949642758965605462 4
```

With a ciphertext that is long enough to reliably use statistics, the monome-dinome cipher is easy to break. The two most common digits will be the row labels; the rest will be column labels. Once we know all the code words, it becomes a monoalphabetic substitution cipher, which we can break with the method in Unit 28.

Reading and references

American Cryptogram Association,
www.cryptogram.org/downloads/aca.info/ciphers/MonomeDinome.pdf

Programming tasks

1. Implement the monome-dinome cipher. Allow for the choice of letters that are merged in the mixed alphabet.

2. Write a function to tabulate the frequencies of digits in a ciphertext.

3. Implement the attack mentioned above.

Exercises

1. Encipher this text with the keyword `AUTOMOBILE` and digit list 3, 1; 8, 7, 5, 4, 2, 6, 0, 9. Use standard choices for mixing the alphabet and merging letters.

   > When the windshield was closed it became so filmed with rain that Claire fancied she was piloting a drowned car in dim spaces under the sea. When it was open, drops jabbed into her eyes and chilled her cheeks. She was excited and thoroughly miserable. She realized that these Minnesota country roads had no respect for her polite experience on Long Island parkways.
   >
   > (from *Free Air* by Sinclair Lewis)

2. Decipher this ciphertext with the keyword `HIGHWAY` and digit list 1, 3; 6, 4, 0, 8, 9, 2, 5, 7. Use standard choices for mixing the alphabet and merging letters.

```
814693514915148717157143639417151710173230030149397173215393026143035938391514383861430359304141638714151430261430363017383614304392614306936364151438386143051732151619143838301438173230714386143041212141538143617383845419439414386143018415161514383891516617361410321915143838814930145173215163917614305293961732389151615148394143817109734329415399157141710938381774939417159151617103630416143961419415717191564068924389193014916286939439438415391415161416391751490171916141530179161710361914938323014915163238141032191514383810439192161416479391416915161614383941514163917415383643014903014939369393041739
```

3. Break this ciphertext:

```
480805287681808451276545287548157525345257518153802686152872848212506450257577577480053257578187808153868051259528062 5
```

```
782148050482575284875215256808187808680645765452572878153012
3462648180872845148450813521525165448457287872849525448051
2575765452878115681535081825218045086845652825752128051025 7
4575281280518754266545275425151818052805252515251480652182 1
5262648180468728483864652802686125765426654525981846819348 1
8684280515952505428045025782545280815952802845481865751534 1
8469521525186505251280518481654528214805048242872845081805 0
5218052518746545952505428045025782148050482575284282825745 2
5168128461818081594502578218195752598 4
```

Challenge

```
BEAHAICIHAICGAHBFAFBEBFDGBEAGGCIBFCFHAFDHCGBEBFBGAGAIBEAHGD
ECFAIFAIBEBFAFGFEAIFAIBEBFAFGFHAICGAHBFAFAGAHAIHAHAICIGDFGA
FAIGEFBFFBFCFBIBEAHCGAFBFBHAIDGBHAFBFBFHAEBGBFBFGHAHFBFBEBE
BFAFAEBHBEAHBFCGFGAICFBEBFDGBEAICIBFCFHAICGAHBFAFBFBGGCIBFA
IBEAHBFAFAECFBFAEAFBEAGAEFBFBEBEBFAFCIAICFBEAHBFHAICGAHBFAF
BEBFDGBEBEAHBFGCICIAIBICFCEBFCFBEAEBHHAEBGBFAGAEAFBGCIBHAEA
FBFGHAHCGFGAICFBEBFDGBEFBFBEBEBFAFAICIBGAECFBFAICFGCIDHCIBE
BFCEGBEAIHAGGDHHGCFDHAEDEBHAICFBGBEAHBFIBFDHAGAEAFBGBEAHGBE
AGGCIDECIBFBG
```

Unit 147
Straddling checkerboard cipher

The *straddling checkerboard cipher* is another prefix-free variable-length code and is very similar to the monome-dinome cipher. The key is a keyword and a pair of digits. The keyword is used to generate a mixed alphabet, which is placed into a 3×10 grid. The two digits label the second and third lines, while the digits 0, ..., 9 label the columns. When the alphabet is laid in, the spots in the top row under the digits that match the two row labels must be left empty. There will also be two empty spaces in the third row, or two additional characters can be added, such as space and period. The code word for a letter is the row label, if any, followed by the column label. Code words are monomes or dinomes.

Let's work through an example, using the keyword `KEYWORD` and the pair of digits 5 and 8. One way to generate the mixed alphabet (remember that there are many ways) gives us this grid:

	0	1	2	3	4	5	6	7	8	9
	K	E	Y	W	O		R	D		A
5	B	C	F	G	H	I	J	L	M	N
8	P	Q	S	T	U	V	X	Z		

If we encipher the message

```
THIS MESSAGE WAS ENCRYPTED WITH A CODE
```

we get

```
83 54 55 82 58 1 82 82 9 53 1 3 9 82 1 59
51 6 2 80 83 1 7 3 55 83 54 9 51 4 7 1
```

Of course, we don't want the spaces to betray our encryption method, so they are removed. The ciphertext is

```
835455825818282953139821595162808317355835495147 1
```

With a ciphertext that is long enough to reliably use statistics, this cipher is easy to break. The two most common digits will be the row labels. Working from the beginning of the ciphertext, knowing the row labels allows us to unambiguously break it into code words. Once we know all the code words, it becomes a monoalphabetic substitution cipher, which we can break with the method in Unit 36.

Reading and references

Practical Cryptography,
practicalcryptography.com/ciphers/straddle-checkerboard-cipher
practicalcryptography.com/cryptanalysis/stochastic-searching/cryptanalysis-straddle-checkerboard

Wikipedia, en.wikipedia.org/wiki/Straddling_checkerboard

Fred B. Wrixon, *Codes, Ciphers & Other Cryptic & Clandestine Communication*, New York: Black Dog & Leventhal, 1998, pages 200-201.

Paolo Bonavoglia, "The straddling checkerboard cipher," La crittografia da Atbash a RSA, 2020, www.crittologia.eu/en/critto/straddle.html

Friedrich L. Bauer, *Decrypted Secrets: Methods and Maxims of Cryptology*, 4th edition, Berlin: Springer-Verlag, 2007, pages 56-57.

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, pages 635-636.

Programming tasks

1. Implement the straddling checkerboard cipher.

2. Implement the attack described above.

Exercises

1. Encipher this text with the keyword `AUTOMOBILE` and digits 3, 1. Use the easiest choice for how to mix the alphabet.

   > What I was going to say is this: wouldn't it be much better to turn your car into the means of making an honest living, and at the same time having some rattling good fun, rather than sell the thing for less than half cost, and not only get no fun at all, but not know how to get out of the scrape in which you've landed yourself?
   >
   > (from *My Friend the Chauffeur* by C. N. Williamson and A. M. Williamson)

2. Decipher this ciphertext with the keyword `HIGHWAY` and digits 1, 6. Use the easiest choice for how to mix the alphabet.

```
151818142173563162626215116526363181711951705611015721653 21
711630563620119186415101061265115956156362649063116161212 29
621911111063011196481529217263621621710621171117181018648 63
122364611162630262161863186190516192181756258236263615191 92
173516567181710186118630715116526363262116659631571861515 16
186263621863011102611196318191918622631118126264905192963 64
```

```
611162011262630111618626332611526201812418165171574181611 17
621523063217626356364611162075171062061217142173515161862 63
632162102170116111651161710571521211263262621111162173516 56
165111563056362011951761115151578116301141816517401805621 01
817115151563056363011611191861106262018 4
```

3. Break this ciphertext:

```
714240734049764717070383037176141456714240456713560705340 46
540704040457064634670107280405146714969357131463417142370 70
534046540370146076940972571373403466474740649646540414914 57
442404654037094049373409714267147404941405714932172497442 35
423705426004046240987646146403411467142405146714964976600 71
424047491471703713146707442354237140467246536714070456768 40
940493734093461499404987671424049720407014141149456001235 42
174370371714267145671424045671357037046171494097254097116 23
264671357167271101276714240707600123704556467140654272704 61
714234624070704046713600764640746469341407340497671423462 45
727071704749346241491457142404749346534704014139404671371 76
714240464073404976714234627042172098405647680401418403462 49
409725409711714267147493465347040649407440714240467116945 37
171426717142404046724653671314670141600714240714240149404 57
074371427442354270145646767310724540706494041300409649401 46
076346934940571746767014170676346271426716370 6
```

Unit 148
Prefix-free codes

A *prefix-free code* is one in which no code word resembles the start of another code word. In other words (pun intended), no code word can be mistaken for another and decoding is unambiguous. Unfortunately, or fortunately, decoding is difficult if we do not know the assignment of code words to letters, not because of a possible monoalphabetic substitution, but because figuring out what the distinct code words actually are is difficult. And, sadly, some people call these codes *prefix codes*, for even more confusion.

As an example, suppose we have this code:

A	013	J	01112	S	0002
B	1002	K	1012	T	123
C	10102	L	01002	U	0012
D	1002	M	113	V	00012
E	03	N	103	W	0112
F	00102	O	1112	X	10012
G	1102	P	01102	Y	10112
H	00002	Q	11012	Z	11002
I	002	R	0102		

Look at the code word for A: 013. No other code word begins with that sequence, so the letter A is always recognized without doubt. The same is true for the code words for every letter. (Do you recognize this code? We fixed it to have the *prefix property*, i.e., the property that no code word is the prefix of another.) If we encode a short sentence, then decoding, if we know the table of code words, is easy. We begin at the start and read until we complete a code word. Because of the prefix property, we know that we can safely assign it to a letter and can proceed from there. Take this encoded message, for example:

123000020020002002000020310310102111210020310020112002123000021112001212
30110201020300102002100120300 02

To decode, we read from the beginning: 1. This is not a code word. Read again: 2. But 12 is not a code word. Read again: 3. At this point we have the code word for T and can safely begin reading the next code word, starting with 0. We leave it to you to finish the decoding of this short example.

If we have a fixed-width code, then decoding is already unambiguous. If we start with a variable-length code we can transform it into a prefix-free code by adding a new symbol to serve as an end-of-code-word marker. For example, the space between letters serves as an end-of-code-word

marker if we reinterpret Morse code not as a two-symbol variable-length code but as a three-symbol prefix-free free code.

In the next Unit we will see how to make the most efficient prefix-free code possible, by making code words for the most frequent letters short and those for infrequent letters longer.

Reading and references

Wikipedia, “Prefix code,” https://en.wikipedia.org/wiki/Prefix_code

Programming tasks

1. Write a function that returns a boolean value indicating whether a code has the prefix property. The input can be a dictionary object, a list of twenty-six strings, or an object whose class you define.

2. Create functions, or a class, for encoding and decoding text with a prefix-free code. If you defined a class in Task 1, extend it in this task. Check that a code is prefix-free before performing any encoding or decoding, and raise an exception if it is not.

Exercises

1. Decode using the code defined in the example:

```
01310303001020010200210102002031031231010211121002030020002111210303123
00002013123031031010211121002030002013123031001212312311121230000203000
20000211120102123030002123011021112000200020021002010020300021230102002
10311021112001021010211121002030002101121131002111201002000 2
```

2. Encode this text using this code:

Any symbolic model is based on a limited number of basic terms and a limited number of basic relations between the basic terms.

For the basic terms and only for them, there are accepted descriptive definitions. All the others terms are generated by the model, together with their normal definitions. These definitions are generated by the model by logical and mathematical procedures.

(from *The Brain, A Decoded Enigma* by Dorin Teodor Moisa)

A	01	J	11101	S	20001
B	001	K	111001	T	2101
C	0001	L	1110001	U	21001
D	101	M	111101	V	210001
E	1001	N	1111001	W	21101
F	10001	O	11110001	X	211001
G	1101	P	111100001	Y	2110001

H	11001	Q	201	Z	211101
I	110001	R	2001		

3. Decode this using the same code as Exercise 2:

1100012101110001200010111110000111110000101200110011111001210121011111
0011001210001100120012110001111100011111001100121101110011111000111001 0
1200010001012001100110001210011110001111000121100012000121012100110111 0
0011001101210111001100120001100111110101111100121001200010001200111000 1
1111000012101200012101110010121010111110012110001012101210110011111011 1
1100001210121011111000110110010001110001111100001110011001200121011100 1
1001111101111100011111001210111001100120001210011111000011111000011111 0
0012000111000121011100011111000111110012101110010121012101110011001211 0
0010001111100011111001210101110001111100121012001210011001011110001111 1
0000111001010011001210111000100010001110010120010100012101100120012000 1
1111012100120001210110011111001101110001111100110001011100011110001210 0
1200110011100011000121011100110011111010111110012100120001000120011100 0
1111100001210120001012001100110012100011001200110110010001110001111100 0
0111001100120011001101110001210111110121001200012101001100100121100011 1
1000111110001111100111010111110011011110001010011111000120011100011111 0
0012100120001000111110001111101111100001012001110001200011111000111110 0
1200010111110011011100101111100001111100001211000111012100110012000120 0
0110012000121011100121001200011101011100011111001110001111100111011111 0
0001111100011100011111001210100121100011111000011111000111000111110012 1
0101111100110111110000120011111000100011001100110111000111110011101200 0
1111000111110001211011110001211000101111100110100010121001210111000111 1
1000121001200011110001211000120001210110011111000010012110001200012101 1
0011111000010100010001100111110000121011100011111001110121011100111000 1
2000101200012101200121001100111000121012110111000111100011110001001100 1
0110111110111000121012101100110121011100101210110012100011001200121100 0
1200110010111100011011100012000100011111100012100011001200121100011100 1
1111001200110011101012001101210111110001210111001100111011001111110011 00
1200101111000120001110001110111110011100011000111000100010121011100011 1
1100011111001111100012001210110011111001111100012001111100011000101111 1
0012110001111100011000121011100110012000110010001111100011011100010001 1
0012000111110001200111110001100010111110012110001111100011000121011100 1
1001110001200120001211000111110100111110001111000120001000111001012001 0
1000121011001200120001111100012001100011100011101210012001100120001111 1
0001200110012100011001111100111000111110012001100110001100120011001111 1
0010001100121011111000121011100110011100012001111100001200111110001111 1
0000110012001111100012001101100120011111000120012001100111100010121011 1
0001111100011111001210111110001111100011111001100101111100111110001210 1
1100110012001211011100011110001111000100110011111000111110011001200012 1
0110011111000011101011100011111001100110121011111000121101012001101210 1
1100110011000111000111110010111100011100011111001210110012001111100001 2
0011001210101210111000111110001111 1

4. Break this ciphertext that was encoded with a variable-length prefix-free code. You will have to do at least part of the process by hand.

```
856398480981983851692569618769298118581999176380585631691816
78160876928563960856398183699765169213579656998080635763801
83806357635691676067969850971878396069960485639676312287679
67081605298192606963583760331811692856396085639812995838762
62608398563984899917636160858580992398160876921858563901765
60485639760331818363983152856063981839348563984838560602836
08385533856318583639648758594608161608585639848098191358991
69283639801835878385616056961816087692856083995485639806081
28580992398018380815858596918585639017656049176376033181806
39698363980183838518185392084189605797606756961481606785639
60699671816592287675484608785635696580981980182806081658363
98315284608760876163858560621848460876569608080182806081658
38098196985671298560093606065921854608169608563569616960636
08076069858118158058391229285639606996718165922995484608785
63569658098191358998460876087616385856083629165567838781956
78998184836081818480183133135797608732831844608185639806081
28360485639603283606961659628581569615696185638160876163639
81639123565985639855765569616041736076516928363976087326318
12384639362831845696185639676087853608728580992392876716928
58099239299161819928560631899101858539460818580992392876783
15285809923929963128362605392635836957969980811858539587838
58563969439802608069167606983858160878378160801830317651831
85181018181938063576348156163859699206085638563963981609838
36085639846487585946081616085856395816487181819356569608080
63185846087819856356965569611060878583152858099239287670878
55855583698583606960636080760698581181580583976069855698792
85809923929954585801838360585675616385091692545858098198360
58580608732090878518358558369855851569858563185833606157580
18385635696556961135798315289981846260358593848063576358385
63909838580184608785604856358380606 02
```

Unit 149
Huffman code

A *Huffman code* is a variable-length prefix-free code. Its construction is based on the frequencies of the letters in such a way that it is the most efficient code. By efficient, we mean that there is no way to encode a long English text with a code that takes less space, given the number of different code symbols.

Let's see how the code words by doing an example of constructing a binary version of the code for this set of frequencies:

A	0.1253
B	0.1204
C	0.0659
D	0.1802
E	0.2945
F	0.2137

The first step is to put them in order, in descending order by frequency:

E	0.2945
F	0.2137
D	0.1802
A	0.1253
B	0.1204
C	0.0659

Now we collect the lowest few frequencies. How many we collect depends on the number of code symbols. Since we are constructing a binary code, we take two symbols and we collect the lowest two frequencies.

E	0.2945	E	0.2945
F	0.2137	F	0.2137
D	0.1802	D	0.1802
A	0.1253	A	0.1253
BC	0.1863	B	0.1204
		C	0.0659

Are the frequencies still in correct order? No, so we need to reorder them again before we collect the two new lowest of the new frequencies:

E	0.2945	E	0.2945	E	0.2945
F	0.2137	F	0.2137	F	0.2137
BC	0.1863	BC	0.1863	D	0.1802
				A	0.1253
DA	0.3153	D	0.1802	B	0.1204
		A	0.1253	C	0.0659

Reorder the frequencies, then collect the new two lowest frequencies:

DA	0.3153	DA	0.3153	D	0.1802	D	0.1802
				A	0.1253	A	0.1253
E	0.2945	E	0.2945	E	0.2945	E	0.2945
FBC	0.4000	F	0.2137	F	0.2137	F	0.2137
		BC	0.1863	BC	0.1863	B	0.1204
						C	0.0659

Lather, rinse, and repeat:

FBC	0.4000	FBC	0.4000	F	0.2137	F	0.2137	F	0.2137
				BC	0.1863	BC	0.1863	B	0.1204
								C	0.0659
DAE	0.6098	DA	0.3153	DA	0.3153	D	0.1802	D	0.1802
						A	0.1253	A	0.1253
		E	0.2945	E	0.2945	E	0.2945	E	0.2945

One last time:

DAEFBC	1.0000	DAE	0.6000	DA	0.3055	DA	0.3055	D	0.1802	D	0.1802
								A	0.1253	A	0.1253
				E	0.2945	E	0.2945	E	0.2945	E	0.2945
		FBC	0.4000	FBC	0.4000	F	0.2137	F	0.2137	F	0.2137
						BC	0.1863	BC	0.1863	B	0.1204
										C	0.0659

What we have done is create a binary tree. As we move from left to right, whenever we split a row into two, we assign a 0 to the upper one and 1 to the lower one. To find the codeword for a particular letter, start at the far left with an empty codeword. Travel rightward and add the appropriate digit at each row splitting until we reach the desired letter. The resulting code is this:

A	001
B	110
C	111
D	000
E	01
F	10

To build a code for English, we have to use the table of twenty-six letters and their frequencies, but the process is the same as above.

Reading and references

David A. Huffman, “A Method for the Construction of Minimum-Redundancy Codes,” *Proceedings of the I.R.E.* 40 (1952) 1098-1101.

David W. Gillman, Mojdeh Mohtashemi, and Ronald L. Rivest, “On Breaking a Huffman Code,” *IEEE Transactions on Information Theory* 42:3 (1996) 972-976.

Programming tasks

1. Use your table of English monogram frequencies to construct a binary Huffman code.

2. Repeat Task 1 to construct a ternary Huffman code, i.e., one using digits 0, 1, 2.

Exercises

1. Encode the following text from *The Efficiency Expert* by Edgar Rice Burroughs in the Huffman code you constructed in Task 1 above.

   > “What's an efficiency expert?”
   >
   > “Oh, he's a fellow who gums up the works, puts you three weeks behind in less than a week and has all your best men resigning inside of a month.”

2. Repeat Exercise 1 with the code you constructed in Task 2.

Part IX
Miscellaneous ciphers

Unit 150
Symbolic substitution

A *symbolic substitution cipher* is like a monoalphabetic substitution except that the ciphertext symbols are not letters. We can attack such a cipher by *transliterating* it, i.e., by assigning a letter to each symbol, then applying the hill-climbing attack from Unit 36.

Reading and references

Fred B. Wrixon, *Codes, Ciphers & Other Cryptic & Clandestine Communication*, New York: Black Dog & Leventhal, 1998, pages 173-180 and 183-184.

Simon Singh, *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography*, New York: Random House, 1999; see the section "The Babington Plot" in chapter 1.

Edgar Allan Poe, "The Gold-Bug," 1843, en.wikisource.org/wiki/Tales_(Poe)/The_Gold-Bug
Edgar Allan Poe, "The Gold-Bug," 1843, en.wikisource.org/wiki/Tales_(Poe)/The_Gold-Bug, www.eapoe.org/works/tales/goldbga2.htm

Arthur Conan Doyle, "The Adventure of the Dancing Men," first published in 1905, now in *The Complete Works of Sherlock Holmes*, London: Simon & Schuster, 2012.

Fletcher Pratt, *Secret and Urgent: The Story of Codes and Ciphers*, New York: Bobbs-Merrill, 1939, chapter VII, section II.

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, pages 90-91, 93-94, and 174-176.

S. Tomokiyo, "First Codebreaking in the American Revolution — Benjamin Church's Cipher," cryptiana.web.fc2.com/code/church.htm, 2009-2014.

Benjamin Church, Jr., George Washington Papers, Series 4, General Correspondence: Benjamin Church Jr. to Maurice Cane, July 1775, www.loc.gov/item/mgw443691

Friedrich L. Bauer, *Decrypted Secrets: Methods and Maxims of Cryptology*, 4th edition, Berlin: Springer-Verlag, 2007, pages 44-45.

Johannes Trithemius, *Polygraphiae libri sex*, Reichenau: Joannis Haselberg de Aia, 1518, www.loc.gov/item/32017914, book 6.

Blaise de Vigenère, *Traicté des chiffres ou secrètes manières d'escrire*, Paris: Abel l'Angelier, 1586,

HDL: 2027/ien.35552000251008, gallica.bnf.fr/ark:/12148/bpt6k1040608n, gallica.bnf.fr/ark:/12148/bpt6k94009991, pages 286-343.

Martin Gardner, *Codes, Ciphers and Secret Writing*, New York: Simon & Schuster, 1972, sections 4, 6, and 7.

Exercises

1. This is the ciphertext from Edgar Allan Poe's "The Gold-Bug" (the original version). Break it.

```
53‡‡†305))6*;4826)4‡.)4‡);806*;48†8
¶60))85;1‡(;:‡*8†83(88)5*†;46(;88*96
*?;8)*‡(;485);5*†2:*‡(;4956*2(5*—4)8
¶8*;4069285);)6†8)4‡‡;1(‡9;48081;8:8‡
1;48†85;4)485†528806*81(‡9;48;(88;4
(‡?34;48)4‡;161;:188;‡?;
```

2. This ciphertext uses the symbols from the Sherlock Holmes story "The Adventure of the Dancing Men." Some symbols did not appear in the story and were designed by the font's creator. Break the ciphertext.

3. This ciphertext is from the 2002 British National Cipher Challenge. Break it.

4. What does the note say?

5. >ΠΓV ΓV >ΠΩ ˥Γ˥˙ΩΩ LΓ˙ΠΩF ⅃L˙VE ⊔ΩEƎΩ ⅃V >ΠΩ
⊏FΩΩƐ⅃VEΩV LΓ˙ΠΩF EF >ΠΩ FEVΓLF<LΓ⅃ΩV LΓ˙ΠΩF <E<
˥FE⊔⅃⊔L˙< ⅃L˙FΩ⅃Ɔ< ⊔ΩEƎ ⅃⊔E<> Γ> VΓΩLΩ LΠΓL˙ƆFΩΩ
<VΩ Γ> E⊏>ΩΩ Γ>V VΠ⅃˥ΩV ⅃FΩ ⊏FE˙ ⅃ >ΓL>⅃L>EΩ
⊔E⅃FƆ ⅃ΩƆ ⊏FE˙ ⅃Ω > ⊔E>Π ∀Γ>Π ⅃ΩƆ ∀Γ>ΠE<> ⅃ ƆE>

6. What is Ozzy trying to tell us?

Unit 151
One-time pad

The *one-time pad* (*Vernam's cipher*) employs a key that has the same length as the plaintext. The key is combined with the plaintext to produce the ciphertext, usually in a Vigenère-like manner. Because for any ciphertext there exists a key that can decipher it to *any* plaintext, the one-time pad is the only provably secure cipher. The problem with it is in key distribution: The sender and receiver must have access to the same key. Sending the key introduces insecurities, and carrying a large codebook of pregenerated key material is inconvenient.

We can view the one-time pad as a limiting case of other ciphers. It can be seen as a Vigenère cipher in the limit of a long key or infinite period. It can be viewed as a stream cipher in the limit of perfectly random key generation.

The key must *never* be reused. If two messages encrypted with the same key are intercepted, then the security of the cipher is destroyed. Also, the key must be random, lest the adversary find a pattern in your key.

Reading and references

Claude E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal* 27:3 (1948) 379-423, DOI: 10.1002/j.1538-7305.1948.tb01338.x, HDL: 11858/00-001M-0000-002C-4314-2

Wikipedia, en.wikipedia.org/wiki/One-time_pad

users.telenet.be/d.rijmenants/en/onetimepad.htm

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, pages 394-403.

Programming tasks

You can just use your routines for the Vigenère cipher for this one.

Exercises

1. These five ciphertexts were encrypted with the same key. Recover the texts and the key. This may require some trial and error.

DNBYDYWSXBNXHPRWYBHMSGHEXVRINUBSQIXEBHBQKUUHAFHXWJSNSNJGALT
YUVCGLXZIWSD

DNBTZHDGGAYBRSWVFZABDLHTNXKLQLINQKAXBCPPPWIQBIXERVIZTLJNIWV
LJQ

SZTCNAOOOWNIZXBXVEHMUDQANIFYAZMFROQYTLTGUWTGJIOKFJPEYCKBELE
RWRBWNYVSJU

DNBHMZQHJXISVEBEGMUUMCSJRWKHUZTKAKUXXTSBPVBGYPYUDNMSFLHBHSZ
QMSJFCNNG

KYPVZAKOTUYGSXQUJYBUQDSJRMDMNUFYCZKYCZSCOLYCWFWKXTFNDAJRHSU
NWVWLSMXD

2. These two ciphertexts were encrypted with the same key. See if you can recover the texts and key.

PPHKLXETAGOTLJGODXRODJBSJQGYGXJLGTKSGPHBTFEJSDXKEESUWLDDPVC
VPKPZQMGFAEFTQLUKSBEJLHHHSKZFXNOYIDGMQOICSMZQQIUFYQBGCEZXVD
JTBOGTXDRPWRIDTORXABUGDGIRGGBYHYRIRIJEKTLOMKIMKERZHFGZVCMOO
RRAMIHIRERZCPLCYFJKWPQRSHKTQIHBAYZXLBZOUJZRXNYBGIEVWJUXBSTO
GWABQHGHPUCRISHTSTHDCCRAWABIZPEOWQVOBJYYSCNUUPYUFOVBZAOSCSO
DBKWGWRRVUQQRHDVHTJYSLGZRHWAKWYWTNPITYMPCQSRUKAHSJLKCZYEIDO
EDVAHFIQGHNYODMLOTWANYAGXIHUVSNMGVLKYHBJTNUSVFFAFNKNMBMZPCN

HRETWPKMQUJEBMKEIMIKXNEEMHOZIETHUNQFKTKWPERUSDTRDTWJIARUGMO
SLDCRFTHPMOJTRHWDEIYUKUCQNRFGRJDGFOABXJCMPNRPBUUPRHMKXOGFCF
JEJFWPKGRPMGPGCKHBTFPVUPIIKBQSBFEJQQCENZWDUWIXUOVXSMMVACYCY
RWCAQOGICTCKWXJGIVPVQQUNYXJBMHDIIXDLYJQRTGECARZHPPCHMFDRCUP
JHIXDRXNPRUDQGVRPDQTVEEFBZVNNZEPCURLANGYFOPVUBLRPVAEUMJZHAU
DAKQUBHLKBSFLEGYJCFDSXDCXHUULMIAWDJXESREIEWQCGECWCFLGMMEDTX
FCMSFSDWTSZWAXCIZEJIJANPAWLWTFMCAEBKMZFUUYVPVAPPDCPTLVXOMRO
VANDAUH

Unit 152
Slidefair cipher

The *slidefair cipher* is a periodic digram substitution cipher. The mechanics of the cipher involves sliding one copy of the alphabet against another. The plaintext is divided into digrams, with padding if necessary. For each letter of the keyword, the lower alphabet is slid until that letter is under the A of the upper alphabet. The two plaintext letters in a digram form the corners of a rectangle, with the first in the upper alphabet and the second in the lower. The ciphertext letters are taken as the other two corners of the rectangle, again with the first in the upper alphabet and the second in the lower. If the two plaintext letters are in the same column, then the ciphertext letters are taken as the two letters in the column immediately to their right.

As an example, let's encipher this message with the keyword FAIR.

THIS MESSAGE WAS ENCRYPTED WITH SLIDEFAIR

First, divide the plaintext into digrams:

TH IS ME SS AG EW AS EN CR YP TE DW IT HS LI DE FA IR

To encrypt the first digram, TH, we line up the alphabets so that the first letter of the keyword is under the A in the upper alphabet.

```
      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
. . . 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 A B C D E F G H . . .
```

The ciphertext letters are read from the same columns as the plaintext letters, starting with the one in the upper alphabet: CY. For the next digram, we use the second of the keyword, A. This continues as we cycle through the keyword. The final ciphertext is

CYSIWUBJBFWEKIWVMHPYWBFUONSHATNUVKRI

There are three variations of the slidefair cipher, corresponding to the three ways of arranging plaintext and ciphertext alphabets in the Vigenère, Beaufort, and variant Beaufort ciphers. The example above used the Vigenère-like variation. For the variant-Beaufort variation, we use the same prescription, with the exception that the keyword letter is taken in the upper alphabet and aligned with the 'A' in the lower alphabet. For the Beaufort variation, the lower alphabet is written in reverse order.

If we have a ciphertext that is sufficiently long, we can find the length of its keyword with the method of Unit 39. The period that we find with that method is twice the length of the keyword, since we encipher twice the number of letters in one period.

A hill-climbing attack on the slidefair can be done in the same way as we did for the Vigenère cipher in Unit 51, by varying each letter of the keyword to improve the fitness of the plaintext. When we can no longer increase the fitness by changing any letter in the keyword, we have likely found the correct one.

Reading and references

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; page 199.

American Cryptogram Association, www.cryptogram.org/downloads/aca.info/ciphers/Slidefair.pdf

Programming tasks

1. Implement the cipher. Allow for the option of choosing one of the three variations.

2. Implement a dictionary attack.

3. Implement the hill-climbing attack described above. Allow for the choice of the three variations of the cipher.

Exercises

1. Encipher this text with keyword `RULER`. Do it three times, once for each variation of the cipher.

   > The slide rule is a device for easily and quickly multiplying, dividing and extracting square root and cube root. It will also perform any combination of these processes. On this account, it is found extremely useful by students and teachers in schools and colleges, by engineers, architects, draftsmen, surveyors, chemists, and many others.
   >
   > (from *Instruction for Using a Slide Rule* by W. Stanley)

2. Decipher this ciphertext with keyword `CIRCLE` and the Vigenère-like variation of the cipher.

   ```
   FVVMCRNCTCAWRPVMWZFVHTNEGVDKBVMJIHWLUVAPNCFVTPNEGNKMBKGNTOQ
   VCNOAAVMGIQALGEUZVMPCNEATFVDMCRPGADZMPGDCBVDQWEOITGFMNKLGWE
   AGRPJCFPPGDPPJCJWZDUJKCTWISUHGGV
   ```

3. Decipher this ciphertext with keyword `ARGUE` and the Beaufort-like variation of the cipher.

   ```
   NMNPTSBUSWHWNKCPQYANEHLDSSMRMATEBDGMDDTAWPROKOIGRAVMNYKTJME
   TXNKVTCNBGAATEONSMNYRPWNZSNAENEWJRGSFBAXLCWNMVVGBWZTUJYATTU
   KQHHNKPYANDMNAZOOGGBXIYSVKOGNBMAPSJGOCGBLZWTUF
   ```

4. Perform a dictionary attack on this ciphertext. The keyword is a common five-letter English word.

```
GMUIVVNAVLFQRCQKPEGPCRYFWRECXDGFQPXWERWEQQDIJKNICZVFNSJFDRP
DRZAZTFISIYHAQVJWRIULSQUZPEFOGATAGCMEEMPWQZAVHCOCTXCEQJQIMI
CTWTEGALEVDJHRQVDSTSXWUZXDLRYEHPVFEWBZAVXYMFMTAZONPECYMFYCA
LTNSOFFRRLNCJFAYOAVYRCZHQAVPLVHGPBKUCHJHTRPFGMHWZIWAWSNBIAG
SECPZFMMMEEHXHZXQPDKNRCTCSQJQIAEPORZQVUVASSYFOUMPEOHPHSDAMG
L
```

5. Find the length of the keyword used to encrypt this ciphertext. Break it with the hill-climbing attack.

```
UGSYGABUJSNMQFARCNMMNUKPJTMLUGJYQQLJHEXGVAVGZMAAJBWINDBFYNA
GHXOJEPJTQLABYOBUNHHRCPMZVSYLRFFJPMRJBFRUOASHEOYEXKAJUGTYZB
BBPINYMVVPGGDXHPPRWZPLFROFSPTSAPNICKGYUMVFSBPMEJTSRHYCSZQBO
ICKIZBGZZSTDNPRURCEGYVCWXZDEXWF
```

Unit 153
Nicodemus cipher

This is a real dog of a cipher. To encipher a text with the *Nicodemus cipher*, first encrypt with a Vigenère cipher, then break the text into blocks of five times the key length and apply a columnar transposition to each block, using the same key as the Vigenère. Here is a short example:

```
KEY        KEY        EKY
____       ____       ____
THI        DLG        LDG
SME        CQC        QCC
SSA        CWY        WCY
GEI        QIG        IQG
SEN        CIL        ICL

CRY   →    MVW   →    VMW
PTE        ZXC        XZC
DWI        NAG        ANG
THN        DLL        LDL
ICO        SGM        GSM

DEM        NIK        INK
US         EW         WE
```

The ciphertext is read off one block at a time, including the final, incomplete block:

```
LQWIIDCCQCGCYGLVXALGMZNDSWCGLMIWNEK
```

Variations on this cipher can use a different block size, or replace the Vigenère cipher with Beaufort, variant Beaufort, or any other polyalphabetic cipher.

Here is how to break a ciphertext encrypted with Nicodemus:

1. Guess the length of the keyword. You probably guessed wrong, so try a range of values.
2. Reconstruct the columns by taking five characters from each block. Ignore the final, incomplete block for the time being.
3. For each column, decrypt it as a Caesar shift cipher. Use the frequency-matching technique from Unit 24. You will, after this step, have a text that is only encrypted with a columnar transposition and a keyword that has been scrambled with the same permutation.

4. Break the text you found in step 3 as a columnar transposition cipher. Use the hill-climbing technique from Unit 93.

5. Use the key from step 4 to unscramble the keyword.

6. Use the keyword to decrypt the full ciphertext. Now the final block is included, and you get the full plaintext.

Reading and references

Helen Fouché Gaines, *Cryptanalysis: a study of ciphers and their solution*, New York: Dover, 1956; previously titled *Elementary Cryptanalysis* and published by American Photographic in 1939; archive.org/details/cryptanalysis00gain; page 216.

American Cryptogram Association, "The ACA and You," www.cryptogram.org/cdb/aca.info/aca.and.you/aca.and.you.pdf, 2005 edition: web.archive.org/web/*/http://www.cryptogram.org/cdb/aca.info/aca.and.you/aca.and.you.pdf, 2016 edition: web.archive.org/web/20160418014322/http://cryptogram.org/docs/acayou16.pdf; the relevant page is also at www.cryptogram.org/downloads/aca.info/ciphers/Nicodemus.pdf

Programming tasks

1. Implement an encryptor and a decryptor. Feel free to import your routines for Vigenère and columnar transposition ciphers.

2. Implement the attack. The key length can be an input, or your program can try a range of key lengths. Again, feel free to import some of your other routines.

Exercises

1. Encipher with the keyword `GEOMETRY`:

   > I will not, from henceforward, talk to any squarer of the circle, trisector of the angle, duplicator of the cube, constructor of perpetual motion, subverter of gravitation, stagnator of the earth, builder of the universe, etc. I will receive any writings or books which require no answer, and read them when I please.
   >
   > (from *A Budget of Paradoxes* by Augustus De Morgan)

2. Decipher with keyword `MURDER`:

```
KWRQDIMJXQAQFSMKPGRJMGRSNHLICFKHHBHLRRXVTPDRQTVBSZRSFYTCVIZ
ZQORNWXIHMLQDZYSZRTJWVJVEVQACOIZBDUXGLVCMOXPRMFEVVSFRRVCLIS
NLVRKIORYIXWFEQAQZPKFRVJKYRBNYJVLNITERISYVYNAYFVFWURXEIZFVL
ZYYOPKVUENITEMREQFGIKPFKLMFTXINLZVUOHKRSSMIDEUFTNWJYKVDWXNI
NNSNHXFBUWRSPITDUZKCFXIGLULRRILHNMWWDPLFLKEHPZUQNLRVKJVCIGV
```

```
IYEUYHDDWDLPWSJESMGQYSNVJZJJSVNBGLUWHDKFJXWEPEDQQPFYNPFFIMI
ZJNYLNWFRWJWLAWVYDEZOZEIRRSVZLVVUAABLHKQWJIHMMADUTKLKIXNHGI
```

3. Break this ciphertext:

```
ZIMSHCUJAROQDMOVOFSNZJVXRDEKFLWRQLEANXAFKXOQISWRWOLZLKSJUKS
MSSMEICJBJCDTXOCHMCGCGBKJVDWOQLLRMKRQJWKNDSOIKOGQZBVTMIUOGS
WUMISHSXWPBNPQTKVBUSVUUFEKIAJWZQIELXRXNWRTZBDCYTHUFSIKVAYHA
XGAKQVJEVNRCNOSKKBVVUVQRRNZTNUAJCRAXVGNJTQMSNNMYFBSSHZZFFFL
FLOEIIIIRNRUAURQNCKHBCSMKOUEUZHHLKXLSHIFJDRDZDUYCSMZQGRVWVP
OTFEWACIKWJYORFRGJVJSD
```

Unit 154
Hutton 1 cipher

The *Hutton cipher* was invented by Eric Bond Hutton as a diversion and later to challenge strangers on the internet. Its key is a pair of keywords; one generates a mixed alphabet (where the remaining letters are added starting with the beginning of the standard alphabet), while the other serves as a set of shifts that are applied periodically (for this cipher, `A` = 1, `B` = 2, ..., `Z` = 26). This may sound like a quagmire 3 cipher, but there is an added complication: after each letter is enciphered, the plaintext letter and ciphertext letter swap position in the alphabet key.

As usual, we will work out an example. Suppose we want to encipher this message with keywords `HUTTON` and `CIPHER`:

```
THIS MESSAGE WAS ENCRYPTED WITH HUTTON CIPHER
```

We begin by mixing the alphabet with the first keyword. Remember that for this cipher, we add the unused letters starting from the beginning of the standard alphabet.

```
HUTONABCDEFGIJKLMPQRSVWXYZ
```

The first plaintext letter is `T`, and the first shift is `C` = 3, so the first ciphertext letter is `A`:

```
HUTONABCDEFGIJKLMPQRSVWXYZ
   3→
```

Then we swap the two letters in the alphabet key:

```
HUAONTBCDEFGIJKLMPQRSVWXYZ
```

The next plaintext letter is `H`, and the next shift is `I` = 9, so the second ciphertext letter is `E`:

```
HUAONTBCDEFGIJKLMPQRSVWXYZ
```

So we swap those letters:

```
EUAONTBCDHFGIJKLMPQRSVWXYZ
```

Next comes `I` which we shift by `P` = 16. We wrap around and get `A`.

```
EUAONTBCDHFGIJKLMPQRSVWXYZ
```

This continues until we have the full ciphertext:

AEAIVQTKANRFZVABZIJZITDURFVLZVSZDPUC

Reading and references

Eric Bond Hutton, “Hutton Cipher,” en.wikipedia.org/w/index.php?title=User:Eric_Bond_Hutton&oldid=840686562

“Hutton Cipher v1,” huttoncipher.netlify.com

Girkov Arpa, “How It Works,” hutton-cipher.netlify.app/howto.html

Programming tasks

1. Implement the cipher.

2. Implement a dictionary attack.

Exercises

1. Write an essay arguing why or why not the inventor should have used `A` = 0, `B` = 1, ..., `Z` = 25.

2. Encipher this text with Hutton 1 and keywords `GEOLOGY` (mixing) and `EARTH` (shifts).

   > We know little of the earth’s internal parts, or of the materials which compose it at any considerable depth below the surface. But upon the surface of this globe, the more inert matter is replenished with plants, and with animal and intellectual beings.
   >
   > (from *Theory of the Earth*, Volume 1 by James Hutton)

3. Decipher this ciphertext with Hutton 1 and keywords `VALLEY` (mixing) and `HILL` (shifts).

   ```
   BSDULXXUPXKHLZXVJRKOQZVXUVOGQHZAGHSSELJXQSKHVLHMBEDGCCAXNAY
   IAZGFNMVFFEEUFPGIYEIDZTPJYNNBLGSNXUYVTWYTIFLUOJTNRTTPLBDEUP
   JMVMNMWAQRMTTQPLBBOGDLUIUJTXZIZXUOBDPDABMNNUJJAMUYDDDCXWYFH
   DHWPNUALTPYYMKVTGJIXFBMXMGIWNZEHDUAFCWZKIIHQIQVEWKIEFILDPEB
   SGNUOCFDQMEWXXTLJVS
   ```

4. Perform a dictionary attack on this ciphertext. It was encrypted with Hutton 1. Both keywords are five-letter English words.

   ```
   ZVFPHUFWADZGGCAXLVLCWBKQGYMFKFWWWHHNOLADQIRTEZHPBHKFSUUQZFN
   MQRLMGCVZFWEJYIGYNCMGAVLXPAUQHKCRCXGENGTKKFUILCKKMOQACODGRS
   JYMJHYXKMVBMQFZQULXZMUQXMHWAFITBDLSIMIHCGLVAIGLICLRVBNNZFPG
   JYJZPLQIGNUZASDGJVGPNGGHEXZCYVCNCLLWECGREILNCNOKYKLSCGJJWPU
   WUOON
   ```

Unit 155
Attacking Hutton 1 with cribs

The Hutton 1 cipher has two features that we can exploit in a crib-based attack. First, the cipher encrypts a letter to itself (a *fixed point*) only when the shift-key letter is Z. Second, the H1 is built on the quagmire 3 cipher, and we understand that cipher nauseatingly well.

Here are the steps in our attack:

1. Find the correct placement of the crib. We do this by looking for a placement such that no letters of the text are enciphered to themselves, or that letters enciphered to themselves only occur in a regular fashion due to (perhaps more than one) Z in the shift key. There may be more than one placement that satisfy this condition; they should all be checked until a good decryption is obtained. Step 4 is where failure is most likely to occur with an incorrect placement.
2. Remove the effect of the swapping of letters that occurs after each letter is enciphered with the Hutton 1 cipher. We are able to do this because we know the letters for the crib and its corresponding segment of the ciphertext. Once the swapping effect is removed, what remains is a plaintext-ciphertext pair for a quagmire 3 cipher.
3. If step 1 involved one or more Z's in the shift key, then we know the length of that key. Otherwise, guess the length. If we guess incorrectly, then step 4 below will likely fail and we will guess again. But if we guess an integer multiple of the correct length, step 4 can succeed and we can get meaningful results.
4. Use the results of step 2 and the length of the shift key to partially fill a key table for the Q3.
5. Use techniques that we saw in Part III to expand our knowledge of the Q3 keys until we can determine the shift key and the state of the register at the start of the crib.
6. Take the register and shift key at the start of the crib and run the H1 cipher backwards to find the register and shift key at the start of the ciphertext. To run the cipher backwards, take the portion of the ciphertext that preceeds the crib, reverse the shift key, and *encrypt*.
7. Use the register and shift key from step 6 to decrypt the ciphertext.
8. Resolve a twelve-fold degeneracy in the register (and another 26-fold degeneracy by rotating it).
9. Use the decrypted text, the ciphertext, and the resolved register at position 0 to find the shift key again. This is necessary because we have a new (equivalent) register, and so the shift key must be revised. Alternatively, we can find the new shift key from the old one with some modular multiplication.

Time now to go through an example in all its gory details to see how this all works. Consider this ciphertext and this crib:

```
FCFGCIFNPODEQLXJXIUWFPSFBAYWZWVQUBOITCDLEWJMSLDWUKLZNQXBLJB
AMDQJOLOMTBNFHLZMUESUFJHDJEPYQNWCJMMRHCBVAZEQUMNVTGMPOYEEGO
GRLWDVDWLGWYLQEJFVMLUPPBOFFFPBHEWJNQQVGJHBCIIKWMAPXQZUKAUJJ
EPCIXCDIPQYUTIZCAWGFZJZWLGHQUBXZPJKMVZEQJWGVIEPKRZLGXHSPEMQ
XPDHCWMXYVZBKNQQVYMEDGDOJEVROCHCRWFLZCSCCRLXVZRYKZCNUEGBETW
PGAHENWSBTCAFLAGMONNPELTNQJRQFSGQTREGYYOWRJVHXVAYBSGXKMIDLG
UTEBSFMCSTOQTTKSEPBYRYNQBWJRWHPRDHHXWNZJSVBOYWEJPMHFXOMXNJS
RPZZAXXP
```

crib: `FOURKINDSOFBATSHAVEBEENFOUNDTWOOFTHESEAREHERER`

The first step is to find a consistent placement. Start at the beginning, where we have several fixed points, but they do not occur with an even period:

```
FOURKINDSOFBATSHAVEBEENFOUNDTWOOFTHESEAREHERER
FCFGCIFNPODEQLXJXIUWFPSFBAYWZWVQUBOITCDLEWJMSLDWU...
```

Move on to the next position:

```
 FOURKINDSOFBATSHAVEBEENFOUNDTWOOFTHESEAREHERER
FCFGCIFNPODEQLXJXIUWFPSFBAYWZWVQUBOITCDLEWJMSLDWUK...
```

There may be more than one possible placement, but the one we want is at position 123 (counting from zero):

```
      FOURKINDSOFBATSHAVEBEENFOUNDTWOOFTHESEAREHERER
...LWDVDWLGWYLQEJFVMLUPPBOFFFPBHEWJNQQVGJHBCIIKWMAPXQZU...
```

The next step is to take the crib and its corresponding segment of the plaintext and remove the effect of the swapping of letters that occurs after the encipherment of each letter. One way to do this is to maintain a monoalphabetic substitution key as we proceed. Start at the beginning with the unscrambled alphabet (*e*) as that key:

substitution key: `ABCDEFGHIJKLMNOPQRSTUVWXYZ`

With this key, the first letters `F`→`F` and `V`→`V`.

```
FOURKINDSOFBATSHAVEBEENFOUNDTWOOFTHESEAREHERER
VDWLGWYLQEJFVMLUPPBOFFFPBHEWJNQQVGJHBCIIKWMAPX
```

Update the substitution key by swapping `F` and `V`:

substitution key: `ABCDEVGHIJKLMNOPQRSTUFWXYZ`

With this key, the second letters O→O and D→D. So there is nothing to do with the texts, but we must update the substitution key by swapping O and D.

```
FOURKINDSOFBATSHAVEBEENFOUNDTWOOFTHESEAREHERER
VDWLGWYLQEJFVMLUPPBOFFFPBHEWJNQQVGJHBCIIKWMAPX
```

substitution key: ABCOEVGHIJKLMNDPQRSTUFWXYZ

No changes happen to the texts for the first five positions, but the substitution key becomes

substitution key: ABCOEVKHIJGRMNDPQLSTWFUXYZ

Under this key, the sixth letters I→I and W→U. The substitution key is updated by swapping I and W (the two original letters).

```
FOURKINDSOFBATSHAVEBEENFOUNDTWOOFTHESEAREHERER
VDWLGUYLQEJFVMLUPPBOFFFPBHEWJNQQVGJHBCIIKWMAPX
```

substitution key: ABCOEVKHWJGRMNDPQLSTIFUXYZ

The next pair do not change in the text, but the substitution key is updated by swapping N and Y.

substitution key: ABCOEVKHWJGRMYDPQLSTIFUXNZ

Next, D→O and L→R and we swap D and L in the key.

```
FOURKINOSOFBATSHAVEBEENFOUNDTWOOFTHESEAREHERER
VDWLGUYRQEJFVMLUPPBOFFFPBHEWJNQQVGJHBCIIKWMAPX
```

substitution key: ABCOEVKHWJGRMYLPQDSTIFUXNZ

This process continues until we have this pair of texts:

```
FOURKINOSDVBATQHFADDJBYYDHBRMRESAVHBOMPLCBGPTU
VDWLGUYRQEJJFMOWPFJEBJBAEWJIVJSEFKMMDCUPGJTUYX
```

The next step is to extract what information we can from the above text pair. If we try to do so with shift-key length six or seven, the process fails due to an inconsistency or an illicit double letter in a row. But for key length five, we obtain these partial keys:

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	.MG.SV.WUB.....XJ....
1	.J...P.....YD...E.....
2	F.....T.....C.R..I..W...B.
3	...J.....V..U.LQM.K..A.
4	...E...M..GP....OJ.Y.....

The techniques we learned in Units 74 and 75 can now be used to fill out the table. We get:

	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ	order
0	EMGOSVRWUBPCDLTFAZYIXJQKNH	13
1	QJLMAPCZTVXNBYDKWGEOIFHUSR	13
2	FLOGVSTXHNADCMREPIJZWYKQBU	26
3	HFYJWXNGDPISVEBUZLQMOKRTAC	13
4	TQFEIZVMNWGPAKSROJUYLHDCXB	26

We need an alphabetic permutation with order 26 in order to unambiguously get the contents of the register. Write row 2 as a cycle by starting with A, which goes to F, which goes to S, etc.:

AFSJNMCORIHXQPEVYBLDGTZUWK

This is the content of the register at the start of the crib (position 123). To find the shift key, we use this register and the first five letters of the original text pair (not unswapped). The first pair of letters are fourteen steps apart, so the first letter of the shift key is N. Then we swap F and V in the register, as if we were enciphering with H1.

FOURK
VDWLG

AVSJNMCORIHXQPEFYBLDGTZUWK

Next come O and D, which are twelve steps apart. This gives key letter L, and we swap O and D in the register.

FOURK
VDWLG

AVSJNMCDRIHXQPEFYBLOGTZUWK

Next come U and W, which are one step apart. This gives key letter A, and we swap U and W in the register.

FOURK
VDWLG

AVSJNMCDRIHXQPEFYBLOGTZWUK

Next come R and L, which are ten steps apart. This gives key letter J, and we swap R and L in the register.

FOURK
VDWLG

AVSJNMCDLIHXQPEFYBROGTZWUK

The last pair that we need are K and G, which are twenty-one steps apart, for key letter U.

FOURK
VDWLG

So now we know that the shift key at the start of the crib (position 123) is NLAJU. We can take this shift key and the register at position 123 and use them to decrypt the ciphertext starting at that position, through to the end of the entire ciphertext. Doing so at this point in time is a good sanity check to verify that we have a working set of keys.

We now need to run the cipher backward to the start of the ciphertext. We can do this without thinking too deeply about it. Take the portion of the ciphertext that appears before the placement of the crib and reverse it.

```
FCFGCIFNPODEQLXJXIUWFPSFBAYWZWVQUBOITCDLEWJMSLDWUKLZNQXBLJB
AMDQJOLOMTBNFHLZMUESUFJHDJEPYQNWCJMMRHCBVAZEQUMNVTGMPOYEEGO
GRLWD
                                ↓
DWLRGOGEEYOPMGTVNMUQEZAVBCHRMMJCWNQYPEJDHJFUSEUMZLHFNBTMOLO
JQDMABJLBXQNZLKUWDLSMJWELDCTIOBUQVWZWYABFSPFWUIXJXLQEDOPNFI
CGFCF
```

Take the register we found at position 123, and reverse the shift key to UJALN. Use them to *encipher* the text with H1. The output is the plaintext portion before the crib's position but reversed. For our ciphertext, we find that chunk to be (after un-reversing it)

```
NEBRASKAHASNONATURALCAVESORCAVERNSHOWEVERTHEREARETWOEXTENSI
VEMANMADELIMESTONECAVESNEARLOUISVILLEINCASSANDSARPYCOUNTIES
WHERE
```

We need to also output the register at the end of the enciphering process and keep track of the rotations of the shift key as we go along. For our example, the register and shift key at position 0 are

register:	AKQHRXNFWULBESIDZPGOVMCYTJ
shift key:	AJUNL

We can use the keys above to decrypt the full ciphertext, and we find that it is from "An Annotated Checklist of Nebraskan Bats" by J. Knox Jones and Olin L. Webb:

> Nebraska has no natural caves or caverns; however, there are two extensive man-made limestone caves near Louisville, in Cass and Sarpy counties, where four kinds of bats have been found. Two of these are here reported as new to Nebraska. The quarries, one on either side of the Platte River Valley, are in a horizontal stratum of limestone forty feet in thickness, and are of the room and pillar type; that is to say, the stone has been quarried away leaving rooms having supporting pillars approximately every fifty feet.

We should not be satisfied, however, until we have also recovered the keywords. To do so, we take the register at position 0 as look at the twelve equivalent registers. The equivalent registers are found by taking the one we have and taking every 3rd letter with wrap-around, or every 5th, or 7th, 9th, 11th, 15th, 17th, 19th, 21st, 23rd, or 25th. You should by now recognize these as the elements of $\mathbb{Z}_{26}^{*}$. Or you should be thinking that these sound like affine ciphers. The twelve equivalent register contents are

```
AKQHRXNFWULBESIDZPGOVMCYTJ
AHNUEDGMTKRFLSZOCJQXWBIPVY
AXLDVJRUIOTHWSGYQFEPCKNBZM
AFIMQUZYRBGJNSVKWDCHLPTXEO
AUGKLOQBVHEMRSCXIYNDTFZJWP
ABCFGHIJLMNPQSTUVXZKEYWORD
ADROWYEKZXVUTSQPNMLJIHGFCB
APWJZFTDNYIXCSRMEHVBQOLKGU
AOEXTPLHCDWKVSNJGBRYZUQMIF
AMZBNKCPEFQYGSWHTOIURJVDLX
AYVPIBWXQJCOZSLFRKTMGDEUNH
AJTYCMVOGPZDISEBLUWFNXRHQK
```

The obvious choice is there in the middle, for every 11th letter, and we can harmlessly rotate it so that the keyword is at the beginning:

```
KEYWORDABCFGHIJLMNPQSTUVXZ
```

To find the shift key, we use this register and the first five letters of the plaintext and ciphertext in the same way that we did at position 123. First come N and F, which are nineteen steps apart, to give shift-key letter S. We have to swap those letters in the register.

```
NEBRA
FCFGC

KEYWORDABCNGHIJLMFPQSTUVXZ
```

Next are E and C, which are eight steps apart, for a key letter of H. Swap the letters in the register.

```
NEBRA
FCFGC

KCYWORDABENGHIJLMFPQSTUVXZ
```

Third are B and F, which are nine steps apart, for key letter I. Swap B and F in the register.

```
NEBRA
FCFGC

KCYWORDAFENGHIJLMBPQSTUVXZ
```

Then R and G are six apart for key letter F.

```
NEBRA
FCFGC

KCYWOGDAFENRHIJLMBPQSTUVXZ
```

The last pair are A and C, which are twenty steps apart in the current register, for key letter T. So the shift key is SHIFT. And we are finished.

There is another way to find the new shift key. Recall that we found shift key AJUNL for the unresolved register, and that to resolve the degeneracy we took every 11th letter. The inverse of 11 modulo 26 is 19. If we multiply the shifts by 19 modulo 26, they go from 1, 10, 21, 14, 12 to 19, 8, 9, 6, 20, and so the new shift key is SHIFT.

Reading and references

Thomas Kaeding, “Breaking the Hutton 2 challenge,” Cryptology ePrint Archive, report 2023/1113.

Programming tasks

1. Automate the entire process.

Exercises

1. Break this ciphertext with the given crib.

```
OAWMUCDGUDSOKRJDJNJBNWWGUWQRIDRQPGFZSHTJODSKEIMSYELJKJYXMYA
SJLGEWVGMXRGDWERRIYWONPUYMGXTFPXBBINKFPZOFLGWQKEMWIYDWJAOAC
JFGIQSJGLTGXUESFHRQBJYYIYQBCKFNWUPGKBAZQMAWFSSLYZRMSYCJMTAC
QKYNSSGPUMSAYAYIDKMOZRAYHFBVXBLOGYDYMRMQATSUVNITVZDQDSANQSO
WHLHGFPOSZGUBEGXBPDNCMKIRGOBGBUOCTRMFUCHGWUIVOUPUHMXZJAZCWN
JWIANMGSVPIQZML
```

crib: MOSTOFYOUAREORDINARYANDHARMLESSIAMNOTINTERESTEDINYOU

Unit 156
Hutton 2 cipher

There is a variation of the Hutton cipher, known as *Hutton cipher 2*, in which the shift is increased by the value of the first letter of the alphabet key at any given time. For reference, when enciphered with Hutton 2 and the keywords `HUTTON` and `CIPHER`, the message

```
THIS MESSAGE WAS ENCRYPTED WITH HUTTON CIPHER
```

becomes

```
JPQIGHMFJOOZXBEJVUMUKJKNYDWEFYSABVKR
```

Reading and references

"Hutton Cipher v2," huttoncipher2.netlify.com

Girkov Arpa, "How It Works," hutton-cipher.netlify.app/howto.html

Programming tasks

1. Implement the cipher.

2. Implement a dictionary attack.

Exercises

1. Encipher this text with Hutton 2 and keywords `OCEAN` (mixing) and `LAND` (shifts).

   > By the present theory, the earth on which we dwell is represented as having been formed originally in horizontal strata at the bottom of the ocean; hence it should appear, that the land, in having been raised from the sea, and thus placed upon a higher level, had been of a different shape and condition from that in which we find it at the present time.
   >
   > (from *Theory of the Earth*, Volume 2 by James Hutton)

2. Decipher this ciphertext with Hutton 2 and keywords `RIVER` (mixing) and `SEA` (shifts).

   ```
   KDXLRZUWOVTDBCZBXMFVIGBWTXMKIDVEOFYBAJSJBUOBQRALSLQTGNJUGFJ
   ZXCIOZUEVOKOKBAMMORVLQREGSGVMGVBZPDCDNRHOSPGEZXOCBCETWNDJLY
   ```

```
MRRIVUQGTLRZFBIYBLHBMGIVPDHASDMJLTPLUJCUBSBBUPQJEBRBDBTVGNW
ELCHCMIWBFUPSSNKBPPXUH
```

3. Perform a dictionary attack on this ciphertext. It was encrypted with Hutton 2. Both keywords are five-letter English words.

```
SABOOKONJEMXGEUWGFUDDZOOFCTMJUVATUYEQYNCNFLBHZSGZHJUTKWGCCY
QLYETQCCSUHMRYHQKJVNPXILEHYNHJEKQGKHEXTZSSGINBQTAJKOGEPHOMK
YATHZHPCCIBXQLOSVNVHLDMYBRZQFLMAWIQLZJGYHLJKTSMZVMTZXYOOZEP
MNJBZTRZMRSSMICIDAGNDSNYNFIRFYXXRHYVFMINDSVVROMLNVLTMEIFOXL
USMZJEUIXHMURCMOHTVOHYNNSDLYLUUPALDFRGDQUUYPVVQRXHRUEWWFSCL
LTWMLBBPXLNJEVUGOQFZVHAHVAPKUHIVGXJZVKLYTBJTGIIQEDRXUGKUUVS
PFXTJCYVPCKOTTZSBKAKGWCEQGHGIJPOYELOMJSOPOSNBDFDRSJWKTEDUCF
MIE
```

Unit 157
Attacking Hutton 2 with cribs

Hutton 2 is H1 with an added feature: the value of the first letter of the register is added to each shift. The first letter changes often as a text is enciphered, so the addition to the shifts changes also. This complicates the crib-based attack we saw in Unit 155. Let's discuss its steps in light of the new feature.

1. Find the correct placement of the crib. Whenever the shift key has a letter that is the complement of the first letter of the register, there can be fixed points (letters that are enciphered to themselves) in the ciphertext. By this we mean that if the first letter of the register and the shift key have letters whose values add to 26, then a letter is enciphered to itself. Because the first letter of the register changes less often, typically, than we cycle through the letters of the shift key, fixed points can occur periodically. So when we try to place a crib against the ciphertext, we want either no fixed points or fixed points that are periodic. The period is the length of the shift key. Furthermore, fixed points can turn off or on in the middle of the crib. There may be many placements of the crib that look satisfactory, and so we may need to do a lot of checking. Steps 4 and 5 are where failure is most likely to occur with an incorrect placement.
2. Remove the effect of the swapping of letters that occurs after each letter is enciphered with H2. This step is the same as it was for H1. What remains is a plaintext-ciphertext pair for a quagmire 3 cipher.
3. If step 1 involved fixed points, then we know the length of the shift key. Otherwise, guess the length. We also have to guess the first letter of the register, or loop over the twenty-six letters until we find the right one. If we guess either incorrectly, then steps 4 and 5 below will likely fail and we will guess again.
4. Use the results of step 2, the length of the shift key, and the first letter of the register to partially fill a key table for a Q3-like cipher. We need to remember that the value of the register's first letter is added to each shift. We also need to keep track of that letter and when it changes. This we can do because we know the plaintext and ciphertext (where the crib is) and so we know all the swaps that take place. The key table that we build will have twenty-six rows, corresponding to all possible shifts. Those shifts are no longer periodic as they were for the proper Q3. Because we need to build a bigger table than for the H1, we need a longer crib.
5. Use techniques that we saw in Part III to expand our knowledge of the Q3 keys until we can determine the contents of the register at the start of the crib. The techniques of Units 74 and 75 help us to fill in the complete tableau of the Q3, and Unit 76 helps us to recover the Q3 key, which is the content of the H2 register. There are twelve possibilities for the result. We must make sure that we rotate each until the first letter is the one that we believe it to be.
6. Find the shift key by comparing the first few letters of the segment of the ciphertext to the crib, while remembering to subtract the first letter of the register at each position. Try this for all twelve candidates for the register's contents. Choose the one that gives the correct decryption of

that part of the ciphertext that aligns with the crib. Remember that at this position of the ciphertext we may or may not be using the first letter of the shift key; which letter of the key we use can be determined by taking the zero-based position modulo the length of the shift key.

7. Take the register and shift key at the start of the crib and run the H2 cipher backwards to find the register and shift key at the start of the ciphertext. At each point, we have to be aware that there are three possibilities:
   a. The first letter of the register does not changed, so we use the expected shift at that point and the ciphertext letter to find the plaintext letter. This occurs most often.
   b. The plaintext letter is swapped into the first place in the register at this point of the original encipherment. So whatever letter that we have in the first place in the register is the plaintext letter that we take. The ciphertext letter at that position of the text then becomes the first letter of the register. From the register's first letter and the shift-key letter we can calculate the shift; if the shift is inconsistent with with what we know about the plaintext and ciphertext letters and the register, then we must reject option (b).
   c. The ciphertext letter is swapped into the first place of the register at this point of the original encipherment. We know whether this is possible because we know the identity of the ciphertext letter. If the ciphertext letter matches the initial letter of the register at the point after this position in the text, then we are almost certain that option (c) applies; there is still the possibility that the overall shift is 26 and we are at a fixed point, so that option (a) is correct, however. But we must then try all possible plaintext letters until we find one that is consistent and does not give nonsense when we decrypt further.
8. Use the register and shift key from steps 6 and 7 to decrypt the entire ciphertext.

To clarify further, we work an example. Consider this ciphertext:

```
NUICSJDQYRXRYVFMSUZVPECKCQYORHGMUCPMKFVIOGDXGWYGMPFYZIJYIUEUBYGURHJMH
HTZAVDDIGSMYDDUUUQQAEQJEYFFCOBSPROXAYGOKECTDMLYCTEVIKWCMGNCMLJDKZRCXP
SKRVHPZZLAAXDVJHDTMUOTEVBGUDVLMQXHIAJBOAUMSDMACWJZIRQYENXKNQQPFRBLQZP
QFBJNTYZUHLCNKRZKWXLXLWMEGKZNUOSIVZGZZSQFEGQZJXOUHBSAEHXJLIFKPVCBYCZE
GIMRTRPDVMZHRJAKQJUYVJBISKJONIDOCBDFGLMJCXUXKOQMQWVYPAQWIODWMCLYMXSRQ
VCPUJRMIJQREJFFVNDSVDMWCDLLAKWFGBNTGJBZQHAQNVHKGUCTWRUQBVWWEONOWAZCIS
CIBJYUNCGSCHOVMKJLRAOXOQSFMTTOYGIYKYTOMQKUUZXENEWZKWXUAZNFUYURFSSTUHB
LNDZTERENDEENHONGHQNVTMWPZHGBOULVFGKXVZVTNWUTOPOBBBCCPBQHXZHZBIHMA
```

We will attack it with this crib of fifty-eight letters:

```
VARIETYOFANIMALANDVEGETABLELIFEONTHESURFACEOFOUREARTHINTHE
```

The first step is to place the crib in its correct position against the ciphertext. We begin at the start of the ciphertext, where we find two fixed points. However, they show no repeating pattern:

```
          1111111111222222222233333333334444444444555555555
012345678901234567890123456789012345678901234567890123456789012345678
VARIETYOFANIMALANDVEGETABLELIFEONTHESURFACEOFOUREARTHINTHE
NUICSJDQYRXRYVFMSUZVPECKCQYORHGMUCPMKFVIOGDXGWYGMPFYZIJYIU
```

We need to try all positions, and at position 65 (counting from zero), we see a pattern:

```
                                        111111111111111111111
666667777777778888888888999999999900000000001111111111222
567890123456789012345678901234567890123456789012345678901 2
VARIETYOFANIMALANDVEGETABLELIFEONTHESURFACEOFOUREARTHINTHE
HJMHHTZAVDDIGSMYDDUUUQQAEQJEYFFCOBSPROXAYGOKECTDMLYCTEVIKW
```

We now have good reason to believe that we have the correct placement, and that the period (the length of the shift key) is six.

Next is to remove the effect of the swapping of letters that took place when the text was enciphered. We did an example of this in Unit 155, so for the sake of brevity, we simply give the results:

```
                                        111111111111111111111
666667777777778888888888999999999900000000001111111111222
567890123456789012345678901234567890123456789012345678901 2
VARIETYOFONVRDLSOOFIRFTZBGBTVHTJNFEHERETTJPPZKNXZVONDSCDKS
HJMVITZJHDOVGSGZNOUFIQFZQTAASHHCCQDPMNXZVIRKRIQOLATKKLULPW
```

Each plaintext-ciphertext letter pair in the above is part of a permutation in the underlying quagmire 3 of the H2 cipher.

To get at the keys of the Q3, we need to know (or guess) the initial letter of the H2's register. If we guess that the letter is A, we can track how it changes by looking at the swaps that occur in the original crip and corresponding ciphertext:

```
position:                                       111111111111111111111
                666667777777778888888888999999999900000000001111111111222
                567890123456789012345678901234567890123456789012345678901 2
shift index:    501234501234501234501234501234501234501234501234501234501 2
plain:          VARIETYOFANIMALANDVEGETABLELIFEONTHESURFACEOFOUREARTHINTHE
cipher:         HJMHHTZAVDDIGSMYDDUUUQQAEQJEYFFCOBSPROXAYGOKECTDMLYCTEVIKW
register[0]:    AAJJJJJJJJJJJJJJJJJJJJJJJJELLLLLLLLLLLLLLLLLLLLLAAAAAAAA
shift added:      111111111111111111111111 111111111111111111111
                110000000000000000000000005222222222222222222222211111111
unswapped P:    VARIETYOFONVRDLSOOFIRFTZBGBTVHTJNFEHERETTJPPZKNXZVONDSCDKS
unswapped C:    HJMVITZJHDOVGSGZNOUFIQFZQTAASHHCCQDPMNXZVIRKRIQOLATKKLULPW
```

For shift-key letters 0, 1, 3, 4, and 5, there are four possible additions to the shift: 1, 10, 12 (all numbers have been written vertically in the above, so that everything lines up). For shift-key letter 2, the addition of 5 is also possible. We can now assign letters from the unswapped ciphertext to the keys of the Q3: V→H for shift index 5 and added shift 1, A→J for shift index 0 and added shift 1, R→M for 1 and 10, etc. However, we have a problem when we place I under R in the row for shift index 1 and additional shift 10 at text position 85, since we have already placed an M there from text position 67.Because of this inconsistency, we reject A as the first letter of the register.

shift index	shift added	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	1	J.....
0	10	...S....F.....J.....
0	12	.....
1	1	.....
1	10	.....H.....G.....M.....
1	12	.....
2	1	.....
2	10	.....V.....D...Z.....
2	12	.....
3	1	.....
3	10	....I.....ON.....
3	5	.....
3	12	.....
4	1	.....
4	10	.....O....T.V....
4	12	.....
5	1	.....H....
5	10	.....U.....G.....Z.
5	12	.....

We can play this game for each guess of the initial letter of the register, and for E we find consistent results.

```
position:                                                  1111111111111111111111
                 666667777777778888888888999999999900000000001111111111222
                 567890123456789012345678901234567890123456789012345678901 2
shift index:     501234501234501234501234501234501234501234501234501234501 2
plain:           VARIETYOFANIMALANDVEGETABLELIFEONTHESURFACEOFOUREARTHINTHE
cipher:          HJMHHTZAVDDIGSMYDDUUUQQAEQJEYFFCOBSPROXAYGOKECTDMLYCTEVIKW
register[0]:     EEEEEHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHSSRRXXXXXXXXXXXXXXXXXX
shift added:                                               1111222222222222222222
                 555558888888888888888888888888888899884444444444444444444
unswapped P:     VARIETYOFONVRDLSOOFIRFTZBGBTVHTJNFEHERETTJPPZKNXZVONDSCDKS
unswapped C:     HJMVITZJHDOVGSGZNOUFIQFZQTAASHHCCQDPMNXZVIRKRIQOLATKKLULPW
```

shift index	shift added	plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ
0	5	J.....
0	8	...S..T.FC....J.....
0	18	.....N.....
0	24	...L.....K.....A....
1	5	.....M.....
1	8	.A...H.....G.C...I.....
1	18	....X.....
1	24	.....P...T.....R
2	5	.....V.....
2	8	.....Q.....D...ZA.....
2	24	.....I..K....WZ.....
3	5	....I.....
3	8	....D.....ON....F.S....
3	24	...K.....Q.....V.....
4	8	.....H.....O....T.V...Z
4	19	.....P.....
4	24	.....I.....L....O..
5	5	.....H....
5	8	.Q...U.....G.H....Z.
5	19	....M.....
5	24	..U.....R.....L

Applying the techniques of Units 74 and 75 to this partial key table yields the following complete tableau for the underlying Q3 cipher:

plaintext alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ	order
ABCDEFGHIJKLMNOPQRSTUVWXYZ	1
BJNHUALFRVMCSDEYKWPQTZOIGX	26
CNSYXLKGBDOMEPZTWJURIHVAQF	26
DHYLRNSCVFUPTGIKEZQOWAXJMB	26
EUXRLOVWSTHZFIGBDPANCQYMJK	13
FALNOHYDXBQGKCWSTIMUEJRZPV	26
GLKSVYTPFCRQWMJEIAOXZNBHUD	26
HFGCWDPNZATYQLRMUXKEOBIVSJ	13
IRBVSXFZTWLACJMDGQNYPOKUHE	13
JVDFTBCAWZSNPHUGMOYKQXERLI	13
KMOUHQRTLSJWVEDXBCZAFPNGIY	13
LCMPZGQYANWKOSVURBEIXDJFTH	13
MSETFKWQCPVOZUHIJNXBAYDLRG	26
NDPGICMLJHESUYXQOVTWRFZBKA	13
OEZIGWJRMUDVHXYANSFCLTPKBQ	26
PYTKBSEMDGXUIQAWZHRVJLFNOC	26

QKWEDTIUGMBRJONZALVFHSCYXP	2
RWJZPIAXQOCBNVSHLKDGYEMTFU	26
SPUQAMOKNYZEXTFRVDIJBGHCWL	13
TQRONUXEYKAIBWCVFGJHDMLPZS	13
UTIWCEZOPQFXARLJHYBDNKGSVM	26
VZHAQJNBOXPDYFTLSEGMKIUWCR	26
WOVXYRBIKENJDZPFCMHLGUSQAT	13
XIAJMZHVURGFLBKNYTCPSWQEDO	26
YGQMJPUSHLITRKBOXFWZVCADEN	13
ZXFBKVDJEIYHGAQCPULSMRTONW	13

If we choose any permutation with order 26 and apply the method of Unit 76 we obtain twelve possibilities for the register contents, rotated so that E is the first letter:

EMLFZOKGHVWQYDJRTPNBIUSCAX
EFKVYRNUAMZGWDTBSXLOHQJPIC
EOWRIXZVJBAFHDNCLGYPSMKQTU
EGJULVTCZQNXKDIMHRSFWPAOYB
EVNMWBLQIFYUZDSOJCKRAGTXHP
EQSGNFJXWUKPLDAVIOTMYCHBZR
ERZBHCYMTOIVADLPKUWXJFNGSQ
EPHXTGARKCJOSDZUYFIQLBWMNV
EBYOAPWFSRHMIDKXNQZCTVLUJG
EUTQKMSPYGLCNDHFABJVZXIRWO
ECIPJQHOLXSBTDWGZMAUNRYVKF
EXACSUIBNPTRJDYQWVHGKOZFLM

Next we recover the shift key at the point where the crib begins. We believe we know that the length of the shift key is six, so we examine the first six letters of the crib and their corresponding ciphertext letters. For each pair of letters, we can find the shift from the contents of the register. Then, if we subtract the value of the initial letter of the register modulo 26, we can determine the identity of the shift-key letter. Remember that the register changes when each letter is enciphered, so that we have to always use the state of the register as it evolves, and that includes the possibility that its initial letter may change. For our example, we need to encipher VARIET to HJMHHT. Using the first candidate above for the contents of the register, we need a shift of 25 to get from V to H. The first letter of the register is E, so the shift-key letter must have value 20, which gives T. Then V and H are swapped in the register before moving on to the next letter. We know from above that on the sixth letter, the first letter of the register has become H, and we must adjust accordingly. The shift key we find for this candidate register is TKGJOR. However, if we decrypt the ciphertext beginning where the crib is placed with these shift key and register, the plaintext is VARIETLUBAJIOFHH.... Clearly, this is the wrong candidate register. For the twelve candidates, here are the shift keys that we find and the first twelve letters of the decryption at that point:

EMLFZOKGHVWQYDJRTPNBIUSCAX	TKGJOR	VARIETLUBAJI
EFKVYRNUAMZGWDTBSXLOHQJPIC	LIYZSR	VARIETSMDWGI
EOWRIXZVJBAFHDNCLGYPSMKQTU	ZSMXYR	VARIETTIGAHG
EGJULVTCZQNXKDIMHRSFWPAOYB	FASLIR	VARIETWFRAFI
EVNMWBLQIFYUZDSOJCKRAGTXHP	RQENCR	VARIETZCUALI
EQSGNFJXWUKPLDAVIOTMYCHBZR	BMOTKR	VARIETJPMATI
ERZBHCYMTOIVADLPKUWXJFNGSQ	NCAVER	VARIETYOFANI
EPHXTGARKCJOSDZUYFIQLBWMNV	XYKBMR	VARIETKXCAZI
EBYOAPWFSRHMIDKXNQZCTVLUJG	JOWDGR	VARIETHMDWGO
EUTQKMSPYGLCNDHFABJVZXIRWO	PWCRQR	VARIETEDQDII
ECIPJQHOLXSBTDWGZMAUNRYVKF	DGQPWR	VARIETACOGSI
EXACSUIBNPTRJDYQWVHGKOZFLM	VEIFAR	VARIETVRUAWI

The only match to the crib is when we have

register: ERZBHCYMTOIVADLPKUWXJFNGSQ
shift key: NCAVER

These are the keys at the point where the crib begins, i.e., position 65 in the text. Remember that our crib starts at index 5 in the crib (65 mod 6), so the true shift key is CAVERN.

Next we must run the cipher backwards from position 65 to the beginning of the ciphertext, and thereby obtain the register contents when the original encipherment of the text began. At position 64, the ciphertext letter is R and the shift-key letter is also R. The register after this position is given above. Under option (a) of step 7, we assume that the first letter of the register does not change. In that case, the shift used to encipher this position is calculated to be R+E=19+5=23. The plaintext letter is therefore S (23 steps after the ciphertext letter, because they have been swapped in the register that we have). Before position 64, the register would have been (swap R and S)

ESZBHCYMTOIVADLPKUWXJFNGRQ

Taking this register and shift key RNCAVE, the ciphertext going forward starting at position 64 can be deciphered to SVARIETYOFANIMAL..., which is correct. Therefore, option (a) is consistent with what we already know. Under option (b), we assume that the plaintext letter was swapped into the first letter of the register when it was enciphered. We know from the register at position 65 that this letter is E. The ciphertext letter is R, and it must have previously occupied the register's first position, in order to be swapped with E. Therefore the shift at position 64 is R+R=18+18=36=10 (mod 26). But the register must have been (swap R and E)

RSZBHCYMTOIVADLPKUWXJFNGEQ

and therefore the shift must have been 2 (the distance from E to R). We have an inconsistency and must reject option (b). Under option (c), we assume that the ciphertext letter is swapped into the first place of the register. We know that that letter is R, but also is E. Because of this new inconsistency, we reject option (c). Option (a) is the only consistent one, so we conclude that at position 64 the plaintext letter is S and before it was enciphered the register was

ESZBHCYMTOIVADLPKUWXJFNGRQ

At position 63, things go nearly the same, option (a) is the only consistent one, and we again get plaintext letter S. Before position 63, the register contains

EUZBHCYMTOIVADLPKSWXJFNGRQ

At position 62 something new happens. The ciphertext letter is G and the shift-key letter is V. The contents of the register after position 62 is enciphered is given immediately above. Under option (a), the shift is V+E=22+5=27=1 (mod 26). Therefore the plaintext letter would be R (one step after G in the register after encipherment). Before encipherment, the register would have been

EUZBHCYMTOIVADLPKSWXJFNRGQ

If we use it and shift key VERNCA to decrypt the ciphertext from that point forward, we get RSSVARIETYOFANIMAL..., which is consistent with what we know, but not likely to be correct, since we cannot think of a word that ends in -RSS. Under option (b), the plaintext letter is swapped into the first position of the register, and we know that to be E. Before encipherment, the register would have been

GUZBHCYMTOIVADLPKSWXJFNERQ

The value of the shift must have been 3. Is it consistent? Adding the shift-key letter to the register's initial letter gives V+G=22+7=29=3 (mod 26). It is consistent, and we cannot rule out option (b). Furthermore, -ESS is a more likely ending to a word. Using the register contents above and shifte key VERNCA, the plaintext comes out as we expect: ESSVARIETYOFANIMAL.... Under option (c), we would need G to equal E, but it does not. We conclude with confidence (not absolute confidence) that the plaintext letter is E and that option (b) is correct at position 62.

Things continue in this way until we reach position 47. The shift-key letter is N, the ciphertext letter is G and the register contents are

GZNDJCLSYUBVATEOKMWXHRIPFQ

This is an excellent opportunity for trying option (c) of step 7. The plaintext that we have reconstructed so far looks like

...SORINTHEBOUNDLESSVARIETYOFANIMAL...

We don't know the identity of the plaintext letter at position 47, so we must explore the possibilities. Suppose that it is A. Then the register before position 47 must have been (swap A and G)

AZNDJCLSYUBVGTEOKMWXHRIPFQ

From this we can see that the shift we need is 12. However, N+A=14+1=15. We have a contradiction and must reject A. We repeat this analysis for each choice of plaintext letter:

plaintext letter	needed shift	calculated shift	plaintext letter	needed shift	calculated shift
A	12	15	N	2	2
B	10	16	O	15	3
C	5	17	P	23	4
D	3	18	Q	25	5
E	14	19	R	21	6
F	24	20	S	7	7
G	26 = 0	21	T	13	8
H	20	22	U	9	9
I	22	23	V	11	10
J	4	24	W	18	11
K	16	25	X	19	12
L	6	26 = 0	Y	8	13
M	17	1	Z	1	14

There are three consistent possibilities: N, S, U. What happens if we choose one and continue on? If we go on for five more letters, we get

```
N:    HEAVENSORINTHEBOUNDLESSVARIETYOFANIMAL...
S:    QNLKXSSORINTHEBOUNDLESSVARIETYOFANIMAL...
U:    TZUWRUSORINTHEBOUNDLESSVARIETYOFANIMAL...
```

The only sensible choice is N.

We continue until we are at the point before the initial encipherment began. There, the register contains

```
WONDERFULABCGHIJKMPQSTVXYZ
```

The keywords are therefore WONDERFUL CAVERN, and we can decrypt the full text to find a passage from *The Subterranean World* by George Hartwig:

> Nature displays her wonders not only in the starry heavens or in the boundless variety of animal and vegetable life on the surface of our earth. In the dark regions underground she likewise shows us much that is remarkable or beautiful, or carries on gigantic operations, which are sometimes beneficent and sometimes disastrous to mankind.
>
> There lie concealed the mysterious laboratories of fire, which reveal to us their existence in earthquakes and volcanic explosions. There, in successive strata, repose the remains of extinct animals and plants. There many a wonderful cavern may be seen, with its fantastic stalactites, its rushing waters, and its noble halls.

Reading and references

Thomas Kaeding, “Breaking the Hutton 2 challenge,” Cryptology ePrint Archive, report 2023/1113.

Programming tasks

1. Automate the entire process.

Exercises

1. This ciphertext is from a challenge at MysteryTwister and was prepared by Hutton himself. The first seventy-eight letters of the plaintext are

   ```
   FEWPERSONSCANBEMADETOBELIEVETHATITISNOT
   QUITEANEASYTHINGTOINVENTAMETHODOFSECRET
   ```

   Find both keywords and decrypt the text.

   ```
   CMVHYRQUDOUAVGYGPDGNBSKLHCZBMJIREXMJPFZYEGIKUDUIZYYZNAKTPYCKZGEU
   WMOREOXXACDPNPFZYAXBDLBCONMMGDBTGNEJRLJVWJARXZMQQLPLBPNWKDZBBXHK
   ITDUTVQQFVLGQWWAIGSSXSEEUTRCLCQBPVGCKOHDNTDQBLZEOPQPMHNJGRKIKASC
   UJJJQONODULCSLVQHWNHZOYRCZFUOHWWBRQAIRCTEIGXKKHIBVGZOEPHBIFRJVVU
   PSFQOTRLBQRXEPUIJREZKIWRYDVVWDOHRXZKACPOHJGEOFQQXENXHCVXRVSFJXPD
   WQRPDPXPZADSJYKPALWOTCVBPPENMIONLLOCJZQTUWJQBDGQQPTVBBTEZFKSVRCB
   WMPLIFVAKEDPGBUJYCNZMXTTYGBPJDKKRRSMLOOQHPPUOMBEOSDQYLPGNECTSVVN
   XXZKHFBJMJTUGRIIXLXJOGDVMFNKTIWVFVOLQVEIZHCFCUPRUITDOWEAHQTCIMNG
   KVLUFSZHGIYAIJQFFJLVQZIUSPMIZAJTIQKZYYQSHAHDPUEHOSEORWZIJEDGFXLA
   MUBMFPIPZROMLXJHXNJSWRNJKUIIKEEVOOSJTEZPMYOYQQDQVRMMXMCPNUFIXPEY
   UWJQAAYZMNHXMRFBWCTTLPTRUTLWLMHLKKQURZZZTYDFZXCDLIVSWTTOZINUJMSN
   XQDPYYNVCZRSNDUSZROXYUYFSGAFCDUSNDUOSRU
   ```

   Links to the challenge on MysteryTwister:
   https://mysterytwister.org/media/challenges/pdf/mtc3-hutton-01-hutton-02-en.pdf
   https://mysterytwister.org/media/challenges/add/mtc3-hutton-01-hutton-02-add.zip

2. Break this ciphertext with the given crib.

   ```
   AMEDLEYOFSMALLROCKSANDFRAGMENTSOFSTONEALONGONESIDEALEDGE

   DZJNMJUHDXOXFKKXQTKYPAXMTTCUQMNIWNEPUAULPQBOWZICPFOPVZRDGKGXVGDA
   OTLAAVATPXKOZKQIBRTTLCUHQSGINMOQOCFLMZGEEWUMYQBKVTCMSPGMLVCPLXSO
   TORKDPIHIBLWMDQGYDYMYSCMCAFJAMSDYNJZKWAEGHCOCBGNEJUBMAEAUCAFPCAJ
   JYWADSAQWGNFIQPGRSUPBWFXCRPJSZSZRECEVDMFBNWIOUSOPAQVIEWVOOEDFBVN
   PXPCKTIJVYNTHATJPBPBQXDJZKGBYHDQFGEJAKQTKDALLYWFVHDYGLQQXRDMXLFS
   PRNGPODVIGIJGHYLAHIRPXJWIRDPYXTIRAVZAGJCKMLYKKLIHWHOGPNBDLGUYWUG
   NJMEYRVFCPRMWOPURKHYOOAQCOQQWUQU
   ```

3. Break this ciphertext with the given crib.

   ```
   WECAMETOTHEKENTUCKYCLIFFSSOCALLEDFROMTHEFANCIEDRESEMBLANCETOTHEC
   LIFFSONTHEKENTUCKYRIVER

   UWSJDDTTBEDRRNCFYTBUCBLMULXNCPSLJRCKLBTEGZXVLNAJOYGJTPFUSCVYLEJB
   SRFVTZSDMZNLBNDHYJTVVUYMOFDLNJIYRLLJKWRHWHIEHUDPXDRKMJFQCVCLCBVQ
   ```

PQCYWHMLKVTULVQQVJLESCCGCTJMDCUALQUVHLKRITHWZCNTCYGOMHETTZGMAVFT
BRUHQTKAAOZHFGHKXJBNTDZQNLZVRCEHQYTUTOIDGPXSUOWMDJTNDWQMTSJZCODL
TMISSSVQIMLDRKMOJOIDBWGIFICMRTZJSYTVFCDQFLZINKLMRNXVYOMXMDXNZZYG
VTQTYGYDKASPSRTUTRXRURCCDDDDWFDSTMKILNKZZJYHPJEUQENUINBVWAARFUZR
HJQTZUUBNKHVOUPTWUOOPGRPIDQCZWIQYTVYJTZSZJYROGEXYDLKBDVBMAOXTSTI
QXUVTRNVTHIMDUFSUTKXPVIJWPMNWCXWBDFISRXVMKXLBHRRTIKDTUKQNKSRDUXB
JSUJSKXHSUWPUTVUELWWYWFVSKZRSNYAMIWRTDGXDABYMJJCACFTMBRTNQWFQQUZ
GLJZIVKCPVLFLAFWDTWRRLVZNJDQFODEJKFLJIXKOUFPLNEQJJSCNCBAUJIQZLSK
YNMVBVCUKXJIJMPUZTXZCOHJSJEWWAUWTQNOQUUJHRLGRVZJILCDSYBKBOJMVGTB
IXQDVXBQQRNFTCPTDMSQYMBQTG

Unit 158
Scrabble cipher

The *Scrabble cipher* is an invention of this author in response to the Hutton cipher. It is an attempt to create a cipher with a similar level of security but with simpler rules. Scrabble™ is a trademark of some large corporation. They seem like nice people, and we are sure they would be happy to let us use the name, but we haven't had time to ask them yet.

The basis of the Scrabble cipher is the monoalphabetic substitution. When each letter of the plaintext is enciphered, it will be done with an alphabet key. That key, however, changes with each letter that we encipher.

The first thing to do is choose the initial key. There are four choices for how this is done.

1. Use a keyword to generate a mixed alphabet. For example, if our keyword is `KEYWORD`, then we have

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

2. Shuffle the alphabet with a keyword. Here is our example with the keyword `KEYWORD`. We begin with the standard alphabet:

   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

   For each letter in the keyword, we do a shuffling step that swaps the letters before and after our key letter. If there are no letters in one of those groups, we just do the swap anyway, with one set being empty. So, we start with the standard alphabet and swap around 'K':

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

   Then we take the result and apply a swap around the next key letter, 'E':

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

   We continue and use the remaining letters of `KEYWORD` to get this keyed alphabet:

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

3. Generate an alphabet from a keyword as in option 1 above, then shuffle it with another keyword as in option 2.

4. Use a randomly permuted alphabet.

The encryption process employs the same shuffling step as we used above in creating type 2 keys. For each character of the plaintext, a substitution is made, and then a shuffling. Each shuffling is swapping the block of letters before the plaintext character with the block of letters after it.

As an example, suppose we want to use the alphabet key from keying option 2 above and encrypt the message “send help”. First, find the substitution for ‘s’:

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

We get an ‘O’. Next, shuffle the alphabet around the letter ‘S’:

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

This new alphabet will be used to encrypt the next letter of the message.

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

We get ‘X.’ Another shuffling, this time around the plaintext letter ‘e’:

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

This continues until the entire message is encrypted.

```
s e n d h e l p
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
O X W D Z M S A
```

Reading and references

Challenges on RingZer0:

ringzer0ctf.com/challenges/300
ringzer0ctf.com/challenges/301

Programming tasks

1. Implement the four keying options.

2. Implement the cipher.

3. Implement a dictionary attack for the cases in which the key is generated from one or two keywords.

Exercises

1. Encipher this text ...

   a. ... using keying option 1 with keyword `QUACK`.

   b. ... using keying option 2 with keyword `MAVEN`.

   c. ... using keying option 3 with keywords `PLAY` and `HARD`.

   Scrabble™ is a crossword game for humans, but did you know that there are computer programs that can play the game? One such program is Maven by Brian Sheppard. Maven was incorporated into the video-game version of Scrabble™. Quackle is an open-source program that also comes with a graphical interface.

2. Decipher this ciphertext with keying option 1 with keyword `CROSSWORD`.

   ```
   JNNQZVMEEAOLRZNQJXWULPCSSURGDXEEVPMKSYZNBKAQIVUXJVWANQCRSBZ
   EQJWMYCPBAPDOWGNXKQJPPJFZGGHRNUTAXUKHHBHPHIQHFE
   ```

3. Decipher this ciphertext with keying option 2 with keyword `POINTS`.

   ```
   LVMXWUTRDMQJQQNGEIBXHIRHUPOMOZDWDDCHKKJFWTNEMTCTOFGDTTGORQF
   LCGJMWXKRGFECSLKJPALVXJJKVIKEDTJTIPINNLEAWDFVBVMCLWKNDXDMAJ
   LFCOBXLNWMVFJEWRTLFCBFWKDDQTUDERPVUJPSHMD
   ```

4. Decipher this ciphertext with keying option 3 with keywords `BOARD` and `GAME`.

   ```
   ZVXDIMUCIXANYNGBSYZIZZFWZTXXZNSKKSYVQCZVCAXECAYOHLPTZOCWACH
   INTHDUEGGHKPQTSKMEGYEEIEEXAXPIONLMACKHWPWQBYQNXTIVIZEPGLHAZ
   BONYNEFYRRXVJNNWIBGPRHRHDEAHNBHJBYIEBGCPQBLFZMGEYEUMPRDCMIB
   XAZWSXYXIYQFKOJMBQBORISUHMGOQZRMXBFSVUXNRBMVCRNKVEJLOPWPPYT
   YEGSHDSOYSXIKWLGTMPRUHATHDOIJSZLCOLSYFIGNOKBVSEUXTZA
   ```

5. Break this ciphertext with a dictionary attack. It was encrypted with keying option 2.

   ```
   ERMQNRLMWFTKUDJHHUNFKHSIUIMXCXFWJRRXLEXRSBLCHRCIYXQPKZZNGHG
   DQUMKYIXMLNZFSOHLDJJWOOPHZXTENDRKZHBDGZGJEKGFMIUJSZXOFMBGIH
   ```

```
JRLBVNXTDHLQJDQCLJMEFJPEQXZJLRCPVLHVAXNOLLUCXJDMLYMJWNZMUQR
JFOESDRWMJZBNQJWEZMPALHOGLFKXVBTUROUYDI
```

6. Break this ciphertext with a dictionary attack. It was encrypted with keying option 3.

```
IMUFQFYMAECPAUAMUSSOYDCVJIOUMIGKVRUHEFTCPIDMAFNORGTREEPPIIF
TMZUNZMTJMNSILAEWEDIUHNIBEATJBDRTDNZNHJBGDQUJDXAGCFGUWWCANY
GQYNERIDVHTTETIQJTMVJTMBLYIAVGFXSEDRWPRVDZCMZPQNDQJWETZNFXE
PXSDADLDDUMEHDLKHWQBJOVAFASTTPPIYQVKJXQREDDKWTKFQCJQTEEWSCJ
IMIQJOAXKBHZPTHEYZIRQNZMUYMMATDK
```

Challenge 1

Here are a plaintext and its ciphertext. Can you recover the key? The initial key alphabet was generated with keying option 2. Can you also recover the keyword?

```
WHENALANTURINGWASSEARCHINGFORNEWCRYPTANALYSTSTOJOINTHEWORKA
TBLETCHLEYPARKHEUSEDACROSSWORDPUZZLEFROMTHEDAILYTELEGRAPHTO
TESTTHEAPPLICANTSTHOSEWHOCOULDCOMPLETEASECONDCROSSWORDINTHE
ALLOTTEDTIMEWEREINTERVIEWEDANDACCEPTEDINTOTHEPROGRAM

PDMCOMMLKRTAENINDKQUKIMHQLRGOWAAJTMFUSVGEZCWNNCSPYOYNOZRYKO
MRFNPNHKHSQZEGBONPGRTPRZAPBYUXNVUCFDVTZFJTHIGTIEOTWXAWDSDCJ
IKTYIQWJMHZHEZSFMWCXKFMPEHTQGSSXQTNJMXYQUVPUNQKNWQTVHUCCYDQ
DGIFNYEFDAAIPLVZXULXDZVLEDWGMGFAQQUUMFNIHTRZFFSIKVBT
```

Challenge 2

Here are a ciphertext and the beginning of its plaintext. It was encrypted with a randomly scrambled key alphabet. Can you recover the key and decrypt the entire text?

```
SUCIEIEOANRWBUHKWYJTPDUTWRBEZJTOQLFHUDTTNBKBRMOZQJWVNDGPRAW
PUOUREAYNAAKNLPMDUSAJHGWHLCPMXUAVHEFFFENJIZNNEGWUFYRDXPXNDA
LMRPXMLPEKHBJQCVCRUDXGYXKVPUKZUCKQWSMYAICBYNMBFOTQBFCGQEPBM
PYINVFLVZRWZNVCHIXCPKDHCSDZLDJQBPHEXKJBCSILJUDZNXFDMSQZZUMK
FGERXBWNHZPWXQRPQKXRTYWDYLVUG

BLETCHLEYPARKISNOWAMUSEUMRUNBYTHEBLETCHLEYPARKTRUSTANDSUPPO
RTEDBYTHOSEWHOVISITTHESITE...
```

Unit 159
Homophonic substitution

In a *homophonic substitution cipher*, a plaintext symbol is enciphered to one of a set of ciphertext symbols. Each plaintext symbol has its own set of ciphertext symbols, and the sets do not overlap. The members of a set are called the *homophones* of the corresponding plaintext symbol.

In the monoalphabetic substitution cipher, encipherment is a one-to-one function from one set (the plaintext symbols) to another (the ciphertext symbols). In that cipher, it is clear that decipherment is also a one-to-one function and as such is unambiguous. This lack of ambiguity in decipherment is one feature of a good cipher. With the homophonic substitution, encipherment is a one-to-many mapping. If the choice of homophone is made randomly, then encipherment is not *deterministic*, but is *probabilistic*. Decipherment, however, is a many-to-one mapping, and is therefore deterministic. So, while one plaintext can have many different ciphertexts, all of those ciphertexts share the same plaintext. A probabilistic cipher has the advantage of being resistant to brute-force and dictionary attacks, since it is unlikely that such attacks will find a matching ciphertext.

There are several ways to build a homophonic substitution cipher. Here are just a few:

- Assign the same number of ciphertext symbols to each plaintext symbol.
- Assign a number of ciphertext symbols to a plaintext symbol in proportion to the frequency of that plaintext symbol. This hides the frequencies of plaintext letters and makes analysis based on monogram frequencies difficult or impossible.
- Assign ciphertext symbols based on a system, such as a grid.
- For each plaintext symbol, choose a homophone randomly from a set.
- Choose homophones from a set in cyclic order as the text is enciphered.

Let’s work through an example. Here is a short message:

```
this message was encrypted with a homophonic substitution
```

And here is a randomly generated key that assigns two homophones to each plaintext letter:

plaintext:	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
ciphertext:	m O F r y q Y A d p U V X L w c E l x z a B e j t Z
	D h i s u H K P g N T v W I J n o R C S b Q f G M k

The first letter of the message, `t`, can be enciphered to `z` or `S`. The second letter, `h`, can be enciphered to `A` or `P`. One possible complete ciphertext is

`zAgxWyxCmKueDCyLFRtnSuregSPDPwWJnAwIdiCaOxzdSbzdwI`

A hill-climbing attack can break a sufficiently long ciphertext. Since we cannot a priori know how many homophones are assigned to each plaintext letter, and to avoid running off toward a solution that gives the plaintext `THETHETHETHE`..., we need to define a textual fitness that incorporates both monogram fitness and tetragram fitness. One possibility for this is to use the cosine of the angle between frequency vectors (see Units 9 and 10) as the monogram fitness, and define the full fitness function as

$$F = F_4 (2 - F_1)$$

where F_1 is monogram fitness and F_4 is tetragram fitness. Since F_1 is always between 0 and 1, the expression in parentheses is between 1 and 2. The algorithm for the attack uses the parent/child paradigm. Child keys are modified in one of two ways: randomly chosen homophones are swapped, or a randomly chosen homophone is reassigned to a randomly chosen plaintext letter. Like many times before, we will use a margin of error to allow occasional downward steps, in order to avoid becoming trapped in a local maximum fitness; 0.1 is a good margin to try. Here is the full algorithm:

1. randomly generate an initial parent key
2. decipher the ciphertext with the parent key to get a plaintext
3. set the parent's fitness to be equal to the fitness of the plaintext
4. set counter to 0
5. while the counter is less than 1,000
    a. copy the parent key into the child key
    b. if we flip a coin and get heads
        i. randomly choose two symbols in the child key
        ii. swap those two elements
    c. if we got tails in the coin flip
        i. randomly choose one element in the child key
        ii. randomly choose a plaintext letter
        iii. reassign that element in the child key to that plaintext letter
    d. decipher the ciphertext with the child key to get a plaintext
    e. calculate the child's fitness of the plaintext
    f. if (the child's fitness exceeds the parent's fitness) or
       ((the child's fitness exceeds the parent's fitness minus the margin) and
       (we roll a 1 on a 20-sided die))
        i. copy the child key into the parent key
        ii. set the parent's fitness equal to the child's fitness
        iii. set the counter to 0
    g. increment the counter
6. output the parent key

Unfortunately, the algorithm does not always settle into a maximum and needs to be restarted with a new random parent key. Of course, if there are constraints on the cipher, such as our example above in which each plaintext letter had exactly two homophones, then the algorithm can be similarly constrained; this helps.

It is possible to specify a key using a keyword. Here is an example to how it may be done. There are, of course, other ways.

```
plaintext:    abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz
ciphertext:   KNIGHTSroundtableABCcDEFfghiJjkLMmOPpQqRsUVvWwXxYyZz
```

Some homophonic ciphers use numbers as the ciphertext symbols. For example, the *Grandpré cipher* uses a square grid to assign numbers to letters. The rows and first column of the grid contain keywords, and all letters of the alphabet should be present. The rows and columns are labeled with digits. The homophones for each letter are the various two-digit numbers made from the row label and the column label. Another example is the Mexican Army cipher disk which had five concentric rotating disks to assign three or four two-digit numbers to each letter.

Reading and references

American Cryptogram Association,
www.cryptogram.org/downloads/aca.info/ciphers/Homophonic.pdf,
www.cryptogram.org/downloads/aca.info/ciphers/Grandpre.pdf

Fletcher Pratt, *Secret and Urgent: The Story of Codes and Ciphers*, New York: Bobbs-Merrill, 1939, chapter IV, sections I-III.

Merle E. Ohaver, “Solving Cipher Secrets,” *Flynn’s*, May 19, 1928,
toebes.com/Flynns/pdf/Flynns-19280519.pdf,
toebes.com/Flynns/Flynns-19280519.htm

A challenge on MysteryTwister C3:
www.mysterytwisterc3.org/images/challenges/mtc3-madness-01-polyhomophonic-substitution-01-en.pdf

Wikipedia, en.wikipedia.org/wiki/Probabilistic_encryption

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, page 322.

Python tips

Python has a dictionary data type. A dictionary is like a list except that items are indexed not by integers but by any object. For us, it is most useful if those objects are strings. To define a dictionary, we could use this statement:

```
x = {"A": [1, 3, 4], "B": [5, 7, 8]}
```

Our new dictionary `x` associates the letter 'A' with a list of numbers. To retrieve that list, we call on `x` with index 'A.' For example, we might want to do something with each number in that list:

```
for y in x["A"]:
    print(y)
```

To add to a dictionary, use the `update()` function:

```
x.update({"C": "See?"})
```

or simply use a new index:

```
x["C"] = "See?"
```

To remove an entry from the dictionary, use `pop()`:

```
x.pop("A")
```

To change a value in the dictionary, simply assign a new value to it:

```
x["A"] = "Aaayy!"
```

The `choice()` function from the `random` module chooses an element from a list or string randomly.

Programming tasks

For these tasks, you will need to decide on a way of storing the key.

1. Implement the cipher.

2. Implement the hill-climbing attack described above. Allow for the possibility that ciphertext symbols are integers.

Exercises

1. Encipher this text with this key. Choose homophones randomly.

plaintext: `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`

```
ciphertext:   P H O N E h o m e A B C D F G I J K L M Q R S T U V
              W X Y Z a b c d f g i j k l n p q r s t u v w x y z
```

```
TOP SECRET XXX FOR YOUR EYES ONLY XXX THE EYES OF ET THE
EXTRATERRESTRIAL WERE MODELED AFTER THOSE OF ALBERT
EINSTEIN XXX DESTROY THIS MESSAGE AFTER ENCRYPTING
```

2. Decipher this text with this key.

```
plaintext:    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
ciphertext:   L y t k A u w b T 4 2 m q X P x 5 h e G o z s 3 v 6
              M   l B   c U   n r Y Q   i f H p
              N     C   d V     Z R   j g I
              O     D     W     a S     J
                    E                   K
                    F
```

```
LevQptOYeBCuhQrGcD2EvHRJcVfA3DitVfEtUxdBiHA3HevqyRngNiCXQHj
FfIiVtHFlJQoxxEjMXlmSsCitOeAnDKJCjgypItLZyDapryBhfSiEzDXxpY
tGoNITPXsQonlvPocMzCxiBuDhjFlJcLKGcDtUxbFhKA3GtSZHOUaElKsQl
TwUIYoqyBhfvSpqVwdHNngQbLzCYQIWtBkIdOGIdEYpryBiRudRqPxcSaCf
uSjDMtcmAGHEjWghRowcmvxjQxSiITRaNnIQVIeujD5pAYtvGbWfrL2DgJd
FtTxcDiJE3KdMjkBjHRyiBM2
```

3. Break this ciphertext.

```
rucdupmdEvrYJbRNGIsRuOElKOHdMMTCOnvPGKFEvblSbfAOvsTJfNOIKGn
mnDrlSdGnvsOWGCFsGNGUbRnOdJMCGNeCTTalnHNOluJJfBOJRmuHsRBRUb
RnOJDfnslhOATKoHrMqTCTnSBTlnrupJlnFAOCoHrMMOKOuSEIOcxrneJbO
CTlCOJGBTOnpcdJbbRNGIbGnOEldKsTdCJGfKEGCTZrSTzgvTMxyqcTWOlC
nyKnBlrcBfcOCOTHCTfFCOHCTlFPRGHWRmcHPbrnTPrnOYKTfaYKOTaDbGC
HYRCHYOKTfxETCrfcbdNboNnUxfdnIcfnTPfKnPGCuWOfvbTKWbTvsOCWbd
cOWrcTDyKKfuvYmKCTuSzfCOzOlCzlxxZlPcfSOTrpsvJTfJOTcrDQTCxri
mGCxTfFxOHqxOLqcTDQJcGluxRnTxduQJcgnL
```

4. Break this ciphertext.

```
afPSKaMjEnjNiaYNzgvLAAPNiWrLBMEtRDBXBvTLRWLAGoXrDYBlEPUNjBY
CWoizlbtyVamJefiNumVqXNJhbtyVjESYeLSKfVGETUJIbGSRufJqlNVgdD
vKZHimMnjUfZSMCgjEfYkjUmMWVTjAQhofYemQNwCofaZNzwDATUYyPWaGe
mjCrojSKPOuXEQYAdnHLRcjCgTtRBtALoXuGZBfjkZUmYSJtwTKNLoebqhN
VbXrtsQuiNkHcrLutwoZWsJBtJSuuZCwiDKlHTYIcKJbqlBzrZoXbbXSqaj
EzMAumLTyDWzhmYkjUfZWPlSvaiXNPwYhooCgKZnVzrjfYnYUmYSigtBPWR
ItgfrmhuumPqvMBMukCofjAAJSzCTaVvjSMuLkhRLBQrfPGESjWPODgKPBo
idWvnZwKUPjUbPdEJUEMHdHIqCSlalEVYArmVGEGWojWsiBDQBoiGKntRfu
TJJnKfleKZHJUQNUVuClIIqHCNvJSVzHqlSGWoZWsPSDJBoinCwrHiumPTc
```

EQwKadqZANJlItxtSRDeHcKrfEMCRmmXHLuChrGZWZcEUEXoKLoVLwWMvMc
HeAcIIYACBoMBwoGjCTzistoPgAYNlsZGvGWRKfigJlnHLRcLrLPgnjNJYs
QELWoMSsJSGQBoJgrZamtomaQlNPHYEBdEPsJEqbGRfrIqrhyiSHCuTLWoQ
GAhIIaYNzgaiNidItyJLSwYcWzemQNiaMcIvHiWYTUMyQSblWRchRiKfPvL
AAPNPWrtluLYSjArfJgYcBvYAajEvwtTZAefJJTgiWoiYAgUJioflBzDrAj
IbZawKmhuefQnMNPfMkYUfMSPwKmJNQdNVLSlWqIdSRChRVufikMEiAlcIr
qtTrmleIXWRCdRJlThgoLVSuLAtoXBzojWsJStVBesJmDoIQZAgUPVomrfG
waGIbHPGbbcwKEluJzLSzCJoMCEgPKfPdorclbojWvtrLZSYAJWRbGwmaLu
mEiTUVoruZfMnMUmYSPwkCTeHJCSzQNgujZzlWzlUUNQotleQzHVAZNQemJ
SdKcNiMAufPDNREYaKmlWzrfiUMwTDHbJkVhWgjAKfJGEkteGRhKLjBaLbI
gQikUNhouLodIFciwKDZST

5. Break this ciphertext.

86 42 25 81 84 63 70 97 87 74 25 27 42 97 84 89 41 59 58 77
84 88 81 21 17 31 18 23 35 84 31 63 75 27 47 84 60 24 23 34
42 27 24 77 74 87 45 93 93 57 45 14 60 24 23 95 70 94 47 31
21 86 14 18 75 88 27 34 25 14 23 70 69 81 81 11 31 18 96 79
84 97 95 57 70 87 63 14 23 66 88 73 21 17 14 18 13 77 22 13
81 69 41 94 69 96 35 11 97 79 95 57 42 94 78 27 87 58 78 88
14 23 86 46 41 79 58 13 74 79 60 42 47 75 47 36 69 27 46 70
78 25 27 87 74 47 97 94 45 41 89 66 57 33 41 95 33 66 11 66
93 70 42 57 86 18 96 22 93 78 86 42 89 96 34 66 24 88 60 32
14 23 84 25 73 21 48 88 34 74 78 93 25 14 60 48 88 34 66 78
93 95 17 74 95 70 95 11 74 66 94 78 95 33 17 73 60 11 14 18
87 89 13 88 87 87 25 27 77 87 46 63 73 73 24 27 66 59 79 13
24 35 13 73 78 48 69 84 77 27 74 94 69 96 22 86 97 78 81 69
97 87 88 81 73 25 66 35 31 75 69 17 23 11 97 75 35 36 78 27
74 59 77 27 77 97 96 57 66 59 86 14 18 88 27 22 33 14 60 63
66 63 73 73 95 45 66 88 97 13 93 24 32 23 11 31 18 75 32 27
13 69 74 57 48 18 93 32 70 18 27 69 69 96 46 57 97 87 70 69
25 27 34 60 24 79 13 88 31 60 32 97 89 81 78 96 75 78 66 94
63 66 48 78 78 81 97 88 34 57 73 23 25 34 74 11 58 77 89 66
97 77 73 48 42 59 77 93 69 97 31 47 41 59 86 14 18 41 57 48
27 86 18 59 41 69 14 46 24 95 33 66 17 32 46 48 93 89 18 94
74 79 31 77 23 63 31 60 73 22 31 77 34 47 46 24 14 22 42 87
35 58 33 75 94 48 47 93 13 77 22 13 73 69 95 33 41 48 35 93
32 97 22 33 31 23 63 31 70 87 35 45 14 23 17 42 45 27 96 66
35 93 33 31 95 88 75 21 66 22 84 63 93 23 74 57 84 31 95 45
66 42 59 69 93 79 87 31 57 66 94 89 14 18 95 69 59 17 96 23
58 69 96 57 84 95 45 42 74 59 24 87 88 58 79 18 27 88 70 58
47 87 87 63 31 60 18 27 88 70 87 21 70 88 31 60 60 77 32 23
97 17 81 79 96 75 77

Unit 160
Polyphonic substitution

In a *polyphonic substitution cipher*, more than one plaintext letters are enciphered to the same ciphertext symbol. Here is a very simple example of a key for such a cipher:

```
plaintext:    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
ciphertext:   A B C D E F G H I J K L M A B C D E F G H I J K L M
```

As you can see, in this example, both `a` and `n` are enciphered as `A`. Deciphering a ciphertext is not unabiguous; there are many ways to decipher a text; we hope that there is only one way that gives a sensible plaintext. In other words, while encipherment is deterministic, decipherment is not. This makes polyphonic substitution a bad cipher.

One can generate a key from a key phrase like this:

```
plaintext:    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
ciphertext:   O N C E U P O N A T I M E I N A L A N D F A R A W A Y ...
```

When the key is created this way, we call the cipher a *keyphrase cipher*.

An attack on a ciphertext uses the parent/child paradigm in a hill-climbing algorithm, very much like that in Unit 36. However, to evaluate each key, we must find the best decipherment with that key before calculating the textual fitness. The method to do this resembles our hill-climbing attack on the Vigenère. We start at the first position in the plaintext and try each possible character that can be there, given the key. We keep the one that gives the best fitness for the entire text. Then we move to the second position and try all possibilities that are allowed there, and keep the one that results in the best overall fitness. This continues until we reach the end of the text. We start over at the beginning. The cycle repeats until an entire pass through the text results in no increase in fitness. At that point, we believe that we have found the best decipherment for the given key. The overall attack starts with a parent key, then modifies it in every possible way, and for each possibility (child key) finds the best decipherment. If it finds a higher fitness than the parent has, then the child key replaces the parent. The algorithm terminates if no child key results in a higher fitness. Unfortunately, this algorithm is not guaranteed to find the solution, and we may have to restart with a randomly generated parent key until we can find an acceptable plaintext. Even then, some human intervention is often needed to finish up.

Reading and references

A challenge on MysteryTwister C3:
www.mysterytwisterc3.org/images/challenges/mtc3-madness-02-polyhomophonic-substitution-02-en.pdf

Fletcher Pratt, *Secret and Urgent: The Story of Codes and Ciphers*, New York: Bobbs-Merrill, 1939, chapter IX, section I.

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, page 787.

Programming tasks

1. Implement the cipher. In your decryptor, use a function that finds the best decipherment for a given ciphertext and a given key.

2. Implement the attack described above, for the case in which each ciphertext symbol can represent either of two and only two plaintext symbols.

Exercises

1. Encipher this text with the given key.

plaintext:	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
ciphertext:	H M F A A K E B C L I D J G J E D K G C M H B L F I

Polyphony is the art of imitating sounds of various kinds, usually, without attempting to deceive the hearer as to their direction. It may therefore be studied independently of ventriloquism. Already the art is much in vogue.

(from *Three Hundred Things a Bright Boy Can Do* by anonymous)

2. Find the best decipherment of this ciphertext. Use the given key. You may have to finish it by hand, once your function gives you something recognizable.

plaintext:	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
ciphertext:	D C I M J K A E M G C L A F L I G J H E D B K H F B

```
DFMAJILLLFHDELFEEJHEMLJDFMEDEJMEEJKELLJHIEJAJLKLMKJKEMIEKDH
DELFIJJHIJHHMBJDFMMFDMJGDDEJDHDHLLDEMLFEJEDEJMKLHCLDJFJEJED
EJMKLHCLDJFJEMAEHEJJJEEJEDEJMEMHHELIDFMEMHKMKJDFMEMHFJMAECL
DJHJBJJFCLJHHJMFJMAECLDJDFMKMEEMFMJHIJMCDCLJCMEEJJFJHHEJEDE
JMEMAHJLK
```

3. Break this ciphertext. Each ciphertext symbol can represent two and only two plaintext letters. At the end, you may need to tweak the decipherment by hand.

```
GFLLKILLJEAFHCJIEEBCHIEAFABCFKCBEAFLCCJEAFBCDEFICLBCCCHBCBC
EBCLKCEHCBCEAILCCJBDGBHDABILLEFBAEFIHCBLBIEJDECFGCECJBCBCKC
EIEIEECLCJBCBDEAFIAJDCAGIBLFBLDCCBFFHABIAFGCECJJDBCLALKIBFH
IDIABIKCEJCEBCEAIBCJIBFJEJIFBIGLDHGECFIABCGFHEEEBCCMGCLACJA
```

FECCFJAEFABDEGLDCCFECHIEAFFCECCEABCJDGEDIDCJLIJKHBFHCAIEECI
EJIICHFABCBHCLLJBCEECJHFHCEHBFLFEKCBECJDELFHAFECEHDABCILBFA
BCBJDJEFALFFCLDCCGFLLKEDJCIFIEBFGGDBLEIEECELIJKLFEJJLACJABC
HAFILDIAIEJABCKEBFAJGAHFEAFBDCEIGIDEABCKLIHCFJADEAFILFKCLKL
FJEGDEGBFFHFJAEADLLEFEDGEFIJBCEECEABCLIJKGJEBCJIFJAAFEIEJIE
FABCBHFHIEBJBBDCJDEABCEGFLLKHIEELDCEADIDLILLKHCIEJBCJIEJDEI
EBFBAADHCIEJHFCBFIHFJCLEHCBCFBFJGBAIFBBCBDEEGCLADFEIEJIGGBF
KILABCECHCBCGLILCJJGFEIFBHEIEJCKCBKJCEDBIFLCJCAIDLFIABCGFHE
EHIEGFDEACJFJAAFIEECIEJABCGDBLE

Unit 161
Polyhomophonic substitution

Polyhomophonic substitution incorporates both homophonic and polyphonic substitution. Here, a plaintext symbol can be enciphered to any of several ciphertext symbols, *and* a ciphertext symbol can represent more than one plaintext symbol. Encipherment is probabilistic, but since decipherment is not deterministic, this is a bad cipher.

Here is a quick example. Consider this (randomly generated) key:

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

With this key, the word `SECRET` could be enciphered as `ENBDQJ` or as `ZQNRVS` or as `INBWQY`, but `ZQNRVS` could also be deciphered as `MEELEY` (not a word).

The attack on this cipher is similar to the one in the previous unit. We must modify it to accommodate the fact that each plaintext letter has homophones.

Reading and references

A challenge on MysteryTwister C3:
www.mysterytwisterc3.org/images/challenges/mtc3-madness-03-polyhomophonic-substitution-03-en.pdf

Programming tasks

1. Implement the cipher. For your decryptor, write a function to find the best decipherment for a given ciphertext and given key. You can do this by modifying your function from the previous unit to lengthen the key.

2. Implement the attack described above, for the case in which each plaintext letter has two homophones, and each ciphertext symbol can represent either of two plaintext letters. You can do this by modifying your attack from the previous unit to double the length of the key.

Exercises

1. Encipher this text with the given key. Choose homophones randomly.

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

There were fine lawns and beautiful flowers everywhere, but Polly and Rose loved the shore, and surely the salt air was delightful, and the beach a lovely place on which to romp.

(from *Princess Polly at Play* by Amy Brooks)

2. Find the best decipherment of this text with the given key.

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

```
WOHWSPCCDXHSDWREKWELDKXIJXCCXCSWSCPOXCEDTWAHJYXMWHOSHJOKLXB
OXOCJJXSWJFBIWHPOPEBCOXCFXCJXLDPSXIXFHPXHPOIPSKPHXCPOHJOGFZ
OHAASPOAOYSWOFPEBCOIGDEIOCREZSXSUZHHJOSWYAOSAWWCCXNQPWDNCYS
EBONPHXXCBUDWSHPWIWQXOLNCCAOKAWOOXXMPXHWAHJWIXFJGFVWACHJODS
PWGPODNICXNSPWJXDDCYAWYIJJPHWCIHJWJEBCWSPWOHPOAWXIGBOMDCKXX
KPXSJOIOEJ
```

3. Break this ciphertext. Each ciphertext symbol can represent either of two plaintext letters, and each plaintext letter has two homophones.

```
QZAGUXPRKKVNFOTAPQBUWNQNVUWMAODHPAWMROHKFMBOCOBNJYUJWHRNOSO
KNPAVQWNVUJRNUSWQAHMWPYAVOFUPRWRQWOFMBUPSSOPNJRCHRPBGUPYTYQ
PGMCTVPSQLAUSXUHWCNGUKNQSNJWCGQHRNMCUSQXQIOGQAUJAQXGYHCHOUJ
WAGUJWAOSRPNHQWQDXKXQNGPKXGPAGEQWWUHPKMWPRTUFVNVPYPNASQLNHM
RXONIOUWJWWJMJUVQDPRJDYKMXOAJXUKMMUBDUQBBPTHQDUJZZOBORTOKHR
BMJWNMCMJURQRWEUABQTHPYUHZCOBUWTUKJRBMJWAMZMJOIIHYYQDIQKPXU
QNIQSRJNGFVNTQDQANBQTAJMWXONRUORHWUHAJWVQDKQDDTMWNSQKNKPDDL
QFGYUZJNJUPPSUZQTTPKUWQRAVHKKJWDYUPAQDURGJDOMRUCUAKPPWMRGOS
UOYPONGULMWZYJLNIGJTVOQLGRUINMSWCOWOSQNJQRLQWRQAXVARQDOPCQJ
RPNAGUWJTNQAHQWQCNVOQCUKQRNVHKTBQRUOWPTOWUNBJWCJWAVQNQXUBJT
PRUDUXOWNNQIVHLGNVUUHTAQNJQWQZNGOPDUPFOQWPSODPAHMOYERQAVHWC
PWUAGOIUPDAGQZAVOUSQXQAJLZJOYWXUTMFUKMXAHQVPJNJPTTBJQVPNVUB
UZMBUAVQAJNVQPNUOWPMYHNNDONMTTGUUHAVQKNOORUWNOBOWXTARMNMOSE
CPSAGOCSUQNSTPKJQWPWUZSOWTVRQAUDJPNPRHAVAVUJSLMRLUWABPAJMRQ
WNGOYHZOJXXUWHQNODESMTWWNGOXHRNGOFPHWJCWMBUJAAGOUWDDHPGGQAU
RMSHOWHAPYJANYUNTARQAMZAOWPRWWMAIHAGFTTVJRPJDVAAVONSVNGKOUF
PNMNUNGQNNVUUTSMGUQRWQNJQWKIHNVAGOHSPASQWDYHROPQZLDOQAQCUVP
```

MOWHZCJLTDNEJRVWUUSKNQWWJWDUPTVMNVUBIVHYUAVUKTWUOSKAQWWPXUB
JLPRMAPAPDDKNOUBUWQRWWKUUHRNVUHBMRRWQNJQWQDGSOBMKPUKPJMRKNG
OESOCQBUMNGUBRQNJQWQYBSUBQKKUPPHMWPIJAGJRUJCZUSORTUPXQYOXOW
NQBVMPAJDJAEAGOBUQBUUZLOGNHQWKNQNGHKKAPAUFOWNMCTMTSKOJPGUPS
MWDKMZCOWOSQDAOWWURTJOKNGOASORWQZOMOWNPPHWLOAVURQSUAUWXQSUA
GQRAVUIQSJAKOYZXSHWCPVQXUAMTPNGUZPLNAVPAAVUOTSMGUPWFJWWHPAB
JXQD

Unit 162
Pollux cipher

The *Pollux cipher* is another cipher that uses Morse code. Like in many such ciphers, a plaintext is first encoded in Morse, with a separator between letters and two separators between words. The dot, dash, and separator are then enciphered with a homophonic substitution to digits. We saw homophonic substitution in Unit 159; recall that each symbol can be encrypted to one of several ciphertext symbols. The key is the mapping between Morse symbols and digits.

As usual, let's run through a short example. Consider this plaintext:

`THIS MESSAGE WAS ENCRYPTED WITH POLLUX`

We can use this key, for example:

0	1	2	3	4	5	6	7	8	9
·	·	×	–	·	·	×	–	–	×

What this means is that dot can be encrypted as 0, 1, 4, or 5, and we can choose how to encrypt each dot separately. The same is true for dash, which can be encrypted as 3, 7, or 8. And the cross as 2, 6, or 9. We can arrange to make keys that can be remembered as two, three, or four Morse-encoded letters; for this example, the key could be thought of as "`IDM`".

The first step is to encode with Morse.

```
–×····×··×···××––×·×···×···×·–×––·×·××·––×·–×···××·×–·
×–·–·×·–·×–·––×·––·×–×·×–··××·––×··×–×····××·––·×–––×·–··
×·–··×··–×–··–
```

Then, replace the Morse symbols with digits, according to the key. For this example, choices of ciphertext symbol from sets of homophones were made randomly. One possible ciphertext is

```
3250509446510223860605561412182884242217894720049952812748 1
2435274336583563612304620339156860444965874673320855903149 1
5868418
```

We might instead choose digits cyclically. So the first dot would become 0, the second dot 1, the third dot 4, etc.

To break a ciphertext we can brute-force the key, or we can perform a hill-climbing attack. Because the cipher has two stages, Morse followed by homophonic substitution, we will attack the substitution. Once that is accomplished, it is easy to decode the resulting Morse code. Since the substitution is between an intermediary text of dots, dashes, and separators and the final ciphertext of digits (or some other set of symbols), we must build a frequency table for a corpus of English text that has been encoded with Morse, and use this table in defining a fitness function that we will work to maximize; we recommend tabulating the frequencies of 10-symbol groups (there are only 3^{10} = 59,049 such groups to consider). The hill-climbing algorithm will be similar to ones we have used before. We will use the parent-child paradigm. For each step, the parent key is modified by randomly selecting one element and changing its value; i.e., pick one digit and change which Morse symbol it represents. The ciphertext is decrypted back to Morse code with the child key and its fitness is calculated. If the fitness exceeds the fitness of the parent key, or if it comes close on rare occasions, we replace the parent with the child and continue onwards. When we are no longer able to find a higher fitness for some number of tries (like a thousand), we terminate the algorithm and decrypt all the way (including decoding the Morse code).

Reading and references

American Cryptogram Association, www.cryptogram.org/downloads/aca.info/ciphers/Pollux.pdf

Programming tasks

1. Implement the Pollux cipher. Be sure to check that the key is valid, i.e., it contains at least one homophone for each of dot, dash, and cross. It is your decision how you will choose from the available homophones when encrypting each symbol. You may include an optional parameter to determine whether selection is random or cyclical.

2. Implement a brute-force attack on Pollux. Remember to try only valid keys.

3. Encode your textual corpus with Morse code (use the cross as a separator between letters, and two crosses to divide words). Compile a frequency table of dekagrams (ten-character sequences) from the result. You can use the table you made in Unit 145.

4. Implement the hill-climbing attack described in this section. Use your frequency table from exercise 3 to define a fitness function. Experiment to find a good margin to use for allowing downward steps about 5% of the time. Terminate after 1,000 tries if the fitness does not change.

Exercises

1. What is the size of the key space for the Pollux cipher? Only include valid keys.

2. Encipher this text with the same key as in the example above.

```
CASTOR AND POLLUX WERE THE TWIN SONS OF ZEUS AND LEDA. ZEUS
SEDUCED LEDA BY DISGUISING HIMSELF AS A SWAN. UNFORTUNATELY, ONE
OF THE TWINS WAS ACTUALLY FATHERED BY LEDA'S HUSBAND, SO ONLY
ONE OF THEM WAS IMMORTAL. WHEN CASTOR DIED, POLLUX SHARED HIS
IMMORTALITY WITH HIM, AND THEY ALTERNATED DAYS ALIVE AND IN THE
UNDERWORLD.
```

3. Decipher this text with the key (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) ↔ (−, ×, ·, ·, −, ×, ×, ·, ·, −).

```
0821286428550299199917291603959310406749550637781245916024 8
1706788541909670251241985422562492100057937684731234548895 6
3823620108251452095875021582758852271918530753321720033669 6
8337635084051809575207635508026848750394506864059767672360 6
8035806534598643255467788586683426345405004623412721582371 7
5307257698154906330856467976049197406848939518913028634326 5
3703549917701803514946334316952832136523767310773579321231 4
8140268225184184858562876796771972160500415373313052229575 5
9833525350265387213050592436877857692866289217926944509163 5
0025907533310937495687818719259847131191823268682680351839 2
5705018333675397614082521830583751823768064876543886357193 5
1078627128354075239172633257503365241722617011222529062910 7
5174696505888712677579211929310446985039862529086418350945 4
21293089
```

4. Break this ciphertext with your brute-force attack.

```
7646364311625374521503865256856122945035748641639569896535 0
9437410458805865668109355528597250277729452518217266683451 2
9245301876891825361908622426572193598110345695722078959862 8
6421993691347584050269817253184093200256827713041898637043 3
0585684112367637758458942702374340473721900347589419255524 4
6356453207436035261098869693777804030118989102264840341083 7
8966134442115355920669252844521041229726963307021399935056 3
6048136598166885442165633531916377426613923102703870929121 6
6342049203601309926704973848009039121908375840748526099254 7
8171635772456279743806299923461181864912092086605393701327 2
5452857034758846930293645139295084410778269356284261663972 7
8344807346246472361841349987742596997303604264624572644620 2
9427032767936652941924369731513080098866998043601429835724 9
5725310603661335452067982469869219672108502122555804023099 2
0274269270981980164309870262253057301871829011357486792600 2
9844398403050564220635638520900302610802854841573586091319 9
3278696180324395720987121613804807629381963986351886480063 8
6608043916789937552140306623745290568252091024773109368800 3
```

```
768817413186402612670622658762401889489662378111131824150 25
572078717039010874308413276926792162960481634927563852999 03
182911985743979309926269133426911864343317502167214193976 12
342961629336903194870830584051315934458060316038430001312 29
962417371773951564
```

5. Break this ciphertext with your hill-climbing attack. You will find that it is possible despite the brevity of the text.

```
194925739853721143540660461564856257706111381886975538846 79
076387357025324608906522728112441627589391520931156988827 29
699024265896428129196095230524023448602146053792 08
```

6. Break this ciphertext which uses a larger set of ciphertext symbols. You may need to reduce the margin in your hill-climbing attack for this one. Does the encryption here use a keyword?

```
CEDAMPSBVEXTDHYVACSBUPVOPCGKHWNNHFRXBETJFTLKBNLQPMNSEAMSADM
HSJKNHUBHGRUEAYCEZAOOQHZYRHOFKBFSXRLTAAWSXLSFUHYSSBGIKHTSTZ
LADBEUVVQNCVZKLIEHTGCALBCTMOIQOQFOFSGSONTQDEPGODRKDEKNBMHDS
YTTLVCVPPPBEMOXPXOQSBXVFUSMXD
```

Unit 163
Doubled-over substitution

The *doubled-over substitution cipher* was invented in 2011 by Viktor Veselovsky, albeit by a different name. His name for it was “monoalphabetic substitution with camouflage,” for a reason that we will explain below. The cipher involves interweaving two halves of the plaintext and a substitution that involves both upper- and lower-case letters. It is a probabilistic cipher, since the point at which the two halves are divided and the interleaving are both done with some randomness. Decipherment is unambiguous; it is always possible, with the correct key, to recover the plaintext.

Here is an example to show how the cipher works: Start with a plaintext, such as

```
The quick brown fox jumps over the lazy dog.
```

Split the text at a randomly chosen place near the middle:

```
The quick brown fox jum
 ps over the lazy dog.
```

Throw away the punctuation, convert the first part to upper-case letters and the second to lower-case, and replace spaces with ‘_’ in the first half and ‘/’ in the second:

```
THE_QUICK_BROWN_FOX_JUM
 ps/over/the/lazy/dog
```

Next, interweave the two parts. This is also done randomly.

```
TpsH/E_QovUerICK/t_hBROe/lWN_aFzOyX/_JUdoMg
```

The key is a permutation of this plaintext alphabet:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ_abcdefghijklmnopqrstuvwxyz/
```

For this example, we use this key:

```
plaintext:    ABCDEFGHIJKLMNOPQRSTUVWXYZ_abcdefghijklmnopqrstuvwxyz/
ciphertext:   ROUNDTABLE_knights/abCcdeFfGHIJjKlMmoPpQqrSuVvWwXxYyZz
```

After making the substitutions, we arrive at the ciphertext:

```
aSvBzDftrXbjVLU_zWfMOsgjzpcifGTZgydzfEbJrnl
```

In decipherment, we first invert the substitutions. Separating the two halves of the plaintext is possible because of the use of upper- and lower-case alphabets.

Optionally, we can remove all spaces from our plaintext, and remove '_' and '/' from the key.

Veselovsky's original name, "monoalphabetic substitution with camouflage," came from an earlier version of the cipher in which the plaintext was not doubled over. In that version, the plaintext was converted to upper-case letters, and lower-case letters were added randomly as camouflage. The substitution proceeded as in the example above. Decipherment involved inverting the substitution and removing the camouflage.

We can attack this cipher with a hill-climbing algorithm that seeks to maximize the fitness of the plaintext using the parent/child paradigm that we employed in Unit 36 for the monoalphabetic substitution. As we have done before with other ciphers, to avoid becoming trapped at a local maximum textual fitness, we will allow downward steps by a margin of 0.2 about 3% of the time. In order to converge on the global maximum, we must shrink this margin down to zero after we have reached a fitness at the lower end of acceptable for English (see Exercise 1 in Unit 11). Another complication is that there is about an even chance that the algorithm will have the two halves of the plaintext in reverse order.

Reading and references

Challenges on MysteryTwister C3:
www.mysterytwisterc3.org/images/challenges/mtc3-veselovsky-11-mono_split-en.pdf
www.mysterytwisterc3.org/images/challenges/mtc3-veselovsky-13-mono_nospaces-en.pdf
www.mysterytwisterc3.org/images/challenges/mtc3-kallick-27-cloakedsub-02-en.pdf

Bruce Kallick, "A Modified Simple Substitution Cipher With Unbounded Unicity Distance," Cryptology ePrint Archive, report 2019/621.

Programming tasks

1. Implement the cipher. Allow for the choice to include or exclude spaces.

2. Implement the attack described above. Remember that spaces may or may not be included in the ciphertext.

Exercises

1. Encipher this text with the following key:

   `TWAIN_story/aBbCcDdEeFfGgHhiJjKkLlMmnOPpQqRSUuVvwXxYZz`

   I have found out that in one way you are quite different from other people. You can see in the dark, you can smell what other people cannot, you have the talents of a

bloodhound. They are good and valuable things to have, but you must keep the matter a secret.

(from *A Double Barrelled Detective Story* by Mark Twain [Samuel Clemens])

2. Decipher this ciphertext with the following key:

```
HOMEWRK_duetomrw/AaBbCcDFfGghIiJjkLlNnPpQqSsTUVvXxYyZz
```


```
BWHzVALJzhaqGyUHzgQimEzGkabltTeFGPRUHMWzaGVHmqqzExJGdTJzokw
rPgBWmiBGRzxLdJaQzBxaqTnlGQkzVlEprbJzOItgpWJEzGRdWTqAMvQWit
FzgGckg_lWQzmgGQiz_dVaGLJOHMzpeGcgUVJHaGBbAmTWzxJEQGVczLWqA
pWJGzB_VqzWGALWJcHPSAzLlEaUGrRG_daGzMjrgVLmJTazMdWqQzmBdrbV
aLJmzjgWaTapGHmE
```


3. Break this ciphertext.


```
dsLIiVEzmLivklyWeIpzdiDqBKIGzmkuLVDzJrIWDABIbvzLcsDbipidIVz
EacSIueNsLIqEjtcdeIekzt/XLiVzMRIqktzvmAiIsLcIyvztmRuuNEKzLI
DXRQiIyteeqzmiKzviyLPkqzmNQzIDdsiDX/LiBYzIvmesLIkzftalWyuYz
LuRAVkzULIaiEMIGstiUsIdNsLIqzNqszmDkBuzIOmLLkAiI_FutyevHziq
IsDKzuiMBIfkKzWDqAvNpztNvdzjsiQkzvRLBzmkuzIEcaQ/NIqKsLczImR
dXbtzmcktuezIKkiuDzSAuBINqjAkazGIXRWyDHpKIzmtiAIsLcwkIz/Llp
RX/acqHIzGteivszIemkuzsLIUvDymuRi/vzNIlz/amckAzmiKzVtkkqA_z
ImOkuzLsNqztVABWjItemIGzistzSiyVVLItANIERqzvcaAeImeksLzIQCF
kQtRLeIBuYFzjidrpQkHIKAtz_smekuIGDzidIqKzAVmkzauRGIVUkyadzt
iA_qItKAzIXeskLqIyavFzmRQkBzvmkIdukzSsDkumiSVzV/mkzXRWLpIOL
KzeGltgeIDNIqKBzSauFNqOyLIjbLkzlDRUuLzVWukpIOLeGLLYAIzemsLI
ekGazjIsLRWcIbpDKddtaAIyaarLzqkwBIF_yHIkteIbuzmDitwkALBIsLc
IeaIcLz/JkkL/OqzVLcWjIdsmLIELzyeizIVtNppeYzIKRGMziVzMRziXiY
zipvRMkvmkuzXNvmzizVvuiqMkzXRQiq
```


4. Break this ciphertext.


```
evEevFfOmBbGoSwFbVsvXMZBbvHOafdfAmiFwfRmBWkafblOjFwSflAPVbR
TNTHSmTNTXfduikbGTaRRffTFSeBFTwfukOafPWMVDTubHFRPwqfIQYmAbm
EEbaMVwWEvWBeBOjvjBufdafWmjBMpXVveBowbCfmEMYWPvBDOlVXfWkdOE
FwWRjFeuxBmqTEXVCeMmdqfePkFEjkIZvTvOdefmlfheEFfBqLmfSWEnLwv
OTfafNbjKlmmVvTSnEXmTWAugTWFNejKXvvmbBluNbqdeZVFqEECfLmefEn
eqmBFjuwSfYOZEFdVTRjSGZooANqTYmfulVOjbxdEmVGvPdkgeQZgkWjqPA
kFmwVWTFutuTeFwvmkWEIvfPfBExAmOjbPHAiuCVYfdlfLOFZveOBmVoHQX
SeQkFjKKjnwEbmafeBEfLFEwftXuACtqFwmdvbuGoPqwqPXRjFwSMkjufuW
oPTfEeSFeNEqTYBbFWmuCWmEEeRgfPvFweBoFjbIfWNmvPMTMTDuaedfDIY
OiwPGqaWmTvBfOifmuEBbdNqDbfEeFTYHmWEFuVmfeFVEEtHfmkfFBPfEEj
qqbZvOmBYOPFvwfDmTfuEfdtjqmFWedZemqFeEWqEfBTFeYfBFSIPnOvKqT
feBToReaCmuVfkOjAuARYPvfOvOmPDFbWLuFMmeqBFPkPfATXAeofVBKuFA
imFwdjvbOGVommPNZTXwkFjwSSTvfMPVkTExmuYOWafEFZvOmPdRRbPBoDn
ujfSvMEejPVZdjqufEmFwjWMFZYOPWvBmSeVTXauWNFPfmGqMVYEPWBvOMD
mVkfBmDbHSMfjDSHeFwvVLfjdFWeVBSXRmCbHfduTXEVDellfdjVveOmVBo
```

jbSMVTBANvOieBDfmVmjVodOffTuImYOGmumjVZFBTbXFReBsjZveuBDjRl
dtjYbaOFwbTEfqmMbjZYlPFwvfOWBTeXavWALNdPSfWFeMbBDfPEeSnjdVf
PfgfdSLndfWqmFfESjlvbPxdmaqDbjibMGDPbmGVIVFeqPttmu

Unit 164
Duplicitous ciphers

A *duplicitous* cipher is one which can be decrypted to two different plaintexts, depending on how the decryption is performed. The structure of the cipher does not involve interweaving two ciphertexts; rather, all ciphertext characters are needed to decrypt each plaintext.

A very simple duplicious cipher was invented by Jack Levine. In his cipher, two sets of numbers are assigned to the letters of the alphabet:

	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
set 1:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
set 2:	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52

The first set is used for the first plaintext, and the second set for the second plaintext. The ciphertext consists of pairs of numbers. In each pair the first number is the average of a character from the first plaintext and a character from the second plaintext. The second number is the difference from the average to either of the two characters. For example, if we want to encrypt “definitely yes” and “possibly not so,” then the first characters are d and p, whose numbers are 4 in the first set and 42 in the second set. The average is 23, and the distance from the average to each of the two is 19; so the first two numbers in the cipher text are 23 and 19. The full ciphertext is

23 19 23 18 25½ 19½ 27 18 24½ 10½ 18½ 9½ 29 9 28 23 26 14 33 8 35½ 10½ 25 20 30 11

Decryption is simple: For each pair of numbers, the character in the first plaintext is the difference between the numbers, and the character in the second plaintext is the sum of the numbers.

Let’s work an example of another duplicitous cipher and run through the encryption of two plaintexts. This cipher will encrypt one plaintext in a base-5/Polybius-cipher scheme, and another plaintext in a 5-bit binary scheme. The first plaintext must be five times as long as the second; if not, it is padded with nulls (usually X) until it is so (spaces are not used). The base-5 encryption will use different symbols for the horizontal and vertical labels of the Polybius square. The binary encryption will involve permuting those symbols. For our example, we begin with two plaintexts:

#1: DUPLICITY IS THE SAME AS
#2: LIES

We also need two keys. We can construct them from keywords `CANARD` and `DECEPTION`. The first keyword is used to fill the Polybius square. Remember that we have only 25 spaces, so we must remove one letter; typically we merge `J` into `I`.

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

To encrypt the first plaintext, each letter in it is replaced by the labels on the rows and columns of the square, just as in the Polybius cipher:

```
 D  U  P  L  I  C  I  T  Y  I  S  T  H  E  S  A  M  E  A  S
AZ DZ DU CX CU AU CU DY EY CU DX DY BZ BV DX AV CY BV AV DX
```

From the second keyword we construct this keyed alphabet, and assign numerical values to the letters:

```
D E C P T I O N A B F G H J K L M Q R S U V W X Y Z
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
```

The second plaintext is converted to a 5-bit representation based on the keyed alphabet:

```
L     I     E     S
15    5     1     19
01111 00101 00001 10011
```

We take the ciphertext from the first plaintext and swap the two symbols if the corresponding bit from the second plaintext is 1.

```
AZ DZ DU CX CU AU CU DY EY CU DX DY BZ BV DX AV CY BV AV DX
 0  1  1  1  1  0  0  1  0  1  0  0  0  0  1  1  0  0  1  1
-------------------------------------------------------------
AZ ZD UD XC UC AU CU YD EY UC DX DY BZ BV XD VA CY BV VA XD
```

The final ciphertext that hides both plaintexts is

```
AZZDUDXCUCAUCUYDEYUCDXDYBZBVXDVACYBVVAXD
```

The key to cryptanalyzing the cipher we have just described is to separate the ciphertext symbols into two sets, those for row labels and those for column labels. Once that is done, the rest is straightforward. Let’s work through an example. Suppose we have this ciphertext:

```
AJCDICHAFBHBAEHADFHGJAHAEIFBBCIEFDJADCICHAFBJICIBFAJHAEIBFI
EBCCAIHEAEBHADHHAAJEIHIHBHDAHAEDCCIFDBHBJHGHBJADCICCBHABFGH
BEHDDCGHBEBECIBFGCICHAFBJADCCIEDFDAHJAAHHBCDFBHGJAAHEIFBGHE
```

```
BHDHAEADCCIFDHDCDHGBEBECIBFGCICEIBCJAJBCIFBJAGEBFHABFCIJAIC
CIBFHAJAHIHBICCAIFAHGJCIACFDGHCAHAJGFBHIIFDECIFDHBAJEIICFBD
HEICACIICHGDHDCAEEBGHHABFAJCIJGAJBEICJAAJCIFDGHFBACHBCIFDAH
GHAJAHEIBFGHBFACBCFDGHHDJAAHEIFBGHJAHAEIFBJAIECDHAACICAJCDG
HAJCBHGDHAJJGJGJG
```

We begin by breaking it into digrams:

```
AJ CD IC HA FB HB AE HA DF HG JA HA EI FB BC IE FD JA DC IC
HA FB JI CI BF AJ HA EI BF IE BC CA IH EA EB HA DH HA AJ EI
HI HB HD AH AE DC CI FD BH BJ HG HB JA DC IC CB HA BF GH BE
HD DC GH BE BE CI BF GC IC HA FB JA DC CI ED FD AH JA AH HB
CD FB HG JA AH EI FB GH EB HD HA EA DC CI FD HD CD HG BE BE
CI BF GC IC EI BC JA JB CI FB JA GE BF HA BF CI JA IC CI BF
HA JA HI HB IC CA IF AH GJ CI AC FD GH CA HA JG FB HI IF DE
CI FD HB AJ EI IC FB DH EI CA CI IC HG DH DC AE EB GH HA BF
AJ CI JG AJ BE IC JA AJ CI FD GH FB AC HB CI FD AH GH AJ AH
EI BF GH BF AC BC FD GH HD JA AH EI FB GH JA HA EI FB JA IE
CD HA AC IC AJ CD GH AJ CB HG DH AJ JG JG JG
```

Now let's see which symbols appear together. In this chart we put an X if two symbols appear in the same digram.

	A	B	C	D	E	F	G	H	I	J
A			X		X			X		X
B			X		X	X		X		X
C	X	X		X			X		X	
D			X		X	X				
E	X	X		X			X		X	
F		X		X			X		X	
G			X		X	X		X		X
H	X	X					X		X	
I			X		X	X		X		X
J	X	X					X		X	

By studying this chart, we can divide the ciphertext symbols into two sets. For example, A and B do not appear in the same digram, but both can appear with C. So A and B are in the same set, and C is in the other. The two sets are {A, B, D, G, I} and {C, E, F, H, J}. To uncover the first plaintext, we reorder each digram so that a symbol from the first set comes before a symbol from the second set:

```
AJ DC IC AH BF BH AE AH DF GH AJ AH IE BF BC IE DF AJ DC IC
AH BF IJ IC BF AJ AH IE BF IE BC AC IH AE BE AH DH AH AJ IE
IH BH DH AH AE DC IC DF BH BJ GH BH AJ DC IC BC AH BF GH BE
DH DC GH BE BE IC BF GC IC AH BF AJ DC IC DE DF AH AJ AH BH
DC BF GH AJ AH IE BF GH BE DH AH AE DC IC DF DH DC GH BE BE
IC BF GC IC IE BC AJ BJ IC BF AJ GE BF AH BF IC AJ IC IC BF
```

```
AH AJ IH BH IC AC IF AH GJ IC AC DF GH AC AH GJ BF IH IF DE
IC DF BH AJ IE IC BF DH IE AC IC IC GH DH DC AE BE GH AH BF
AJ IC GJ AJ BE IC AJ AJ IC DF GH BF AC BH IC DF AH GH AJ AH
IE BF GH BF AC BC DF GH DH AJ AH IE BF GH AJ AH IE BF AJ IE
DC AH AC IC AJ DC GH AJ BC GH DH AJ GJ GJ GJ
```

We can now break this ciphertext as a Polybius cipher. To recover the second plaintext, we compare this to the original ciphertext. If we had to reorder a digram, assign a 1, otherwise a 0.

original:	AJ	CD	IC	HA	FB	HB	AE	HA	DF	HG	JA	HA	EI	FB	BC	IE	...
reordered:	AJ	DC	IC	AH	BF	BH	AE	AH	DF	GH	AJ	AH	IE	BF	BC	IE	...
bits:	0	1	0	1	1	1	0	1	0	1	1	1	1	1	0	0	. . .

The full bitstream, broken into groups of five, is

```
01011 10101 11110 01100 11110 01100 01011 10101 11100 01100
11100 11000 10000 10001 11011 10101 11110 11011 11011 11100
10001 01111 10010 11010 11110 10001 01011 11100 11101 01011
10100 01010 01100 01011 01011 10000 10000 01011 01101 11110
11000 10011 00111
```

When we try to decipher this as a Baconian cipher, we have a problem: code words like `11110` are not valid. However, if we reverse the rolls of 0 and 1, we can decode the bitstream. This simply means that we should swap the row and column labels that we found earlier. To finish decrypting, we must break what we get as a monoalphabetic substitution cipher.

We have another example of a duplicitous cipher. Before we present it, we should briefly describe the cipher that inspired it. The final challenge for the 2019 British National Cipher Challenge involved a modification of the bifid and trifid cipher to a fourth dimension. A 2×2×2×2 “grid” has 16 cells into which we can put letters; this is not enough. A 3×3×3×3 “grid” has 81; this is too many. As a compromise, the four-dimensional box used in the cipher has size 2×2×2×3. This allows 24 letters to be used. Since we need to remove two letters, we can merge `J` into `I` and `Z` into `S`. This should make the British happy, since they never use `Z`’s. The key for the cipher is a keyword for mixing the 24-letter alphabet in the box and a period for seriation and fractionation. Each plaintext letter is first encoded as the mixed-radix number describing its location in the box. A *mixed-radix number* is one whose digits have different bases. The base is often written as a subscript. For example, one such number used in this cipher is $1_2 0_2 1_2 2_3$ (or simply 1012). This number has three binary (base-2) digits and one ternary (base-3) digit.

Let’s encipher a short text to see in detail how the cipher works. Suppose we have the keyword `KEYWORD` and period 5, and this plaintext, which is padded to the fit the period:

```
THIS MESSAGE WAS ENCRYPTED WITH A NEW CIPHER XXX
```

The mixed alphabet is

KEYWORDABCFGHILMNPQSTUVX

The box is four-dimensional, so it is very difficult to draw it on this two-dimensional page, but we can instead list the contents and their coordinates:

K	0000	D	0100	H	1000	Q	1100
E	0001	A	0101	I	1001	S	1101
Y	0002	B	0102	L	1002	T	1102
W	0010	C	0110	M	1010	U	1110
O	0011	F	0111	N	1011	V	1111
R	0012	G	0112	P	1012	X	1112

We can now encode the plaintext as a fixed-width code using the four-digit mixed-radix numbers as code words.

```
1102 1000 1001 1101 1010 0001 1101 1101 0101 0112 0001 0010
0101 1101 0001 1011 0110 0012 0002 1012 1102 0001 0100 0010
1001 1102 1000 0101 1011 0001 0010 0110 1001 1012 1000 0001
0012 1112 1112 1112
```

Write each vertically, and group them in fives (the period):

```
11111 01100 00010 10001 10001 11010 00111 00111
10010 01111 00110 01000 10100 10100 01000 00111
00001 00001 01000 11101 00010 00010 11010 01111
20110 11112 10111 10222 21001 20111 00120 12222
```

Read off each group by rows:

```
11111 10010 00001 20110 01100 01111 00001 11112 00010 00110
01000 10111 10001 01000 11101 10222 10001 10100 00010 21001
11010 10100 00010 20111 00111 01000 11010 00120 00111 00111
01111 12222
```

Unlike the bifid and trifid ciphers, the box in this cipher does not have the same lengths in each dimension. Therefore we are unable to recombine the digits into letters. So the final ciphertext is

```
1111110010000012011001100011110000111112000100011001000101 1
1100010100011101102221000110100000102100111010101000001020 1
1100111010001101000120001110011101111122 22
```

To break this cipher, we first have to identify the period. This is easy if we break the ciphertext into blocks and adjust the block size until the 2’s only appear in every fourth block. Then we can reverse the transposition to the digits. We then decode the result using an unmixed alphabet. What remains can be broken as a monoalphabetic substitution cipher using the hill-climbing technique in Unit 36. The keyword can be recovered from the key to the monoalphabetic substitution.

Before we move on, we should note that the ternary digit need not be the last coordinate. In the BNCC challenge, the ternary digit was the third. Which coordinate uses the ternary digit should be clear during cryptanalysis, since 2’ can only appear there.

We are now ready to describe a second example of a duplicitous cipher. Like the first example, it encodes symbols from the first plaintext with coordinates, and then permutes them according to the contents of the second plaintext. Like the BNCC cipher described above, we the coordinates that we use are for a four-dimensional grid in which we place a mixed 24-letter alphabet. However, since we cannot rely on a positional number system (i.e., one that keeps digits in order), we have to expand the set of ciphertext symbols so that each coordinate has its own set; we did this for the first example cipher above, when we used ten row and column labels instead of five. Unlike the BNCC cipher, we will not use seriation and fractionation.

Let’s work through a short example to see how the cipher works. Start with these two plaintexts. One is padded, since they must have the same length (spaces don’t count):

```
#1:   DONT MAKE FUN OF ME
#2:   RIGHT BACK AT YOU X
```

The key consists of two keywords; suppose they are `SNICKER` and `LAUGH`. For labeling cells in the box, we can use these sets of symbols: {`A`, `B`}, {`C`, `D`}, {`E`, `F`}, {`G`, `H`, `I`}. Then encoding the first plaintext is done with this assignment, using the mixed alphabet obtained from the first keyword:

S	ACEG	R	ADEG	H	BCEG	T	BDEG
N	ACEH	A	ADEH	L	BCEH	U	BDEH
I	ACEI	B	ADEI	M	BCEI	V	BDEI
C	ACFG	D	ADFG	O	BCFG	W	BDFG
K	ACFH	F	ADFH	P	BCFH	X	BDFH
E	ACFI	G	ADFI	Q	BCFI	Y	BDFI

The first plaintext is provisionally encoded as

```
ADFG BCFG ACEH BDEG BCEI ADEH ACFH ACFI ADFH BDEH ACEH BCFG
ADFH BCEI ACFI
```

The second encryption scheme enumerates the permutations of four objects. Luckily, there are 24 of them. We use the second keyword to assign a mixed alphabet to them:

L	(0, 1, 2, 3)	C	(1, 0, 2, 3)	M	(2, 0, 1, 3)	S	(3, 0, 1, 2)
A	(0, 1, 3, 2)	D	(1, 0, 3, 2)	N	(2, 0, 3, 1)	T	(3, 0, 2, 1)
U	(0, 2, 1, 3)	E	(1, 2, 0, 3)	O	(2, 1, 0, 3)	V	(3, 1, 0, 2)
G	(0, 2, 3, 1)	F	(1, 2, 3, 0)	P	(2, 1, 3, 0)	W	(3, 1, 2, 0)
H	(0, 3, 1, 2)	I	(1, 3, 0, 2)	Q	(2, 3, 0, 1)	X	(3, 2, 0, 1)

B	(0, 3, 2, 1)	K	(1, 3, 2, 0)	R	(2, 3, 1, 0)	Y	(3, 2, 1, 0)

The second plaintext then becomes this list of permutations:

(2, 3, 1, 0), (1, 3, 0, 2), (0, 2, 3, 1), (0, 3, 1, 2), (3, 0, 2, 1), (0, 3, 2, 1), (0, 1, 3, 2), (1, 0, 2, 3), (1, 3, 2, 0), (0, 1, 3, 2), (3, 1, 0, 2), (3, 2, 1, 0), (2, 1, 0, 3), (0, 2, 1, 3), (3, 2, 0, 1)

We apply these permutations to the blocks of symbols from the provisional ciphertext of the first plaintext:

```
#1:   ADFG BCFG ACEH BDEG BCEI ADEH ACFH ACFI
#2:  (2, 3, 1, 0) (1, 3, 0, 2) (0, 2, 3, 1) (0, 3, 1, 2) (3, 0, 2, 1) (0, 3, 2, 1) (0, 1, 3, 2) (1, 0, 2, 3)
     ---------------------------------------------------------------------------------------------------
      GFAD FBGC AHCE BEGD CIEB AHED ACHF CAFI

#1:   ADFH BDEH ACEH BCFG ADFH BCEI ACFI
#2:  (1, 3, 2, 0) (0, 1, 3, 2) (3, 1, 0, 2) (3, 2, 1, 0) (2, 1, 0, 3) (0, 2, 1, 3) (3, 2, 0, 1)
     ---------------------------------------------------------------------------------------------
      HAFD BDHE ECHA GFCB FDAH BECI FICA
```

The final ciphertext is

```
GFADFBGCAHCEBEGDCIEBAHEDACHFCAFIHAFDBDHECHEAGFCBFDAHBECIFICA
```

Decipherment requires that one knows the alphabet key(s), the choice of symbols for the digits of the mixed-radix numbers, and which digit of those numbers is the ternary digit. However, this is not a particularly difficult cipher to cryptanalyze, as we shall see. First, we break the ciphertext into blocks of four symbols. Then, by noting which symbols do not occur in the same block as others, we can assign them to groups, each of which represents the first, second, third, or fourth digit of the mixed-radix numbers. We can then assign a mapping from those numbers to the 24-letter alphabet. The resulting text can be analyzed as a monoalphabetic substitution cipher, which can be broken with the hill-climbing technique in Unit 36. The orderings of the digit symbols (i.e., the permutations) can also be mapped to the 24-letter alphabet. The resulting text can again be analyzed as a monoalphabetic cipher. Finally, it is possible to recover the keywords (if any) that were used to reorder the key alphabets.

Let’s crack this ciphertext as an example:

```
ZTXSXTYRVSYXXZUTRWYTTSYWTUZXTWZUTZSXWTSZWTYSVRZXVUWZSTXZYTX
RSTYWRTZXWVZUVSZXYUVXTUXYWZVRTRYXUTYWTZUWYUTWTUYWZVXUTUXZRT
XYYTXRRTZXTSWYSYVXSZTXVZSWTXZSVUWYRTZXRXYTZRTXYSTWVRZWRTWYU
VYXWYTSTRXYYRXTZTXRWTYSTRYXTXUYTXZSTUXZTZWUTRYWTYUWXVSZWTYS
VUYWTRXZSTWYYTXRRTZXWTYUVUZWUVYXZTXRTYRXUZTXVSZXYTWRXTZRTUZ
XZVXSTRWYUTXYYTWUUVZWVSXZTWYRXYTRZTXRTYSWTYUXTWSYZTWRXTYUTS
YWYUVWXRTZSYVXTSYWVSXZTXYRVUWYYWRVVRWYYRWV
```

Our first step is to break the ciphertext into blocks of four symbols:

```
ZTXS XTYR VSYX XZUT RWYT TSYW TUZX TWZU TZSX WTSZ WTYS VRZX
VUWZ STXZ YTXR STYW RTZX WVZU VSZX YUVX TUXY WZVR TRYX UTYW
TZUW YUTW TUYW ZVXU TUXZ RTXY YTXR RTZX TSWY SYVX SZTX VZSW
TXZS VUWY RTZX RXYT ZRTX YSTW VRZW RTWY UVYX WYTS TRXY YRXT
ZTXR WTYS TRYX TXUY TXZS TUXZ TZWU TRYW TYUW XVSZ WTYS VUYW
TRXZ STWY YTXR RTZX WTYU VUZW UVYX ZTXR TYRX UZTX VSZX YTWR
XTZR TUZX ZVXS TRWY UTXY YTWU UVZW VSXZ TWYR XYTR ZTXR TYSW
TYUX TWSY ZTWR XTYU TSYW YUVW XRTZ SYVX TSYW VSXZ TXYR VUWY
YWRV VRWY YRWV
```

Then we tabulate the occurrences of symbols in the same block. In this grid we see an X when two symbols appear together in at least one four-symbol block.

	R	S	T	U	V	W	X	Y	Z
R			X		X	X	X	X	X
S			X		X	X	X	X	X
T	X	X		X		X	X	X	X
U			X		X	X	X	X	X
V	X	X		X		X	X	X	X
W	X	X	X	X	X			X	X
X	X	X	X	X	X			X	X
Y	X	X	X	X	X	X	X		
Z	X	X	X	X	X	X	X		

We can see that R, S, and U never appear in the same block. Therefore, they must be the symbols for the ternary digit. Likewise, T and V do not appear together, so must represent one of the binary digits. The complete set of sets is

{T, V}, {W, X}, {Y, Z}, {R, S, U}

We can now map combinations of these symbols to the 24-letter alphabet. The choice of mapping is irrelevant, since we will be analyzing the result as a monoalphabetic substitution later. So, without loss of generality, let us use this mapping:

TWYR → A	TXYR → G	VWYR → N	VXYR → T
TWYS → B	TXYS → H	VWYS → O	VXYS → U
TWYU → C	TXYU → I	VWYU → P	VXYU → V
TWZR → D	TXZR → K	VWZR → Q	VXZR → W
TWZS → E	TXZS → L	VWZS → R	VXZS → X
TWZU → F	TXZU → M	VWZU → S	VXZU → Y

With this mapping, we obtain this intermediate ciphertext:

```
LGUMABMFLEBWSLGBKSXVIQGCFCCYMGGKBULRLPKGKBQAVBGGKB
GILMFACXBPKBGKCSVKGMXAKMXAICSXAGKBIBDIBPKUBXGPNNN
```

Using our hill-climbing attack on the monoalphabe monoalphabetic substitution cipher recovers the first plaintext (clearly, the trailing XXX is padding):

```
IT MADE ALICE QUITE HUNGRY TO LOOK AT THEM. I WISH
THEY'D GET THE TRIAL DONE, SHE THOUGHT, AND HAND ROUND
THE REFRESHMENTS. XXX
```

The substitution key is M?EABDVKL?YFUXC?WIPGS?RNQ?, where '?' denotes unknown parts of the key because the plaintext does not contain B, J, P, V, or Z.

To decrypt the second plaintext, we need to identify the permutations used to reorder the symbols in each four-character block of the ciphertext. To that end, we need to order the sets of symbols, so we will say that they should be in the order {T, V}, {W, X}, {Y, Z}, {R, S, U}. The first block of the ciphertext is ZTXS, so its permutation is (1, 2, 0, 3), because it takes TXZS to ZTXS. The permutations for all blocks of the ciphertext are

(1, 2, 0, 3), (1, 0, 2, 3), (0, 3, 2, 1), (3, 0, 1, 2), (3, 1, 2, 0), (0, 3, 2, 1), (0, 3, 2, 1),
(0, 1, 2, 3), (0, 3, 1, 2), (1, 0, 3, 2), (1, 0, 2, 3), (0, 3, 2, 1), (0, 2, 3, 1), (1, 2, 3, 0),
(1, 2, 0, 3), (1, 3, 2, 0), (1, 3, 2, 0), (1, 0, 2, 3), (0, 3, 2, 1), (2, 3, 0, 1), (0, 2, 3, 1),
(2, 0, 1, 3), (0, 3, 2, 1), (1, 3, 2, 0), (0, 3, 1, 2), (2, 3, 0, 1), (0, 3, 2, 1), (1, 2, 0, 3),
(0, 2, 3, 1), (1, 2, 3, 0), (1, 2, 0, 3), (1, 3, 2, 0), (0, 2, 3, 1), (2, 3, 1, 0), (2, 3, 1, 0),
(0, 3, 1, 2), (0, 1, 2, 3), (0, 2, 3, 1), (1, 3, 2, 0), (3, 1, 2, 0), (2, 3, 0, 1), (2, 3, 0, 1),
(0, 3, 2, 1), (1, 2, 3, 0), (1, 3, 2, 0), (2, 0, 1, 3), (0, 2, 3, 1), (3, 2, 0, 1), (1, 2, 0, 3),
(1, 0, 2, 3), (0, 3, 2, 1), (0, 1, 3, 2), (0, 1, 2, 3), (0, 2, 3, 1), (0, 2, 1, 3), (0, 3, 2, 1),
(0, 3, 1, 2), (1, 0, 3, 2), (1, 0, 2, 3), (0, 3, 2, 1), (0, 2, 3, 1), (1, 2, 3, 0), (1, 2, 0, 3),
(1, 3, 2, 0), (1, 0, 2, 3), (0, 3, 2, 1), (1, 3, 2, 0), (1, 2, 0, 3), (0, 3, 1, 2), (2, 3, 1, 0),
(0, 3, 2, 1), (1, 2, 0, 3), (1, 0, 2, 3), (0, 3, 2, 1), (1, 2, 0, 3), (0, 2, 3, 1), (1, 2, 3, 0),
(1, 2, 0, 3), (1, 3, 2, 0), (0, 2, 3, 1), (0, 1, 2, 3), (2, 0, 1, 3), (1, 2, 0, 3), (0, 3, 1, 2),
(0, 3, 1, 2), (0, 1, 3, 2), (1, 2, 0, 3), (1, 0, 2, 3), (0, 3, 2, 1), (2, 3, 0, 1), (2, 0, 3, 1),
(2, 3, 1, 0), (0, 3, 2, 1), (0, 2, 3, 1), (0, 1, 2, 3), (0, 2, 3, 1), (3, 1, 0, 2), (0, 2, 3, 1),
(3, 2, 0, 1)

The mapping of these permutations to the 24-letter alphabet is arbitrary. Let us take a canonical one.

(0, 1, 2, 3) → A	(1, 0, 2, 3) → G	(2, 0, 1, 3) → N	(3, 0, 1, 2) → T
(0, 1, 3, 2) → B	(1, 0, 3, 2) → H	(2, 0, 3, 1) → O	(3, 0, 2, 1) → U
(0, 2, 1, 3) → C	(1, 2, 0, 3) → I	(2, 1, 0, 3) → P	(3, 1, 0, 2) → V
(0, 2, 3, 1) → D	(1, 2, 3, 0) → K	(2, 1, 3, 0) → Q	(3, 1, 2, 0) → W
(0, 3, 1, 2) → E	(1, 3, 0, 2) → L	(2, 3, 0, 1) → R	(3, 2, 0, 1) → X
(0, 3, 2, 1) → F	(1, 3, 2, 0) → M	(2, 3, 1, 0) → S	(3, 2, 1, 0) → Y

We get this intermediate ciphertext:

```
IGFTWFFAEHGFDKIMMGFRDNFMERFIDKIMDSSEADMWRRFKMNDXIGFBADCFEHG
FDKIMGFMIESFIGFIDKIMDANIEEBIGFROSFDADVDX
```

Analyzing it as a monoalphabetic cipher yields

```
THE QUEEN OF HEARTS SHE MADE SOME TARTS ALL ON A SUMMERS DAY.
THE KNAVE OF HEARTS HE STOLE THE TARTS AND TOOK THEM CLEAN AWAY.
```

with key `D?ONFH?G??BSRAE?TKMIWCV?X?`, where again '?' denotes unknown elements.

Having recovered both plaintexts, we could stop at this point. However, if we desire, we can try different ways of assigning the symbols R, S, ..., Z to the four mixed-radix digits and search for a recognizably keyed alphabet. It is easier to begin with the second plaintext. If we take the mapping of permutations to letters and apply the substitution key that we found above, we have

(0, 1, 2, 3) → N	(1, 0, 2, 3) → H	(2, 0, 1, 3) → D	(3, 0, 1, 2) → Q
(0, 1, 3, 2) → K	(1, 0, 3, 2) → F	(2, 0, 3, 1) → C	(3, 0, 2, 1) → ?
(0, 2, 1, 3) → V	(1, 2, 0, 3) → T	(2, 1, 0, 3) → ?	(3, 1, 0, 2) → W
(0, 2, 3, 1) → A	(1, 2, 3, 0) → R	(2, 1, 3, 0) → ?	(3, 1, 2, 0) → U
(0, 3, 1, 2) → O	(1, 3, 0, 2) → ?	(2, 3, 0, 1) → M	(3, 2, 0, 1) → Y
(0, 3, 2, 1) → E	(1, 3, 2, 0) → S	(2, 3, 1, 0) → L	(3, 2, 1, 0) → ?

By reassigning the symbols 1, 2, 3, and 4 we can try to find a mapping that corresponds to an easily recognized keyed alphabet. In this case, it is simple: we merely exchange 2 and 3.

(0, 1, 2, 3) → K	(1, 0, 2, 3) → F	(2, 0, 1, 3) → C	(3, 0, 1, 2) → ?
(0, 1, 3, 2) → N	(1, 0, 3, 2) → H	(2, 0, 3, 1) → D	(3, 0, 2, 1) → Q
(0, 2, 1, 3) → A	(1, 2, 0, 3) → R	(2, 1, 0, 3) → ?	(3, 1, 0, 2) → U
(0, 2, 3, 1) → V	(1, 2, 3, 0) → T	(2, 1, 3, 0) → ?	(3, 1, 2, 0) → W
(0, 3, 1, 2) → E	(1, 3, 0, 2) → S	(2, 3, 0, 1) → L	(3, 2, 0, 1) → ?
(0, 3, 2, 1) → O	(1, 3, 2, 0) → ?	(2, 3, 1, 0) → M	(3, 2, 1, 0) → Y

This gives us `KNAVEOFHRTS?CD??LM?QUW?Y`. The missing letters are obvious at this point, and the keyed alphabet is `KNAVEOFHRTSBCDGILMPQUWXY`, so that the keyword is `KNAVE OF HEARTS`. We also learn from this procedure that the ternary digit must come third in the mixed-radix numbers, rather than fourth as we had used earlier, since we had to swap the last two elements of each permutation.

We now turn to the task of recovering the keyed alphabet that was used on the first plaintext. If we apply the substitution key that we found, we get a new mapping:

TWYR → D	TXYR → T	VWYR → X	VXYR → ?
TWYS → E	TXYS → ?	VWYS → ?	VXYS → M
TWYU → O	TXYU → R	VWYU → S	VXYU → G
TWZR → F	TXZR → H	VWZR → Y	VXZR → Q
TWZS → C	TXZS → I	VWZS → W	VXZS → N
TWZU → L	TXZU → A	VWZU → U	VXZU → K

Since in recovering the second keyword we had to swap the last two elements of each permutation, we will do the same to each block above, and then reorder.

TWRY → D	TXRY → T	VWRY → X	VXRY → ?
TWRZ → F	TXRZ → H	VWRZ → Y	VXRZ → Q
TWSY → E	TXSY → ?	VWSY → ?	VXSY → M
TWSZ → C	TXSZ → I	VWSZ → W	VXSZ → N
TWUY → O	TXUY → R	VWUY → S	VXUY → G
TWUZ → L	TXUZ → A	VWUZ → U	VXUZ → K

We now look at various ways in which to assign the values of 0, 1, and 2 to the symbols representing digits. If we take T=0, V=1, X=0, W=1, U=0, S=1, R=2, Y=0, and Z=1, and reorganize the mapping accordingly, we get

TXUY → R	TWUY → O	VXUY → G	VWUY → S
TXUZ → A	TWUZ → L	VXUZ → K	VWUZ → U
TXSY → ?	TWSY → E	VXSY → M	VWSY → ?
TXSZ → I	TWSZ → C	VXSZ → N	VWSZ → W
TXRY → T	TWRY → D	VXRY → ?	VWRY → X
TXRZ → H	TWRZ → F	VXRZ → Q	VWRZ → Y

We see that the keyed alphabet is RA?ITHOLECDFGKMN?QSU?WXY. Again, the missing letters are obvious, and the full key is RABITHOLECDFGKMNPQSUVWXY, giving a keyword of RABBIT HOLE.

Finally, we would like to say that this cipher was dubbed the "MadHatter" cipher by a participant in the online forum for the BNCC. A ciphertext encrypted with the cipher was offered as a bonus challenge in the 2019 competition.

Reading and references

Merle E. Ohaver, "Solving Cipher Secrets," *Flynn's*, October 2 and November 13, 1926, toebes.com/Flynns/Flynns-19261002.htm, toebes.com/Flynns/pdf/Flynns-19261002.pdf, toebes.com/Flynns/Flynns-19261113.htm, toebes.com/Flynns/pdf/Flynns-19261113.pdf

Chris Christensen, "Lester Hill Revisited," *Cryptologia* 38:4 (2014) 293-332, DOI: 10.1080/01611194.2014.915260

Thomas Kaeding, "MadHatter: A toy cipher that conceals two plaintexts in the same ciphertext," Cryptology ePrint Archive, report 2020/301.

Grant A. Niblo, "The University of Southampton National Cipher Challenge," *Cryptologia* 28:3 (2004) 277-286. DOI: 10.1080/0161-110491892935. The current (or most recent) challenge is at www.cipherchallenge.org. The 2019 challenge is archived at 2019.cipherchallenge.org.

Programming tasks

1. Implement Levine's simple cipher. Try it on a some plaintexts of your own.

2. Implement the first example of a duplicitous cipher.

3. Implement the attack described above for the first example of a duplicitous cipher.

4. Implement the mixed-radix cipher from the BNCC.

5. Implement the attack on the fractionated mixed-radix cipher from the BNCC.

6. Implement the second example of a duplicitous cipher. Feel free to call functions you wrote for an earlier exercise, or to use class inheritance.

7. Implement the attack described above for the second example of a duplicitous cipher. Feel free to call functions you wrote for earlier exercises or to use class inheritance.

Exercises

1. Finish breaking the example of cryptanalyzing the second duplicitous cipher.

2. The following ciphertext was encrypted in a way similar to the first example of a duplicitous cipher. Break it and find both plaintexts. What are the keywords?

```
272515275805153890580751688591850745386172034985703851260 78
343701985158358477868302507194517858623275805853007890783 34
438361868907381560850387875083870525305185253425686883708 64
515831692278307255186520972516845250778301717257090512785 50
580305587885388786850985071568851985073427388925716825268 55
038610798036083611543035417037098830786549130712615278519 17
588649857130511903711316850530895871512785508503058578588 38
768853427389883708315175858190370177225902785175103498392 30
152583702507788368257172520709512785238683380534251572157 28
552506083831571512758508503055887583887865809850751868591 85
074372835085236171851538173054157203155127854530508527308 77
845307098850725835452079030231686866825328534725807238350 52
071715030792581572582505150583161785501771855178508387585 08
654152758505803055878858378868523250730866845432783895217 86
522685980371272507090307984372255086250709036083161532830 50
783830625582951323043072507900307986825922625070951725889 16
711503078930063849850386869085075186851958077883262570901 52
758250507387158174327855058512758450305857083514303701585 89
254958518368984583610386193817518549855045512725079090857 01
568589185070738433086863834198515383452512798503034477474 74
```

3. Encipher these two plaintexts with the same cipher and keys as in the previous exercise.

#1: CRYPTOGRAPHY IS
#2: FUN

4. This ciphertext was encrypted with a cipher similar to the second example above of a duplicitous cipher. Break it and find both plaintexts. Are there any keywords?

```
5869517371845237781448791846849686418956418661853426418764 8
9741886597258493669584936428731654187572384616895537247181 4
8748628247815727537285742856988752518758691357487117537285 3
6498946814748618247315698565371714862538641325741367531573 1
4186537274814618487231568659573141873157864159686985418757 2
3849741877325841751374861569386427482714365988174418648618 9
5648626895816461358527523763255287934648697148748964186498 6
4894286698565984826428771435186418765894879748147188427598 6
7315417836527815614869846985841664982436847184177234184648 7
2615369853517487172856958351741873275958679487285849748712 5
7341874396498642867482471368245986486189648641643134694869 5
3176428258772847235841749868956864274287431586975314718628 4
3527418724366581875247896498782574824862658946187284351665 9
8614842365862648284723147752869534871237564286589896451377 1
4861485136653984715237739469438427641328575236157852877325 6
9858416518663244187185758727253698535174187356178255237536 2
7285469386497148859671537285364931473157895616483642648234 6
1487172858427423673944168498642866985649864819846342687414 7
8134276248849649363427437257286984781486425986561372857418 4
6892487698562358426428763518956153781745689486186154236714 8
7498864198464861624364817235471378145287487281748651643985 6
9586951737184598673155372814781477948841752377148486136424 6
2361347523852749876315714851368956527814377481895651377148 7
2531846598678254287487131468257735248727814723575285178628 4
8596428772585372827552875623782452875372827575286532895617 5
3874147936413352764287841513687526934695889741753471878245 9
8653718174437964894618748964186234628434164187631452875623 8
6248742743185697153874157237418648972343246874187245986439 7
5287815775237582857236526489846162846513486164326428613484 1
7428678246351428673147248641874826958462373944186487231658 6
4915734187486984616895513658727984827486427482461862346418 3
5727413487164815723235643613715418684263752816458697582472 8
7148418742871356852758716985351748716859684151866432714 8
```

5. Encipher these two plaintexts with the same cipher and keys as in the previous exercise.

#1: WELCOME TO
#2: MY TEA PARTY

6. This is the final challenge from the 2019 British National Cipher Challenge. Break it. What is the keyword?

```
11101 00101 00001 21122 10001 10111 00101 10001 01111 00210
11011 11111 01011 01001 01011 21001 11000 01011 01000 00012
21222 10011 11010 01000 01001 10201 00000 11101 10010 01011
00012 00212 00011 01000 01101 01110 00000 20021 10010 00100
11010 11011 00002 10000 10000 11010 00001 00022 00110 00101
10101 10001 01100 00210 01111 10010 01001 00000 10111 02020
00101 00101 01001 11000 01022 21210 00001 00110 10101 00010
21020 02111 00100 00110 11101 10000 21012 01110 11001 10011
00000 10021 10011 01101 01100 00011 00002 10011 11100 10010
01001 00101 10020 00021 00001 01010 01010 01001 10002 21011
00011 10111 10000 00120 12102 01001 10001 10010 10110 01021
02011 01111 00101 00000 10001 10022 20000 00100 01010 01100
01110 21021 00101 00100 11101 10000 21002 12111 10100 01100
01011 10002 01202 01010 11001 00010 01111 00011 00101 00010
11110 10000 00012 01122 00111 00001 11110 10000 00021 12210
01001 10100 10100 10010 22002 00001 10000 01101 11101 00100
21001 01000 11110 10000 00001 01202 20200 00011 10110 01000
01002 22100 21101 11000 00100 01110 00200 12011 00000 11010
10010 00111 10221 00100 01010 11110 10001 00102 21220 10000
00110 10100 10000 02100 22101 00000 11010 10001 01012 00200
00100 00001 11101 10100 01021 10001 00010 11000 11010 01101
02111 10001 01110 11001 00000 01002 11002 10110 11110 10011
11011 01000 00210 00010 11101 00110 00001 22221 20000 11010
00110 11100 00100 02210 01101 10000 01100 01011 11122 10001
01101 01000 00110 10021 10021 10101 11100 11101 10110 01021
20010 10010 01110 11101 00100 21220 01010 00011 00010 00100
11010 12201 00100 10011 10000 11000 21112 00111 00011 10010
00100 00001 20010 11101 00011 10000 11001 01112 00211 10011
01011 01011 10001 10200 21100 10000 11010 00001 00020 20010
01111 00111 11001 11011 20112 10101 00111 11010 00100 01001
12202 11100 10100 00100 01001 10002 11010 11110 11010 01100
00002 00202 00100 00110 01011 01000 01200 02111 11101 01001
10000 00100 20001 20001 11101 00011 10000 11001 01112 10011
10000 01110 10010 10021 10101 11111 11001 01000 01011 11002
11110 11110 10101 10110 00021 02220 11100 11010 11000 01000
00202 01100 00100 11011 00111 10021 00200 11000 10100 11000
00001 12020 20100 01110 11001 01100 01011 11002 11110 11111
11011 10101 01001 10220 11100 01010 00100 01001 02002 11101
1111
```

7. This is the bonus challenge from the 2019 BNCC. Break it and recover both plaintexts. What are the keywords? (This ciphertext used the *inverse* permutations, so you will have to adjust accordingly when you look for the second keyword.)

```
JADGGBJDEJGAJGEBJDHADLHBGJDADLGBDBJHJADHDAGKAJEGDKAHAHKDAJE
GAKHDKAGDALEGDKBHDKGAHJDAEAGLLADHHJDADLGAELHBELGAEAGJDGLAHA
JDJEHABHKDJBEHGBLDEAGJALEGEAGLGJEAHDAKGKDABDJHLBHEBKDGDJAHL
ADGGBLDEJGABDLGLAEGHAJDDLGAHKDBDAKHALEGDKGBDHJABHDJDAGJAKDH
EKAGDAGJADHKAHJEDHKBEJHBBGDKAJDGDLHAJEGAKDHAGAKDAHKDDHKBGAL
```

EKDGBEAGJDAKHKDGAGALEBHKDEHJAHBKDBHKDAKDHEBGJHDAJBDKHGDALEL
GABLDHJAEGEGBJDJHADGLBDBGKJAHDDBKGDJGAAEJHAKEHDJHAADJHHAEKJ
ADHHAJEEJGAHDAKGALEDBHKELAGDKBGJAHDHDBJDKHBHDBKEGAKGKDBADJH
AGLDDLBGEJGAGLDBELGADJAHADLGBLDHGAJDBGLDDALGDHJAADJGAEHJAEG
JDAHKGKDADBHJBLEGELAGJAEGGALEDKGBADJGJAEHHAKDDKHBHKEAEJAGEJ
GBELGABGLEDALGJAHDGDLBHDKBADLGEBJGDKHBKADHHALDDJHALGDALDBGD
JGAELGAEAGJDHKBHAKDBLGDAEJGKADHJAEGLEGADJGADKHAEJAHBDHLDKGB
GJDAAELGJAGEBLDGELAGDBGKJADHLBEGGBLDDJHADHBKEBHJDGJABHDJBDK
HHBDKKAEGGALEDKGBDHKBBELGADGKBDGKEAGLGJDADBJGGAJEEJGBDAJHDB
HLAGJEELAGKBDGBDKHGBLEELGAHKEAEAJGEJGBAEGLGBELLADGHAJDDLGBK
HDBDALGJBEGBDKHADKHDLAHDAJHADLGDBLGJADGAGLEAEGJBDHKKADHGBLD
DBGLKBDHDBLGGBKDDJHAHLBDJAGDBGLDJBEGDKHBHADKBDLGGJEAAJHEADJ
HLADGJAEGHAKDDKGAJGEAKDHADKGBHJDADLGADBHKBDHLAHKDKEBGEJGADA
HKAJEHDJAGDBLGHBLDDJHAJDGBDBGJDAJGBGLDLBGDDBKGJADHBJGEBDKHB
GLEBJDGAJHEHJEBBDKHLAGDALEGDKBGJADHBDKGDBKHELGAEJHAAGJDHBEL
KBEGGAJDEJHAJDHABDKGHJDADLAGJBEHJDHADJHADJGBDLAHADHJDLGALEH
BDBGLDBJGJDHAHJDAAHKELBEHJADGDKHAAJEHDBGLLAEGLBEGHAKEEJGAEH
AJDBHLDGKBAHDJAELGGBDKDJHADAGLEGLAGBKDDHJAAEKHBDGJADHJDAGJG
LDBLEGBGALDDJHABKDHEBJHGBKEDJGAKEGAEAGJDAKHDAKGJADGEAHJGJDA
JAEGBDGLELHBEJBGDGBKJGDAEAGJKDAHBDLHBHKDEGLBDJGBEHJABDJHHDA
JBHLDBKDHALGDGAELGKDBAELGGDBKJADHGALEDLGABDKHBLEGDJHBGDBJDA
HJHKDBBHEJDKGADJAHELAGLAGEGEAJAHKDGADKLGEBAEKHJAGDKADHHAJEA
HKEAJEGBGEJKEGAEJGAHKDADKGALEGADBGKJHADJEHAADJGGJEABGDLEAJG
HAJDDLGBDBLHDBGKAHJDDKAHLGDBBELGJAHEAEJHJAHDAKHDGBJDLBEHJAD
GBDLHGDKBEAGJLGEAJHDAADLGJAGDBHDJBJDHAEJGLEGALDHBEAGJLGAEKD
GBAEKHGJEAAHDKEAKGJDHAELHBHJDADLGBALDGJADGHDAKEBGJDJBGDKHBL
BDGADJHDHBJELHBBDGKADJHADLGAGELDKGBDJAHDAGLJADHHBDLADJGGLDB
AKDHBDKHLAEGGBKDAGJEAKDHAKGDBLDGBDKHHALDADJHLDGAEHLBADLGAJD
HKBEGGJDADAGLGKEAAGJDJDHBBGJDDAJHEJGADAHKELAGDKBGDJGAELAGKA
DHHBKDDLGAAEJHAJEGEJHAGDAJDBGJGJEABGEJDJHAELAGDKBGEJGADKHAA
GKEGAEJLGEABDLGKBHDLADHHAJDAGLDBLEHBGEKLEGBEJGBGKDBDKHBEBGL
AEGLBHKDJEBHELGAKGDBADJHLBHDADJGLBHEALGEHBKDKBDGJADHADJGDAL
GGELADKGBDAHJAGDLAGJDHDBJDBJHJAGEGLEADLGBAGJDHEBLLEAGDKHBGJ
EALGEADBLGADJHGJDBHJEBKHDBKDGBEAHJDAJHDAGJADGLBDHKDBGKAEHJA
HJDDGJADLGAEGJABDLGKBGDADGJDJBGDBJGDBHJJHDAGAELBDKHHKDBJBDG
GAJDELGAJDHABDLHGKDBDJAHDAKHBDLGGBDKJADHGALEDKGBBDKHBEGLDKA
GDBKGAELGAJEGALGEBHDKDLHAHDJALADGGJDABHJEALEGAJHDAGLDBLDHAJ
GDLADGHAJEGLDBAEJGLAGEBKHDGBJEGBJEELGBGLDAGALDDJHAEAHJELAGB
DHKBGKDAHDJDLGALAEGGBKDDJGALAEGBGLDDGKBAHDJBHKDBLEGAKGDBDKG
GAELEAGLDHKBBDGKADJGDHLAAHDJDBHLBKDHDKAHEAJHHAJDDLGADHJAAEJ
HJAGDAEGLELAGGBKDEAGJGLDBDJBHBEGLLGEAGAJDELGAAEGLGDKBDAJHEA
GLJGEABGKEAJDHAJGELAEGGAJDDJGBGDBJBDLGJAHDJADHBEGKADJHJAHEE
KBHLBEGJEGAELGADJHADKAHADGJELGAELBGLAGDGDAJDBGJBJDHELBGLAEG
LDHBDKGBDJHADKAHAEGLDKGBDJAHDLGADJGADBHJBJHDGJEAELGADJGAEJB
GLAEGBELGGAJDDBGJJDBGELHBEAGLDHBKDBHKKAEGJADGBDHLDJGAGAELBE
JGGKDBKBDHLEGBELGADKHBEJBHAEGJELGADLBGLBHDGDAJEAJGDBGLAGLEE
JBGKBDHJDGAELGAEKHADKBHBEGJEKGADJAHELGADJGAHAKDEAHJDJBGDBKH
HDKBAEKGAHJDAEHJDJGAELAGGAJEAGLEDAJGAKDHAHJEEGLADKGBAJDHDKH
AKDGBGLEBDAGLLADGJAEGADHJEJHABHKDKAHDDAJGJBDGJAEGGBJEDJHABL

DGGAELADGJDLGALEGADJAHAEJHLEGABKDHDBGKHDAJADLGBJEHDAJHJADHG
LEABEJHKBHDDALGEGJAGLEABEJHJDGBDAGJDBLGDGBKADHJEAHJDGJABGEJ
DAGLDBKHDBGLLBDGDBGKHJDADGALEBGKGJEAHKDAEJHAAEGLKDGBGAJDAGL
EDGLBGBKDHADJGKDBAGJDAJHEHAKDJADHDLHAAJDHDLGADJBHDHAJBEJHKB
DHGALDJADHGBLDDJHAHAJDDKHAGJDAADLGDJGADBJHDBJHAEJGLAEGHBLDE
JGALGEAGKBDJAHDAGJELAEGDKGBHADJADLGGJDABLHDADJGJAEGLBDGGALE
EJGBBHKDGJDAGLAEKAHEDBHKGJEBGKEADJHAAEGLBHKDAGDLAJDGBDLHJDG
ALEGAGBJEDBGKLAEGKHDBAELGJAGDGKEADJHAKBDHGBLEAGLEBKDHBHEJAG
JEALEGAJGDDAHKEHAJBDJHLBGELADGDAKHEAGJAHKDDKAGLBDHJEGAELGAD
KGBEJBGBEGLDLGAEJAGKBHDGDBLEGAJLAGELBEHBGLDKBDGAHJDDALGAJDG
KADHDAJGBEJGALDGBDKHGBLDBDLGLEGADGKBADJHBJEHJAEGHJDADBGJHJE
AAGJDJEHBAGLEDAJHDLGAEGJAAELGAJDGKADHAHJEBLDHJAGDDLGBEJAGLB
EGLDGBELGAEJGADKAHAEGLEJGAEKBGDJHAELGABHKDGBDLAHJDJADHJEGAE
AGLEAKHDBKHAEKHAHJEKBDHHDBLAHKDAJGDGBJDJADGDGALDKGADAHJALDG
DJAGBHJDHBDJAGJELAEGKDGBDKHBGJDBDJHAEBLGADKHAHJEJADHLDGAAEL
GKBGDADHJDKBGDHAJEJHADGAKJAHDJAEGDAHKDAJGAHKDKBDHAGLEDBKGAJ
DHLADGBGKEKBDHEBGKAHJDAKDHLAEGJAEHHBKDDBHLKADHDBLHADHJADKHA
LGEAGJDDGJBAGJEJEGBDAHJDAJGJBEHGALEAHJDALDGAGEJAGLEAKDHAJHD
DAHLADHJAGLDDHKBDKHAEBJGDAHJBGJEDKBHKDAHDLGBGJEAJHEAHDJAADL
GAEGJAHKDKADGDKGBBDHKLHDBJEGAHAKDAGLEDGKBHAJDBHLDDBKHLADGGB
JDEJHADGBLDKGBDAJHLBDHJDGADBGLLGEABDKHKAGDJADHAELGHBDKLBEGG
ALEAGJDDAKGJDGAAEJGADKHELAH

Challenge 1

1000001100223300101101012120110000001000223200001001012120 1
0000000011122123001002112001222010002122012312000012000122 3
2211001110010103200000000003202301000020202022200001000023 0
1200001120011220120000020000002230000011120123000001000112
2312000001000010223010002120012302000000020223122000000000
2302200111001113220100000000000223310000100002320200000000
0230120000000100300010000000011230020000000000320120000000 1
0030001000001000020123000100101200010000100001022012000000 0
2003230301100120023112010000120002101200001001113001200011 2
0010221100100000111032020101012000312031110012122212110110 0
0000123033110012002111202000001111210231010011220020223011 0
1011200121211100121022212001000010010123000100111012333201 0
1001012120211100100200101230011110001230320000011020112020 0
0110010030102000100121202021101000011000223000001100021012 0
0000001113001200000122223033 3

Challenge 2

AFKGOWTPY472DKCGMTXQ1805DCKHNTVRY852FALHNTVPY186HDCKSNXP720
5GEAKNQTVY681EKHBMSPX81Y6HFCJROTV248YCDJHPVMTY681FGKAOTWP53
Y7DJBGOQVT2608BGJEXSNR9Y16BJGDWQOT27Z4DBHKVUMP168YKECGNVTPY
681ECGKSPMW5Y27JFAITROVY481FKAINPWT14Z8AGKEWMPS0762LDBHPVNT

1Z59GFAKNUWPY274GBKFMXQS2074KEAINWUP167YBJDHNUVQ7Z24BDJHMWU
PZ482CJDHPOTV5Z29GBJEVROT61Y8CHKEWPNU274YDIJBMQXU51Z8DHKCNV
RTY671GFBKNPVU4Y72ALEHQTVM1Y67GFBKTNPV159YBJDHOTWR24Y7FAHKV
TNQ167YFKBGPNTV347YAFHLNWPT7Y24EBGJNPVT24Z9AHLFTNPV167ZKEBG
PMVT34Z8DAGKWSMP0681JFIAOSWPY836DBGKUMQW61Y7JEAHWNRS25Y7AFK
ISPMW167ZDBGLTPMV6Z38KGBDPUVNZ482BIJDPTNW724YAHLFTMQWY274GA
FJTQOV91Y4FHJAXOSR5207JFAIVRTOY716JFBGVQUN259ZKFAIPWMU2067J
AHENTXR2Y47JAFHOPXTY752CGLERNWS961ZEGKBWPUM0186AJHDPSNVY274
IDBKTNVP1Y48JHCDQTVO0176BDHJVNUQ248ZBJDINTWPY742AIKFWPSN106
8JHCDSPMW83Y6DGKBMVQTY276HFALTMWP347YJDBHTMPW72Y4DBLHPXSN15
8YEIJAWPNT247YCKGDQXUM4Z27IBJENUWQ72Z5KBDGVMQT169YBJDHOWSP9
1Z6EKHBSMWQ0716EHJAOVQUZ472JGCDUQOW72Y5JDGBPTVN91Z6FGJANVQU
Z592GDBJOTXPY284HCKDMWPS1Z96JDBGNXSR3Z48EKBHNPWU35Y7BGJEPMT
V924YFCHKVTPM2Z47IDBJSOQV2Z95BGJFTOVP86Y1KDCHPVNTY671FJBGQO
SV259ZBFGKMWPSZ671LGBDPTVMZ836BGKDTRMW724YEAGKXSNP1690BKDGM
TWR248YADJHPWOSZ582JDHANVPTY671HFAKWPSMY681FAHKOTRVY418AJID
OUVQZ247AFKGPOSW357ZAGJDUNPX247ZAKFGOTWP24Y7CDKGPUWN257YDGB
LMTVP29Z6EAHLQTWM0247AKFIPOSX41Z8GAEJPTVMY924KEBGPVTM4Z72AJ
DIPTWN824YAFJIQUWMY725BGDKMVUQ71Z4LBDGVPMS196ZGFAKTMPV0752B
GKDVUMQZ427BGJEMPSV159ZJDBHWOPS168ZCKDHMSWP38Z4BGJDSQNX916Y
BJEGWPNTZ274GFAJTQVOY572HDCJRSXNZ481GCEKQMTW5207GBDLNTPWY48
2AKGDWNPUY691BDGJPSXOZ472HADJORWT247YGKFATWPO581YGKECTXPM48
2YHJDCSXPM4720

Unit 165
Combination-lock cipher

The *combination-lock cipher* was introduced in the 2020 special coronavirus edition of the British National Cipher Challenge. It is a fractionating cipher modeled after a combination lock with a number of numbered wheels (think about bicycle locks, not safes). Letters in the plaintext are each expressed as three trits (base-3 digits). Since there are 27 characters possible with three trits, spaces are included as character zero (in ciphertexts they are written as @). The most significant trits (MST) (the first digit of a three-digit base-3 number, including leading zeroes) are shifted some number of characters to the right, with wrap-around to the beginning of the text. The least significant trits (LST) are shifted by a different number, and the trits in the middle are shifted by a third number. Hence, the key is a set of three numbers that represent the shifts. Each set of shifted trits works in analogy to one of the wheels on the combination lock. After the shifts, trits are converted back into characters.

As usual, let's work a short example for clarification. Suppose our key is (2, 5, 7), and that our (very short) message is

```
COMBINATION LOCK
```

First, we decompose each character into trits. To help with that, here is a table of the plaintext symbols and their trits:

```
     _ 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
     ---------------------------------------------------
MST: 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2
     0 0 0 1 1 1 2 2 2 0 0 0 1 1 1 2 2 2 0 0 0 1 1 1 2 2 2
LST: 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2
```

The decomposition of the message is:

```
C O M B I N A T I O N _ L O C K
-------------------------------
0 1 1 0 1 1 0 2 1 1 1 0 1 1 0 1
1 2 1 0 0 1 0 0 0 2 1 0 1 2 1 0
0 0 1 2 0 2 1 2 0 0 2 0 0 0 0 2
```

Now we execute the shifts. Trits that roll off the end are brought back to the beginning.

```
2→  0 1 0 1 1 0 1 1 0 2 1 1 1 0 1 1
5→  0 1 2 1 0 1 2 1 0 0 1 0 0 0 2 1
7→  0 2 0 0 0 0 2 0 0 1 2 0 2 1 2 0
```

Trits are converted back into letters to obtain the ciphertext:

```
0 1 0 1 1 0 1 1 0 2 1 1 1 0 1 1
0 1 2 1 0 1 2 1 0 0 1 0 0 0 2 1
0 2 0 0 0 0 2 0 0 1 2 0 2 1 2 0
-------------------------------
@ N F L I C Q L @ S N I K A Q L
```

An automated attack on this cipher that is better than brute force works as follows. We fix the first number in the key to 0. We vary the second key number from 0 to the length of the ciphertext. For each choice of second key number, we decipher the text and calculate the index of coincidence of the plaintext. For those key numbers that give an IoC in the top 10%, we also vary the third key number and again calculate the IoC of the resulting plaintext. The best overall IoC wins. Unfortunately, it is difficult to know where the beginning of the plaintext is located, so there is an overall shift of all three key numbers that will not be found by this algorithm and must be found by hand afterward.

A participant in the BNCC suggested an extension of the combination-lock cipher in which two or more characters are grouped together. During encipherment, the number of wheels is three times the number of characters in each group. With more wheels, it is possible also to represent the key as a keyword.

Let's rework our example with two characters per group, and use the keyword `CIPHER`. The plaintext has an even number of characters, so we do not need to pad it with `X` or `_`. First, break the groups into trits:

```
C M I A I N L C
O B N T O _ O K
---------------
0 1 1 0 1 1 1 0
1 1 0 0 0 1 1 1
0 1 0 1 0 2 0 0

1 0 1 2 1 0 1 1
2 0 1 0 2 0 2 0
0 2 2 2 0 0 0 2
```

The shifts are (since space is zero) `C` = 3, `I` = 9, `P` = 16, `H` = 8, `E` = 5, `R` = 18. Some of these are longer than the number of groups, so we must take them modulo that number.

```
 3→  1 1 0 0 1 1 0 1
 9→  1 1 1 0 0 0 1 1
16→  0 1 0 1 0 2 0 0

 8→  1 0 1 2 1 0 1 1
 5→  0 2 0 2 0 2 0 1
18→  0 2 0 2 2 2 0 0
     -----------------
     L M C A I K C L
```

```
I H I Z K H I L
```

The ciphertext is

```
LMCAIKCLIHIZKHIL
```

To break the extended cipher, we can partition the ciphertext into sets that are enciphered with the same key using the original cipher. If the characters were grouped into twos, then every other character forms a partition that was enciphered with the original cipher and three key numbers. The remaining characters form the other partition; they were enciphered with another set of three key numbers. We can then break each partition separately, then inteleave the resulting plaintexts to get the complete plaintext. Since each partition may have a different overall shift, we need to adjust them relative to each other until the textual fitness is acceptable.

Programming tasks

1. Implement the original combination-lock cipher.

2. Implement the attack described above for the original cipher.

3. Implement the extended cipher.

4. Implement the attack for the extended cipher.

Exercises

1. Encipher this text with key (9, 22, 3).

   > It was a very ill time to be sick in, for if any one complained, it was immediately said he had the plague; and though I had indeed no symptom of that distemper, yet being very ill, both in my head and in my stomach, I was not without apprehension that I really was infected.
   >
   > (from *A Journal of the Plague Year* by Daniel Defoe)

2. Decipher this ciphertext with key (15, 7, 11).

```
JIDJGFH@@IL@TRTJYCSBRFE@KWZ@WEACHBFQJILO@DNAETAKTFIDCUBTKZR
G@OHVIRRX@@KTSBUFCYAIHDIKNB@RHGBUFRUDJFHKXDKBIFCAIWTSCAF@W@
CIJFKV@JLGUS@CIRGKRTTGERHETFVCADIK@BVACREWCRBZSRMAACBFFNLBL
L@GKMO@IHERBTHLXBBTGRAASCRBRBJYCZFR@AZNUA@SBUJP@GU@I@ACXKP@
PU@KTHIYC@RXD@WNN@UBAANHUGJRHDLJOGHUJICGALQBUCTCKB@HUHX@@GZ
NUB@YB@QM@BBTCR@EORHWVUI@ZBHIFCRBTBISCASRAU@K@BJYCZFR@GTQU@
CRDCRJKG@GL@KBHIYC@UXG@UJJF@SIRURBRDJWEKCZS@Z@HCILXDCWNKBRB
CIEC@JVB@BLE@AXRCDMIBBAU@B@EPVCR@VBFIQHCLKUL@AQ
```

3. Break this ciphertext that was encrypted with the original cipher.

```
SFI@DWDBHJRJO@CACNKRDIHBMWCQDAKUEBRSCJTDFNR@DCRC@KXUSK@FJTH
CTAAEIKQZT@YWSFJIEUW@ZBW@CLQAOKRFXSFRFCS@@HR@DD@LB@HJFF@JLC
ERXJA@OT@TALAIH@HXHBEC@KB@KD@BSCD@OZCET@RBOTBKO@VDIXDLWJJCN
KHQLUIBVCWAQVNI@MNEJORLCTBEVCA@NUBBTC@CS@FNRARVFNICTDIBH@SF
A@ECIOKV@A@W@BFBQBILM@F@VT@ANJRB@C@LSBXCBSHCRDCJ@RBXBRU@CH@
LOIJQLIZ@U@WZAFAISKVCD@EXDFBLR@BSCB@HWWROKA@HWQ@JIVACUL@HTB
CDORNOE@EIYDTI@OLR@@LOCBTECAIIC@N@AHUBA@CLUKRHECCIRHAUAEU@U
EDC@RI
```

4. Encipher this text with the extended cipher and keyword `GARDEN`. Feel free to use `X` to mark the end of a sentence, and `XXX` to mark the end of the message. All other punctuation should be discarded.

   Every afternoon, as they were coming from school, the children used to go and play in the Giant's garden. It was a large lovely garden, with soft green grass. Here and there over the grass stood beautiful flowers like stars, and there were twelve peach-trees that in the spring-time broke out into delicate blossoms of pink and pearl, and in the autumn bore rich fruit. The birds sat on the trees and sang so sweetly that the children used to stop their games in order to listen to them.

   (from "The Selfish Giant" by Oscar Wilde)

5. Decipher this ciphertext with the extended cipher and keyword `VEGANS`.

```
@RDLLB@@QLFRCUFWTFLKIOBFIXAZJVN@FJBLO@IK@FUJJGECEARGDOCCHMS
CSXEJHIZC@QO@CUFCQTSLI@RZCNLSRLECSRHGFAC@KFHILR@XSTHCURI@BH
XSOBTBCU@JQEFIIVETIBKZLSCWIBCIFUNA@DTG@XIBGXKU@D@R@JISPPRAO
KDGAYOSFLAXBRABVDJGBGOE@@E@BJNPUKKCJLK@@AGHRAUKTKH@DNL@BDXF
@AVQSCL@WRTIERFEB@BLIASQWBRXNLORANH@MWRRH@UHKE@@FJCBHHJEACE
INXELATYBLL@EEK@EHIEBKWDAYUAIWADKCTJBHAFMP@GR
```

6. Break this ciphertext that was encrypted with the extended cipher. What is the keyword?

```
BCHD@ETTW@BNOATNU@@Q@KXKCOBTOWDTL@SCHKLXAFQVSFC@CLEJFAABCOR
ZWHKAD@YFHCCDIRKTCW@IISS@UBFZTWCB@FUEICXA@FIRWTTLW@FXAIEHD@
QBNITCA@SURL@IFSUCBFZLACRKNIQC@USRME@VZVHLBZSUAD@TB@FU@DOSD
DSBZR@TYC@@CO@EWKTC@AGEKUUBRSBIC@EXA@EBI@XE@EBWTCKAMDDUYEKU
R@@BTFDEEXARGCBWBU@@KZ@UULAQKTXIBBEORUDCZ@ZO@BYMENRFJHCNCAX
AACMBCX@IKSIRC@BSINR@HUXBIELJODNSUTL@FRCRITRIDRLH@K@BM@@P@D
KD@@XCDRGB@USJP@ASLBDE@@IFICDSLGCJIT@USAJHEWRBTBBD@MDKEBBFC
BO@AZCIOBFAJFCIIIUT@JISI@@RH@UIJLXRZK@VEGFHCKHIARDECURZAS@U
CWAFCE@NZTTSVDCISEU@WBM@BXF@NT@@@D@SACQBSQLEBRI@QKLFHQTBDCS
@RJD@KXALITAK@FGHPYKOCRCICAPEAL@DG@E@WFIRSK@RHCEFKHLRLI@RAW
AAINGBLNDX@@KAXCCMLDL@HGRK@GFIWDTRUYT@FDEEBGJ@DPJWIVIIACICU
TEIRCAIDTL@AE@TICCATRKHCYYGDNCWT@YFR@ANUCTTJ@I@VUCCC@ARJCCC
IY@SIC@KRT@B@KR@L@AGJ@CNWBF@KEIZKEEIJZNABCYC@JMDXRFATBKLC@@
TDLTUBMTVCM@@ICUWMARIN@SUCEESRXJNLAVJ@YIE@BYBITTALNEGRGUAJ@
CTCACCVBFSTIRRS@XRAW@BHFBFEKCBBZTBUKODEEIHENCFC@XLBETICEHJS
```

```
LWSRTKCFTVYEHWIKFAW@TFAOCSFCIQ@BCBJCHK@TVKLSRDDAWBNZYT@QEER
BDTTBC@QKLE@AXHC
```

7. I seem to have made a mistake, and the following ciphertext was encrypted with a broken cipher that only has only one wheel. Can you break it?

```
GSRFTUDURFTRORFTSXAR@OARDQOOSXCAXICDRFTSRFLZI@MSIGDSJX@FTSR
GROSX@@MSICDRFTSREM@MXXIGTUABRBOEGRH@@YFRFTSRAEGDSCD@EB@FJB
WD@OFIBROIFILAIFTRORFTSXAR@OARDW@DQOOSXCAXI@MSIFTSRAEGDSCD@
EB@FJBWD@OFICVWGXI@MD@FTUDTOIH@@MSIGSRDARODSSIDPABRBFEDIEGF
@FSRJBSXFIH@GSRDARODSSIFTRORFTSRAROFTRCFIBD@OR@MSIFTRORFTSR
FPDRFAYNDGJXI@OEGDSICFRGTUABRA@PFJXIFTSRAIQI@MSIFTSRDUBKXRH
@GSRDARODSSIFTSRDRMJXIEB@@MD@FTSRGUDSOIGTUABR@VEGREGJX@FTSR
FJROI@MSIEPFKRFTSRFILDFIEB@EGF@BOARORFKUEOIH@GSRDARODSSIBWG
RFW@@VASSIDJVRFW@AGFARFTSVIEB@@MD@@LDDJVXXIHHH
```

Challenge 1

```
CY__ALHYBXLDBJQGTQWASQUWBCYADMTQFAD_Y_XGAFDXWPTELAROTCIN_MI
WFI,OBKIBH_AMCHAQDHE_JDFBLKDQFEUKUJPDGADODVDQXBIQ_LIPEYFFRR
K_XSA,AQB'I_TNW_PIPNG'NTB_UQQB'X_D!G__._,GW_UO_TYU_!QFDHQ_!
WDJPBFQGM_DEC,AEREQCPDSIAVBMLQTQFWPMC_DSX__DYTALBD_FGP'AQRI
ZBPDEMPYEBGIG,CLSCPDAMECAE_U_IED_DQ!GXN?_.GTDIJ_ACISDEBC_EG
PEHE_JADRNMPK'U_Q__.A_FTXKGFOI._QF?GH,AT_YED_DQNGXLIBYARIBD
EBC_QCDA_AFEFAPED,CQ!G_YEGBEL_TAFWHIYFOEHXTD_?!_'_DDITH_HQI
FMACDDGPAJDCEFT_HGQMHD_K__P,C,EAT_IFF'NDB_AQANUXJPVEMMPITCW
HRNWNYKPAF?GHLA_TIA_DNQFNUHHET_UDBXADNGGCDHQFT?XBJU_DAFGHMQ
PILAPOELXD
```

Challenge 2

```
KFHASWBKFRCOORNDFA@SIMDCC@EAPWAHCSCJR@@BFALE@SII@CQIGZ@MFUK
T@INAKNBU@SBN@EDFHRO@I@COORWRW@UBERE@FOXCSI@KVBHAREJ@SSX@PQ
ID@N@FBT@TOAESEB@PTQCRRI@A@Z@RBETEI@FUITKLE@GICTBI@QVBENFLI
C@BFBREOFTFJ@CRSB@UASABLRQ@NA@BKMAXXUNM@THJSIFI@C@A@SETY@CR
IIIUR@LKTTNE@MAEBTJRCDAKLING@LN@INC@UDARAQDAYZOUKIKG@ROBTOC
AYBI@SWEW@ALD@RHELFILOR@GBD@O@ZFMO@TIEALRTBSNI@REW@KLCHWR@U
HOLDCB@QEAU@BE@FIW@CSLOUT@RRCRTCBGGBANFBODZ@KMOWRCBGGTBMGBQ
ANBIKSIRUMOBCRTIMWJJLEARF
```

Challenge 3

The ciphertext alphabet has been mixed with a keyword, but the plaintext alphabet has not.

```
QRTANTXWUPNORQJR@QRAQSBIINEFTRTBCNGWCYKUFZTRCOOOBOCRCTD@TNT
PGTQCCHIERKIOVRXETCGUSCYYRBBSETFNTDXNTSURYNUQTZCJRRFOGAOYOD
```

```
JNRDONKOOTQMSRUSTEXOCWORTZECU@TAHOTDN@FCINU@DNNJTM@P@Q@GCTF
@IIDZXOGBGEONTOKT@OURFJTRW@INPRLNTOUGNEEMNUVUGSRGZNTNERKGQB
CZNREQTDIAGTNFHTUSTG@EDKGRFRINRMZONNJRTFTX@RGQCZNT@KNPGRJTP
KSNTIRCBEQRKRTREKLTVYNXTTONTOPUHHUQDOIUNRTZPTGSGORG@DT@NOVN
VSRQEOBRCRAIDXACITRJNNOTNNCCXJOLMDTERBNUKYOGNFRTFCXKQBQFBCO
C@PXU@WNRNTUXFONNRTQWWTVGPFSRD@XNTHQLTDSC@PAU@INVNOPTNHC@R@
A@IRSA@VNTUTMCTEFBSEDFUCT@VQKBUEN@RSTKXGRSSPTSNEFGQRCSTOAQU
CNQ@QJSPTMYHTCBESUFNFCOSQPHREQINNNQURNODMWRSWRFGRQOSRSIQCVD
CF@TREWKOMKORKMVTRFFKEXBWH@SDSXT@XSQTNVCVNTFOT@OPRCESQRTNRC
GTGOXZDJGUPCN@OTCZZTELNOCYWR@XSQTNJTVNONNTCCYWOOJRCQGGFFEPB
TFURMQX@CRTDJNXCCNRHZ@SCKQSJO@VOVKRUNOFINPTZCTREP@DXOUUOQOT
NSRNFRVKSNTAZNQBGRCUXSTVGRCGRRONGPAQNTPWPTVNRHJECXKFNAJDNXC
TSCJNEUXOGNTQWFFJ
```

Challenge 4

Both plaintext and ciphertext alphabets are mixed with the same keyword.

```
@CLE@EEEEBBMBSYMCTLBVJKQULICZ@CT@EVOYEBHNTLEBUEH@EWJERRJUKZ
R@IE@DBFRDUN@ILALZMA@EGPEEAYUAJC@@M@@EDUAEAXKIBYPESJLMGB@LS
G@QM@MVL@CDAKSUBANNXRDRMPRYMLIGNEBXEJL@NILVQRBYOKARUKEUZAOL
TFB@BVJIYLAFHLRCTRXENK@UKMVBAMVRLMHNTE@RHFMP@BVFKHAFWJEIJBL
NGXLOU@IA@MIGL@CERUENJYTJDHNM@XUFCYUVNGNDMALHG@FTDETBRJ@MAT
LGRHLAGLRQANRWUZMLL@MJBIEULN@MATLELKX@EUUJEABBTUAS@THEAJLTN
XHZULBAMGRMDXVZ@MYTEGRLOLMHLENOGWRLMBUG@MUXGX@M@OLBKIUBHTLE
IJJZT@MSLRUW@ELMGRLZJHFMP
```

Challenge 5

Plaintext and ciphertext alphabets are mixed with different keywords.

```
PFVHOIBDHTDMSKC@EKZOGLUYKTX@TLYVAKJBRYBY@YESMOAKVEYXYWTAAZL
DCUBMAACWKBMYYBEASVAZYAEKYFZBTDJHLNUDXKEFKAYYEMBCYABLBYQE@Y
AMCAAUIMOFGDAJWCTRDPBAPDAPKVDEM@XLBFMAETDBMFXJEAYAECLIYZLKY
HH@KEBYSDBNDO@BX@HRFUMBXPSNBRGEABGPMUFFIA@CAMBEY@HEPAKBEQIL
MCDBCXX@@ODEUOJEESQAXI@CHESEAAZFAVMZZELIARHF@SLKCDBYIAESLKB
LZJERFBIAQJFBUOO@EYZBEMJCIASGAKGCU@ACNAAYICAPBRYUPNAHFIBEBZ
CVMOZBE@QL@EMYELEFJDYKLESYTDYLXVBCOBROENFDUKXSVIGLPAS@RFV@K
XXBFADHFYQJFEAMJHUSFIUQCDVVDXLKSDAKMRYSIUDTIJYUIBKHWDMRFYYA
QKEQBEHLEKDBKHGNKLWEYG@@RECMOMKRHBBCKTARMMFO@IAABFQVIEFBLFZ
@@KMEEJECLYAGYAFPASYKAZODVMKDURFGX
```

Unit 166
Chase cipher

The *Chase cipher* is an invention of Pliny Chase from the 1800s. It begins with a 3×10 grid in which we place the alphabet, mixed perhaps with a keyword. The remaining four spaces are filled with other symbols; we will use the digits `2`, `3`, `4`, and `5` (`0` and `1` look too much like `O` and `I`). The coordinates of the items in the grid are written above and to the left in this example:

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	K	E	Y	W	O	R	D	A	B	C
1	F	G	H	I	J	L	M	N	P	Q
2	S	T	U	V	X	Z	2	3	4	5

We take our secret message:

```
MY SECRET WORD IS OBFUSCATE
```

We need to decide on a period. We can take the entire message at one go, or break it into words, or break it into blocks of the same length. For security, we should either use the entire message at once or use blocks whose lengths are all the same. For this example, we take blocks of length five, and pad the last block.

```
MYSEC RETWO RDISO BFUSC ATEXX
```

For each block, we fractionate by writing the coordinates of the letters in two rows, like this:

```
MYSEC
10200
62019
```

We take the bottom row and treat it like an integer; in our case, the integer is 62019. Then we perform some mathematical operation on that integer. Here there is some flexibility, but both parties to the message must agree on what operation to use. The only requirement is that it must be reversible, so that the recipient can decipher the message. For example, we might multiply by 7. Our new set of coordinates is

```
010200
434133
```

Notice that we add a zero to the beginning of the upper row, so that both rows have the same length. We could add any of `0`, `1`, or `2`, and should choose randomly, for security. The new coordinates are converted back into characters, according to the table.

```
010200
434133
OIOTWW
```

One possibility for the full ciphertext is

```
OIOTWW VRA5WB WCOGUB ZDGX2W XC4KS4
```

The Chase cipher is very flexible in how it is set up. The length of the blocks can be changed, as can the mathematical operation used in the enciphering. With another choice of operation, such as taking the first few digits of the logarithm after the decimal point, the blocks in the ciphertext can be the same length as the blocks in the plaintext, rather than requiring the addition of an extra digit, as in our example.

Reading and references

Pliny Earle Chase, “Mathematical Holocryptic Cyphers,” *The Mathematical Monthly* 1:6 (1859) 194-196, books.google.com/books?id=SVNLAAAAMAAJ&pg=PA194

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, pages 203-204.

Python tips

In the `random` module is the function `randrange()`. It returns a randomly chosen integer in the range from 0 to one less than its argument.

Functions can be used as arguments to other functions. This allows us to choose which function another function uses. For example:

```
def mult(x,y):
    return x*y

def execute_function(x,y,f):
    return f(x,y)

execute_function(2,5,mult)
```

Programming tasks

1. Implement the Chase cipher. Separate the mathematical operations as their own functions, so that you can choose which operation with each use of the program. Implement multiplication by a constant, as in the example, and any other operations you would like.

2. Implement a dictionary attack on the Chase cipher.

Exercises

1. Apply a dictionary attack to this ciphertext. Can you find another way to attack it?

```
U5EFA3 KEBPU4 AVMIJM KIGBJU UOHKI3 2UUIFE 2AHIUP WTORBU
UUJAK4 V5EFA3 2AJTMA DXXLBY MOIFMC BUTOJM DBTPUP V2BGKI
IHQTOR NKODQI DSSGBG ZCALQI UIFMGS NCHTAO GGINIS VOHPIO
ZLIPAR NKODQI ASWFAS JUTAHY TBGNTH DQTBAG GUENJB AEB2NI
YJSTBU XTRIQI V2FUBY KEFUWI AGBPLY JOIG2F DUBINI NRATVM
GUGTIF NAJTOH MRAVMA NCBMTR ZOTWHE NAHMSA IQIAYG GTIKIG
VK2PAG GSTBBG SYTNTH VIFUIE WLRUFU WSTPTS ALZIGE 2TRUU4
MIPJUU SENKIG VKINIS GBLAIF DMNTSL SZIJIG DBMAIF 2AIGDL
DWAPTA ZMIVMA JAOTWF OMAHMS WUQINI NBTUHM 2NWREM VITRLM
GPTHMG YINDOT MJDBOE BIWURF GT2COO GHIJFU 2JBUNT VI5INI
2RATSA GNIADA MGJJTY U4OIFS WOEMJ4 IBTUHM NPLIWU SQINLY
GLPCJT YHHTKE NAHMMG AF4OIF SWOCOO WGUQQI AVCLWI IK52NI
IRATSA JPE3EI BMBAHM VEBOLM GUSEKJ ZIVMWU OOLIKT YHNUQM
WNLIKT YHJVIF DUIET4 MAIGBG NAIHIF IUOONF FIUQ2F AUIFAS
MUTAGD GBPL2F IAJBPI DZJIOK AWTFIG 2CQIAS YIHRFU TSTKEA
NBKIPI NK4MGD MIGDLS WAK5IF ITRTBU XOIVUR SBHEMM VOQEUP
AIBBCD OOLMMC BOEOWU 2JIUQY TLPCJT JH2KEO THDQIA VMPIMM
GLPLIS F4LOCD 2POWIA FYT5IF BEARUI DVINIP ZUOSTH DVINCB
YEHDSA MGFINI GDRTAG VFEACD ASMGFM WFUBIF NAH5IS GFBMUF
DUWIUY GLPCNO WMOFUS ZIGURE KLMBUE AIBKIG ZISMGD TFCEMG
TEIBKT YPIV4T UIFBOC WUQIFA VETGBG WTMTDH ITAOIF MIKNIF
SUBUNC KISV5I ORJT2E GLQIOB TUEIEA 2BKIPI ASMGFM TPAEHL
AZLUNC ZISV5I MALBCI 2JUGKI 2HVMMG AFINEE NTBFMG MEOBOR
TLJEMU YJTGOT VGESTO AMNJTM TFUWLM WNLEME BLILAG ODO5IF
ZEARUI ASMGFM WNCCEH ADSMGD FINLFC WOSTHS SMGFIF ZEDRUI
YHFEAC SDWXXV
```

Challenge

```
JQQJYJ A4OJ5F NDUAPF OCQGEF HOKVXC REOAXH E3RM5P CFRQG3
JZ2PT3 RZIDPT QGNNUR JJDOXH RSDLS3 WJ5FTU D23QG3 PQZS3U
TWIDPT DGKYY3 JCJDOJ AP3URT QGKC5V CNSCHT NJY2IW C3Y3Y3
GURAAC RCEA5V 5NSKVQ YFZJTU H3XCGT DGNS3U TWJINP T5IO5V
Y3GRMA OMU5NJ AAHA5V CEFQG3 RAHSYJ ZS25ZF NCXCHM OYFKNF
GYUXCG TSOQG3 JYQZJT CAFUDP KX23RF RSDPYM HC5EAT YFY3UX
```

RSFHFT DHXCKP OMRQG3 GR54AC QT2XCG LD55QG BHEA5F QYH2LJ
Y3ULAC 5AC3TT NXCKOJ YEA5FT QG2WD3 AROAXH D3RFXU R3DXVD
JILJ5V EOUUSF O2ZTMU GXSOMA YAY2LJ YEAQG3 TNQA5J QYCQG3
YZMLPF WMHQCV ZX23RF 35JWAA YKCWJU BHAQG3 RNQA5J QXKCWP
5FRQEP OMSTDP BHHWY3 KSMLMT JJY2JT 5VUTXU KJWFFR TUTI3U
WJLRX3 CHKGRU LNCTRF D2EYVM TPJDOJ GSZRTF BMQG3R ZIGHXF
OHD3Y3 YUSP5V 53YHX3 GY3QRS C2OAEO OHHWNQ TA5J5V RULP3U
RRQGKY GYNSKC RWPYNJ EHZKHA NQYCHT ZPDODT QG3GRU NMDUZJ
L23RFX NLJU5F HCQGNP TDYCWJ Y3UWJX EAY3XC TGUTPQ ZPICRW
YEAUTT QG2WD3 LRDMT3 PXCGFH TMD2MQ LVUDFQ 35XAHA WFECQQ
BCKV3R 3MJTRU OCCCAC CPDENF JFFFRT GMSIKM BHD3RT QGNXHR
BMJXEO QOAIOD LAEFYU NQZIDP YCPNSV 53RGXU HENHWU HHD3Y3
YAQAWU GYAQG3 BHKTSF HZ3OZF GYCHXU NLXYJT D3YMCV NX2S3U
GA23DP WIOZEF HHHG3R CCPNSS RPQCVT YKVXCG LDLIG3 LKAMCV
GJ5QCV RQDP5V CESDAW OHCPEA RNQATW E3RVMA QSYC5V 5NPYNJ
QHOANY RPJU3T EGKYY3 NCJDOJ BXDH3T DG3XUT R53RIF QA2EAU
ZIGCLY RI2LJU OHPPPQ CKHAI3 PLXYIW EGKCGR RFN2YQ TZJWJU
DSG3ZJ T5KUDY EYCQG3 OYXUWP CFPD5F GYCN2Q ZVHG3R JPYHCO

Part X
Proto-mechanical ciphers

Unit 167
Disk ciphers (cipher clocks)

Consider a device that looks like a pocket watch. Around the outer rim are symbols like letters and digits. Inside that is another ring of symbols. The outer ring has *m* symbols equally spaced around it, while the inner ring has *n* symbols, also equally spaced. The device has two hands like a clock. When the longer hand moves from one symbol on the outer ring to the next, the shorter hand moves from one symbol on the inner ring to the next. In other words, the hands are geared in the ratio *m*:*n*. The device can be used to encipher a text by rotating one of the hands clockwise until it points to the first plaintext symbol; the ciphertext symbol is the one to which the other hand points. The first hand is advanced to the next symbol in the plaintext; the second ciphertext symbol is indicated by the other hand. This continues until the full text is enciphered.

The *Wadsworth cipher disk* uses the outer ring for ciphertext symbols. It holds 33 symbols, the letters A-Z and digits 2-8. They can be removed and replaced in any order. The inner ring is for plaintext symbols, of which there are 26 (just the letters). Rather than having hands, in this device the two rings that rotate in the same direction under a single pointer. To encipher a text, the inner disk is rotated until the plaintext letter is under the pointer. The gearing rotates the outer ring to bring the ciphertext letter to the pointer. This operation is equivalent to the “cipher clock” described in the previous paragraph. The disks are always rotated in the same direction, presumably counterclockwise.

There is no evidence that there was ever more than one Wadsworth disk in existence. We do not know if it was used more than once. It was invented by Decius Wadsworth in America in 1817, more than fifty years before Charles Wheatstone reinvented it in England (too slow, Britain!).

The *Wheatstone Cryptograph* is better documented, especially by Wheatstone himself. Its outer ring holds the plaintext symbols, which are fixed in place. There are 27 of them, the space and 26 letters. The inner ring has 26 ciphertext symbols, which are the 26 letters in any order. The initial position before the first letter is enciphered is for the longer hand to point to the space and the shorter hand to point to the first letter of the mixed ciphertext alphabet. The hands are always rotated clockwise. To encipher a text, the hands are rotated clockwise until the longer hand points to the plaintext letter. The shorter hand points to the ciphertext symbol.

Figure 167.1: Wadsworth cipher disk. Photo from NSA.

Figure 167.2: Wheatstone Cryptograph. Photo from eBay.

In his paper describing his device (citation below), Wheatstone introduces a novel way of mixing the ciphertext alphabet from a keyword. It goes like this:

1. Write the keyword in a row.
2. Take the 26-letter alphabet and delete any letters that appear in the keyword.
3. Add the remaining letters in the alphabet under the keyword in rows with the keyword letters at the top of each column.
4. Duplicate letters in the keyword are deleted.
5. The mixed alphabet is read off in columns.

Here is an example using the keyword `WHEATSTONE`:

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

The mixed alphabet is

`WBPHCQEDRAFUTGVSIXJYOKZNLM`

The procedure for encipherment on the Wheatstone disk is

1. Remove all punctuation from the plaintext. Keep only letters and spaces.
2. Either:
   a. Put an `X` between any double letters in the plaintext. If the double letters are already `XX`, then put a `Q` between them.
   b. Replace the second letter of doubles with `X` or `Q`. Be sure not to accidentally get a doubled `X` or `Q`.
3. If the plaintext does not end with space, then add one. The last ciphertext letter serves as a sort of checksum.
4. Position the longer hand of the device so that it points to the space on the outer ring of symbols and the shorter hand so that it points to the first letter of the mixed ciphertext alphabet.
5. For each plaintext symbol (letter or space):
   a. Rotate the longer hand clockwise until it points to the plaintext symbol. Do not go all the way around. Let the shorter hand move on its own, according to the gearing.
   b. Read the ciphertext symbol, which is indicated by the shorter hand.

Step 2 is necessary because there is no combination of ciphertext letters that the Wheatstone disk can decipher to a repeated letter, since the ciphertext alphabet is shorter than the plaintext alphabet. This is not true for the Wadsworth disk; for double letters on that disk, we would simply rotate 26 more steps for the second letter.

Let’s look at a short example and encrypt this message with the Wheatstone disk. We will use the mixed alphabet from the example above.

SECRET MESSAGE

First, we have to prepare the plaintext:

SECRET_MESXSAGE_

We begin the encipherment by putting the hands of the device in their initial positions.

The first letter of the plaintext is S, so we turn the large hand clockwise 19 steps until it points to S. At the same time, because of the gearing of the device, the short hand also moves 19 steps and lands on Y, which is the first letter of the ciphertext.

The second letter of the plaintext is E, so we turn the large hand 13 steps clockwise to E, while the short hand also moves 13 steps and lands on E.

Since we always rotate clockwise, to encipher the next letter, `C`, we have to turn the large hand most of the way around the circle. The short hand indicates that the next ciphertext letter is `Q`.

The process continues, always clockwise, until the entire text is enciphered. By the time we get to the space at the end of the plaintext, the short hand has moved so far that it points to `A` in the inner ring.

The full ciphertext is

YEQORNCXFLHMRVGA

The *Pletts Cipher Machine* was another cipherclock based on the Cryptograph. Its innovation was to mix both the plaintext and ciphertext alphabets, to make breaking its cipher more difficult.

The astute reader may have noticed that these three devices give us stream ciphers. Let's consider the Wheatstone disk. The internal state can be thought of as the positions of the two hands, (h_{long}, h_{short}). The initial state is (0, 0). The cipher is factored into a stream cipher that corresponds to the disk using the key ABCDEFGHIJKLMNOPQRSTUVWXYZ, followed by a monoalphabetic substitution cipher M that uses the mixed alphabet as its key. For each character p_i from the plaintext, which we think of as a series of integers (space = 0, A = 1, B = 2, ...), the action of the encryptor is

$$\begin{aligned} x &= p_i - h_{\text{long}} \mod 27 \\ h_{\text{short}} &= h_{\text{short}} + x \mod 26 \\ h_{\text{long}} &= p_i \\ c_i &= M(h_{\text{short}}) \end{aligned}$$

For the Wadsworth disk, exchange the rolls of h_{long} and h_{short}, and replace 26 with 33 and 27 with 26. For the Wadsworth disk, we also need to specify that if x is ever zero, it should be changed to 26; this allows for the encipherment of repeated letters, which is never done with Wheatstone.

Let's now redo our example encipherment with the Wheatstone disk as a stream cipher. Start with an internal state of (h_{long}=0, h_{short}=0). The first plaintext letter is S=19, so x is 19 − 0 = 19. The internal state changes to (19, 19), and the output of the stream cipher is 19. Applying the monoalphabetic substitution gives us the first ciphertext letter, which is Y.

W	B	P	H	C	Q	E	D	R	A	F	U	T	G	V	S	I	X	J	Y	O	K	Z	N	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Next is E=5, so $x = 5 - 19 = 13$ modulo 27. The internal state becomes (5, 19 + 13 mod 26) = (5, 6). The output is $M(6)$ = E. Then comes C=3. The internal state becomes (3, 5). The next ciphertext letter is $M(5)$ = Q. Etc.

A more interesting way to define the internal state S is as the total number of steps taken. From this point of view, the encryptor does these things for each character that it enciphers:

$$
\begin{aligned}
x &= p_i - S \mod 27 \\
S &= S + x \\
c_i &= M(S \bmod 26)
\end{aligned}
$$

During World War II, the Danes used a derivative of the Wadsworth and Wheatstone devices, called *Urkryptographen* (Danish for “clock cryptograph”). Like the Wadsworth disk, it had two concentric rings of characters that rotated counterclockwise under a single fixed pointer, and the key was written on the outer ring. Like the Wheatstone Cryptograph, the plaintext alphabet contained a space which the ciphertext did not; therefore, double letters had to be disguised with X. The letter Q was used to indicate numbers, which appeared with some letters on the plaintext disk. The two letters X and Q are not used in native Danish words. Unlike either of the other two devices, Urkryptografen had three additional letters in each alphabet: Æ, Ø, and Ü, so that the rings had 29 and 30 characters. The plaintext alphabet was also mixed, and there was a number of preprinted disks. One such plaintext alphabet was (where _ denotes the space)

_ØACUHDGVMKBÜIWRNPOZÆEXFSYJTLQ

Figure 167.3: Urkryptografen. Photo from Museum of Cypher Equipment in Fife, Scotland.

Reading and references

Charles Wheatstone, “Instructions for the Employment of Wheatstone’s Cryptograph,” *The Scientific Papers of Sir Charles Wheatstone*, The Physical Society of London, 1879, pages 342-347. archive.org/details/scientificpaper00londgoog (the last two pages of the article were completely ruined by Google in that copy), books.google.to/books?id=CtGEAAAAIAAJ

William F. Friedman, Several Machine Ciphers and Methods for their Solution, Riverbank Laboratories Department of Ciphers Publication No. 20, 1918, www.campx.ca/Several_Machine_Ciphers.pdf and www.marshallfoundation.org/library/methods-solution-ciphers

James Stanley, “The Wheatstone Cryptograph,” incoherency.co.uk/blog/stories/wheatstone-cryptograph.html

William F. Friedman, Six Lectures on Cryptology, www.nsa.gov/Portals/70/documents/news-features/declassified-documents/friedman-documents/publications/ACC15281/41785109082412.pdf

“Ciphers and Cipher-Writing,” Macmillan’s Magazine, XXIII, Feb 1871, pages 328-338, babel.hathitrust.org/cgi/pt?id=mdp.39015004979913;view=1up;seq=340

NSA file 41788379082740:
www.nsa.gov/Portals/70/documents/news-features/declassified-documents/friedman-documents/patent-equipment/FOLDER_515/41788379082740.pdf

Basic Cryptography, Dept. of the Army Technical Manual 32-220, April 1950, section 142 in chapter 11, www.nsa.gov/Portals/70/documents/news-features/declassified-documents/friedman-documents/publications/FOLDER_238/41748889078809.pdf

Niels Faurholt, "Urkryptografen (The Clock Cryptograph)," *Cryptologia* 27:3 (2003) 206-208, DOI: 10.1080/0161-110391891874; this article is available also at www.jproc.ca/crypto/crypto_watch.html

Greg Mellen, "Cryptanalyst's Corner," *Cryptologia* 8:1 (1984) 55-57, DOI: 10.1080.0161-118491858773

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, pages 195-198.

Auguste Kerckhoffs, "La cryptographie militaire," *Journal des sciences militaires* IX (1883) 5-39 and 161-191, www.petitcolas.net/kerckhoffs/crypto_militaire_1_b.pdf, www.petitcolas.net/kerckhoffs/crypto_militaire_2.pdf, section IV.

Donald W. Davies, "Charles Wheatstone's Cryptograph and Pletts' Cipher Machine," *Cryptologia* 9:2 (1985) 155-160, DOI: 10.1080/0161-118591859870

Programming Tasks

1. Write a function that mixes a ciphertext alphabet using the prescription of Wheatstone.

2. Implement the Wheatstone disk. For the decryptor, think about how to handle double letters in the ciphertext.

3. Implement a dictionary attack on the Wheatstone cipher.

4. Implement the Wadsworth disk. You do not have to worry about double letters in the ciphertext, since they cannot appear with this cipher. To handle double letters in the plaintext, advance the device 26 steps between them; the shorter hand will return to the same place, but the longer hand will point to a different ciphertext character.

5. Learn Danish and implement a simulation of Urkryptografen. Brug det for at skrive i sifer din favorite aventuren av Hans Christian Andersen. Skift din bakkeopstigningangreb på Wheatstone-kryptografen (Unit 173) at tilægge de tre nye bogstaverne, og brug angrebben din at løse siferteksten din.

Exercises

1. On average, how many characters are enciphered before the Wheatstone disk passes through its initial configuration? Use your monogram frequency table in your calculation.

2. Encipher this text with the Wheatstone cipher using keyword `CONVOLUTION`.

   Viewing the Wheatstone Cryptograph as two rotating pointers is only one way of seeing this cipher. Another way is through the lens of modular arithmetic. The modulus for the plaintext is twenty-seven, while the modulus for the ciphertext is twenty-six.

3. This is the example ciphertext from Wheatstone's article. It was enciphered with the keyword `FRANCE`. Each paragraph is enciphered separately; i.e., the disk is reset to the starting position at the beginning of each paragraph. Decipher the message.

```
PZLSPQREQAJDITFBUFZOHQOSUQUDIKITORTWEZACMTPLERAUESKGSOFGFDK
HLSJIRKHFHMFADAYIVUOHAOBLNOGREJAIBKMPJZTMJABQCNFPOMYHYRCZDC
WBXUBZ

ZBILIJTEJYSPFDLCXETKQASOXOUNNODQJCWECLXPUYIEMMCMSYVCFPOKWCD
EDVDAGLPEEKNAGVKMNUULSHXYXYVGFQPUYIORQKLPTCZHHK

ZBKUPVSWZWXAQXDREKTKQASOXOUIRSKOMFSTIIXGWTQJJVDYFNAHLSIIXIA
GQLZXVOGNHGRBUOHYZOOPWVYDDMQJKFMOBJPDYVRBAWKGWSJURJGITOWTVE
ZBHSOSLVUNBCHQSOTEIEBDQMGWHGJAMISXFIFBBPAVPESVCJUTAD

PZLPTYVXQXDTGLTTAFCVMHOMBINJKWVYAZOCQLAIUKFEGFNCNFIZHHKVZYQ
UGLIVENKAHTRVFVEBWHWLRYCMLXWSYSYJHUFSFPOKEGZRXLUBFT
```

4. Break this ciphertext with a dictionary attack. The keyword is a ten-letter English word. The cipher is Wheatstones's.

```
ZUKMNXQCDJKEADWCJVQHKCWNGGFVMAGQFEOCOWOIXTVIAKJPNIJJXUUQKCL
BXVABOFIAATPQHIVKJQYUWZQRZKXWXYZVILALHDSSIGGMQQVKZVYCVZUPMH
RTCTJXCFXBIJGLNUVWEBTJZTOQLIKPRYEDCXIGDMETEYYPOWFEBARSZZCOH
WPABUIZCLWJJYQQKGPTTLWAIZUKXNGWDVHVDXUXEXEBJWJPPSXPNEPUHRHT
ILXRKSGXKWMCZFZMBTVMNUDCDVLAFCQRRMGGFOUWAHVXHNHABCUPXKZOABA
WMMZGFVNOQCKYIOVGXDBUARIYSTJGYVGWLPLKEWXHJSZVIMSYFPL
```

5. Decipher this text with the Wadsworth disk cipher in which the ciphertext alphabet is not mixed.

```
TABLV3BTJWYCVKNRCOXFLSD2GTL6JUBUJNQ7CUXZG4AK4NT2LBO7DMXYLQD
QVCK8OW3AL6T2GWOU3MCPR4ODKVZK56APHABIZ36JUBUN2J8Q5M7XM3B2MU
C2K8YH4TAK6BMXM6DXGL24EF7XHLV7KBE2M3FMZMOERU3CZG5OSVDHZ46LA
FPAYBVG4IOH4IL3S6WYADMXQ3CILMUKYOYQHULPSAEWZ3I6CM6RBKXZN5P2
DEHL56LBMOEJ6CHYGKPVCZQBZ2O5RARJQGU78K6OBZDK6CKT4BMEUBYP38Z
NFY27LYH26P3GKY7YG7OCPBPSDSFT6XYFHMNSCQR7V8BY5RW3L2LPAOQ5FO
UIU8H7K2LAFU8YRBZJYL5V5THVL5HITGXKRFRGJW
```

Unit 168
Attacking cipher clocks with cribs

If we know some part of the plaintext (a “crib”), we can use it to break the ciphertext. To make it easier, we will encrypt the crib with an unkeyed cipher clock before we compare it to the ciphertext, since the difference between a ciphertext and a text that was encrypted with an unkeyed device is a monoalphabetic substitution.

An example will make the technique more clear. This ciphertext was encrypted with a Wheatstone Cryptograph:

```
DGBRHDRYDURNFTWLOMIBLSPDZCIJNSMBZXCSHDYCVMGOAZSAEGNVQLJRFTX
CWJHACZZMPEHMZNPZJYKMRXQGOUIUCRBOYQLOTOBUNZGJTEQBOFYLEHBOBZ
RRGWOHWJKCUKPZCIQELLOUEJVAYVYPQWHEOBJNVGMAXRGLYYHNAHYCYFTGR
ILIJNSMHQFUWYYUEJVNFIXNSDMXZSAHCNFVDVXVZKMCJWLXTBKJJGHHDRSK
CIMACCWTUKZVSRDGZSKCNUQQGCDKWKHVHIIFIXUMYIPWYSOPCGGSGHZBFWW
XPCEWEQMISKVXEMIZXSHAZEWOMRGQUTKCTKMPEUQJYLHTIDHFPERYUKSKON
NCEPLARCSBLMKOBCBSEFGXN
```

We have this short bit of its plaintext:

```
never wear out no matter how long it may stand in times
```

Before we use the crib we will encipher it with an unkeyed Cryptograph. The ciphertext alphabet for this process is just the unmixed alphabet. Before encipherment we need to disguise the double `tt` by replacing the second with `q`.

```
never wear out no matqer how long it may stand in times
```

Enciphered this is

```
OGXHUDAJGXGVBBIWXJWLECRENVCKOADDXRALSFUSUNOWJAXGLYSIMFT
```

Notice that we did not add a space to the end before encipherment, as Wheatstone had recommended, since we do not know if the next character after the crib is a space.

Now we compare the enciphered crib to the ciphertext, one position at a time. Start at the beginning:

```
DGBRHDRYDURNFTWLOMIBLSPDZCIJNSMBZXCSHDYCVMGOAZSAEGNVQLJRFTX...
OGXHUDAJGXGVBBIWXJWLECRENVCKOADDXRALSFUSUNOWJAXGLYSIMFT
```

Notice that B in the crib matches up with F *and* T in the ciphertext. Also, A in the crib matches up with R, S, C, and Z. These things cannot happen with a monoalphabetic substitution, so we know that this is not the correct position.

The only acceptable line-up we can find is at position 107:

```
...TEQBOFYLEHBOBZRRGWOHWJKCUKPZCIQELLOUEJVAYVYPQWHEOBJNVGMAXRG...
    OGXHUDAJGXGVBBIWXJWLECRENVCKOADDXRALSFUSUNOWJAXGLYSIMFT
```

Now we can begin to reconstruct the key by tabulating the substitutions for the letters of the enciphered crib:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ERCLKABFGHIJMPQ??UVXYZWON?
```

The missing letters are D, S, and T. Missing three letters means that there are six possibilities. If the key is mixed in some less obvious way, then we would have to try all six. But in our example, we can guess immediately that the key is

```
WONDERCLKABFGHIJMPQSTUVXYZ
```

and the keyword is WONDERCLOCK. Usually, we also have to try all twenty-six ways of rotating the key before we find the correct key, since we do not know the position of the cipher clock's hands when the plaintext was encrypted at the point that the crib begins.

Deciphering with the reconstructed key gives us the plaintext, which you should recognize as a paragraph from *The Wonder Clock* by Howard and Katharine Pyle.

```
CLICK BUZQ WENT THE WHEQLS AND THEN TICK TOCK TICK TOCK FOR THE
WONDER CLOCK IS OF THAT KIND THAT IT WILQ NEVER WEAR OUT NO
MATQER HOW LONG IT MAY STAND IN TIMES GARQET DOWN I SAT AND
WATCHED IT FOR EVERY TIME IT STRUCK IT PLAYED A PRETQY SONG AND
WHEN THE SONG WAS ENDED CLICK CLICK OUT STEPQED THE DROLQEST
LITQLE PUPQET FIGURES AND WENT THROUGH WITH A DANCE AND I SAW IT
ALQ
```

Now let's work through a more complicated example. This ciphertext was also encrypted with a Cryptograph:

```
VDZYHDCDOFCLZSCPWTKEIVTTRNVGSYSCPUVEUJOWVFTZBLXDYXNZAU
NICZVYFHIKQFZLEMNLQIJHRCGOJSRODYLLCKDTWQUAOXTAVJOOYOGU
ZGBRZXZIRDXAGYBLHRGUINGCCNQHJSUWPLEUVDDAWVORXIRJBUOOYW
FQOTXDFRDYUJYEVXUAITGMLIIHKGAEFFVDAKXEHGSYXAVVAQCNECYK
ZOXWFXRRQIJYAZVCPZDUWSHHPAWLVFKDYNLRBTSCYREOBLPVVXZMCF
DUNQSXBXSRKAPMWCYHTBUERKAVCCARWYDNEHXSWXBPHRACQIWIOYBX
PFMNUKGEFFDNBXASYSXDHQFLDNWYTDGMCWLTPUXZFUIIALEKEILREZ
JHNPAEIBYHRNVGDLKUFECLXCGEWMEFVLCZBUONKUISJCSOFCLZTNES
DUWSXNLROKNNGWCHBOUIZZATRKNNPQHXJMONSMZDWKGTBZKPDQFTIJ
NVXJRKZP
```

This crib is known to be part of the plaintext:

```
information exists of course but it is scattered
```

After we replace the double `tt` with `tq` and encipher with an unkeyed Cryptograph, the crib is

JOHQTPEXNTTGLEQABBJYQLOAGEFSOQJJQZKRAKSLWVOMBOCC

The first position that does not give a contradiction in regarding the monoalphabetic substitution is 177. This is actually the only position that works.

```
...YEVXUAITGMLIIHKGAEFFVDAKXEHGSYXAVVAQCNECYKZOXWFXRRQI...
    JOHQTPEXNTTGLEQABBJYQLOAGEFSOQJJQZKRAKSLWVOMBOCC
```

We begin to reconstruct the key:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
EFR?GSHU?VCKWLXTANYI?OZMDQ
```

There are three missing letters (`B`, `J`, and `P`) in the key, so there are six possibilities. Plus, there are twenty-six ways to rotate the key. That makes 26 × 6 = 156 total possibilites. But we don't need to try them all. Let's just try one and see what happens. Start with

```
EFRBGSHUJVCKWLXTANYIPOZMDQ
```

There are twenty-six ways to rotate this key, and we try them until we find something useful. Deciphering the ciphertext with the key as written above gives gobbly-gook:

```
IXUPCUFTPVDGPYCMDGBQIYDCPDVPQCPUDQSIPQCTPGUAHRSBVQTYR...
```

Next we try `FRBGSHUJVCKWLXTANYIPOZMDQE` and get more nonsense:

```
HWTOBTESOUCFOXBLCFAPHXCBOCUOPBOTCPRHOPBSOFT GQRAUPSXQ...
```

We continue to rotate the key, and soon we arrive at `TANYIPOZMDQEFRBGSHUJVCKWLX`, which gives this plaintext:

```
THE NEQD FOR INXORMATION OF AN EOACT AND RELSBCMFADIB...
```

Aha! We are getting close. Most of the key is correct, but now we need to find the correct placement of `B`, `J`, and `P`. By swapping `B` and `P` we get the true plaintext, which is from *The Modern Clock*, by Ward L. Goodrich.

```
THE NEQD FOR INFORMATION OF AN EXACT AND RELIABLE CHARACTER IN
REGARD TO THE HARD WORKED AND MUCH ABUSED CLOCK HAS WE PRESUME
BEQN FELT BY EVERY ONE WHO ENTERED THE TRADE THIS INFORMATION
EXISTS OF COURSE BUT IT IS SCATQERED THROUGH SUCH A WIDE RANGE
OF PUBLICATIONS AND IS FOUND IN THEM IN SUCH A FRAGMENTARY FORM
```

```
THAT BY THE TIME A WORKMAN IS SUFQICIENTLY ACQUAINTED WITH THE
LITERATURE OF THE TRADE TO KNOW WHERE TO LOQK FOR SUCH
INFORMATION HE NO LONGER FEQLS THE NECESQITY OF ACQUIRING IT
```

Notice that instead of trying 156 possibilities, with this method we only need to try at most 26 + 6 = 32 possibilities.

Now that we have the plaintext, we can stop. But, no, we can't stop. Never stop solving puzzles. Let's go onward and get the keyword. The key we found was

`TANYIBOZMDQEFRPGSHUJVCKWLX`

It looks random, but notice the sequence `Q`, `R`, `S`, (`T` is missing), `U`, `V`, `W`, `X`, with one or two letters between them. Also see `F`, `G`, `H`, `J`, `K`, `L`. So the key was likely constructed with Wheatstone's prescription. We proceed by breaking the key into columns and getting the sequences each to appear on one row.

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

If we search our dictionary for `TIMEP??C?`, we find the only choice is `TIMEPIECE`.

Programming tasks

1. Implement a boolean function that detects whether two (short) texts could be related by a monoalphabetic substitution.

2. Implement the attack that uses a crib to break a ciphertext encrypted with the Wheatstone Cryptograph.

Exercises

1. Break this ciphertext that was encrypted with the Cryptograph. Use the crib "the average collector would be bored." What is the keyword?

```
XRIHAOZJMWGRABONYAJXBGPMWMFDHLGBDBYDPVHTBKATCDNUIXEENOLSUHV
CLRCFHRVUQFKBMLMSFGLHMAIAYSWAMNEOPTSJQJBPGITLRYLAYCRJTGPWAK
WAZXSZKONBVNHLXFQDKSPUONMFEGQGBQBCNXNUHUXXTMRVXPVUQFKCQLMNY
XQUIKNGPHANTBVKIVDBAVWONAZGVEJAMSJKXLHQTDZFFPJMSFLKMVXGYSKB
TBIWTYDNGXOCNUYOHEDACVHXYPKUGNDOIJGQRLSFQEOMMXHQSPRYUMBIFYC
VFYGAQJWSJOISLFVESKJNIVMXQXEKYMYXRVELATVXPNBOMEABVQNUUBVIVU
YOZQXYETDKKGGWWDEIPXNEXTMUFXNESNZXUNJIHVAPMFKYZBKWSXIZVMSCD
ZEQVCBIGGWUAPMGWMZKBIZPJURNLOKMUAQJGSNVKAODZMYJGJIMCNUCYKMV
VLZCQFSLNWWDYAKQHQGQRVELDBYSGONJEREYNPKEXSAEUCRQJREGWOIOIUK
```

```
LADDEZQRONHZALWJPEYHMDKIYVYDPFZBSFNMDVEOSQTKDNYIAXUDYAKNQEC
UULNLCJICNOOHMVBTCYQLSXSIWBNESKTPOYCXOCOLMKHXYFKCNWIPRDGMWQ
```

Challenge

crib: `NOT A SOUND BROKE THE SILENCE OF THE STILL NIGHT`

```
DOW#GLGHUJ#VOWNQJLPNWBDZQDAQ$R@CVZIXRMEFP#HIGNPAMGCIKGJTPCI
DQTCSLF$UHAXNE#VNGJUK@WDITNJXNHYRKIYQGEQ$NE@IUVHRT#$KCWLVKS
JEOXCLESNBFKFMJAS$LIOSCNVKLFKMWN@#GHPJOXEYVLOST#RHGOMFASYRT
F#AVPHFGUS@TQHUBIXYB#JLTCJRDUS#QND$UEKBTCTAZPWLCXQYPSLUPG#J
@AMPXB@JI$VYJRTZAFRZU$MNOKASVKABVTBHK#ELPYF$KXQSUJDOCHFLESG
CBZLDHXCRZFENO#BC@YPGW@TGJVTANFVIYFKGYOIZSV@JCSLUPG#BUV#LNS
GHILJZTYNVZUFCPQHRKCESL$ATUZE@IUOUKCWYKBHVOAUPF$PSGSADHP@ZQ
M#TGJ#H$N$UQRHDFN$JXE@IOQLMKRYJNWNQBSW@DROQT#VPENJ@YJIU$R@C
VHUXRIYABOYQZGJDKZJ@QIDQTCH#PHXAHKQSTEUO#N@RSTYPBZTFWLZ$CSZ
@CLVDP@RXE$ZGKXJADKWITE@JMJPCTLYOAZXAQRBOWRIJSNDJ@YRXEFEHXO
HYR@Y$ZCDKMRLAGJFTIEBNK#LIWNCXQYP#YJNWZKTBGOPTGBHPLI
```

Unit 169
Digram-counting attack on cipher clocks

Suppose we have a cipher clock for which $n \geq m+2$, i.e., its ciphertext alphabet has at least two more characters than its plaintext alphabet. Then, given a configuration of the clock, there is at least one letter in the ciphertext alphabet that cannot be the next letter of the ciphertext. Here, for example, is a clock with $m = 10$ and $n = 12$. We have put the plaintext alphabet on the inner ring and have not scrambled either alphabet. From its current configuration, there are only ten possibilities for its next configuration. Even if the short hand goes all the way around the clock, the long hand cannot reach two of the letters. In the figure we see therefore that A can never follow B in the ciphertext. Similarly, B can never follow C, etc. We can use this phenomenon to break the cipher, given a sufficiently long ciphertext.

Our strategy is to list all digrams in the ciphertext. Then write down all digrams that do *not* appear in the list (except for doubles like AA and BB). Reverse them all, so that we have letters in the order in which they appear clockwise on the disk. The list may be long, so we will find it easier to keep them in a grid. There is a way to eliminate those that simply do not appear in the ciphertext but do not indicate a restraint: Whenever there is only one digram in a row of the grid, we know that that digram appears in the key, and so we can delete all others in its column. Likewise, if there is only one digram in a column, we can delete all others in its row. If we should end up with a 2×2 or 3×3 or other $n \times n$ rectangle that cannot be reduced further, then we have to try $n!$ possibilities. In the end, we hope to have only the digrams that appear in the key so that we can reconstruct it. Then we must try each starting letter in it to find the correct plaintext.

This attack only applies if the length of the ciphertext alphabet is at least two more than the length of the plaintext alphabet. If n is two or more greater than m, then the missing digrams indicate what letter(s) is found on the key just before the last ciphertext letter that we wrote down. When $n >$

m+2, there is a complication: We look for rows or columns with exactly two (for $n = m+3$) digrams before we can eliminate others. In the end, there will be a list of letters that cannot appear before each ciphertext letter in the key, and we have to reason through them to reconstruct the key.

Let's work through a simple example with $m = 5$ and $n = 7$. To have such a short plaintext alphabet, we will first run the plaintext through a Polybius square. Here is a piece of text from *Flatland: A Romance of Many Dimensions* by Edwin A. Abbott:

> Imagine a vast sheet of paper on which straight Lines, Triangles, Squares, Pentagons, Hexagons, and other figures, instead of remaining fixed in their places, move freely about, on or in the surface, but without the power of rising above or sinking below it, very much like shadows—only hard and with luminous edges—and you will then have a pretty correct notion of my country and countrymen.

After a Polybius encoding we have this:

```
2432112224331511511143444323151544342135113515423433522324132343444211 2
4222344312433154344422411332231154343414511421543351533441122343343231 5
5311223433431133143444231542212422454215432433434415111434214215321124 3
3243322212453151424334423152442353111131543323451152142151531541112344 5
4434333442243344231543454221111131512454452244423344544442315353452154 23
4214224432433221112345115344243243325243322121531345224445115425432451 3
2331242515432311143452433433315423114214113314522444233145322433344543 1
5142215431133145434455224313144231533231151151135421544445413344242151 3
4433344424343334213254133445334442541133141334453344425432153 3
```

Here is the ciphertext encrypted with a cipher clock with seven ciphertext letters:

```
EAHGENHOAOXHAHENAGXHAHENGEONXNHGENAGHGNEXHXGOXNXAGENEAEXNEAOXAGE
XHOEOAGNXHONHOXONEOANAHGOAEGXGNGNGENHAEOXNHGOHAGOANXNAHGXGOXOAGX
GXAEXAHONAHEGENXOAGXAGENHGHOHOEGEXHOEXOXNXHGOHAEAGEGEOXNAOXHEONG
NAGXHAHEGEGHGEGEOAGNAEOHGXNGXNHXANXNHANEOAGXONXNEAEGOHANGXHXOXNA
EONHEXHGNANXNAGEGHGEAENHEXHEOAHEOAEXHAEAHONGHEGEOXAGEGXNGXHXOXAG
NHGXAOAOAEGOAGXAHOAEGOEOEANHEAHGEGONXNXOAOAHGNAEOAHOAGONHENANHOE
AHGNAEOEAOXHGXNHGEGHXAGNGNGXAGXNANXNAHXAGNAGXONEGHEOXNAGHOANAGOH
XNGOXHOEOXONXHANANENENXONENENGHENXONXNGNAENHONAHAHEGHOXNAHAEAEGH
EOXAEXHXHXNXGHAEGOEOENAGHOXAEONGNEOEOXNGXHEOHXHENXGHGEXGOAHOAGXH
AOXAHOAGHGNANAOHONGOENHOANXNENENXOHXHENHAEXHOEGEOXNANG
```

We make a list of all digrams in the ciphertext and make a grid, where the first letter of a digram is the row label and the second is the column label:

	A	E	G	H	N	O	X
A		X	X	X	X	X	
E	X		X		X	X	X
G		X		X	X	X	X
H	X	X	X			X	X
N	X	X	X	X			X
O	X	X		X	X		X
X	X		X	X	X	X	

Now we make a new grid with the digrams that are missing in the first grid:

	A	E	G	H	N	O	X
A	X						X
E		X		X			
G	X		X				
H				X	X		
N					X	X	
O			X			X	
X		X					X

Remove all the double letters (on the diagonal).

	A	E	G	H	N	O	X
A							X
E				X			
G	X						
H					X		
N						X	
O			X				
X		X					

Reverse all of the digrams by flipping the grid about the diagonal.

	A	E	G	H	N	O	X
A			X				
E							X
G						X	
H		X					
N				X			
O					X		
X	X						

Since there is exactly one in each row and column, we can now reconstruct the key by chaining the digrams together:

A→G→O→N→H→E→X→A

There are seven ways to choose the starting letter of the key, and we need to try all seven. The correct decryption is, of course, when the key is HEXAGON.

Let’s try a shorter text in order to see how the elimination process works. Take this piece of the same text:

... Triangles, Squares, Pentagons, Hexagons, and other figures, ...

and encode with a Polybius square:

44422411332231154343414511421543351533441122343343231553112234333431133
1434442315422124224542154 3

Encrypt with a cipher clock with plaintext alphabet 12345 and a mixed ciphertext alphabet:

AENXHXGXGXNGOENAHGOXAOEXAEGHGEOXHXAHAEXHXHENHEOAGEONXNGHAEXHEXNG
OXOAOANXNHOANHAHGHOXAOENEXNXNAHG

The first grid of all digrams is this:

	A	E	G	H	N	O	X
A		X	X	X	X	X	
E			X		X	X	X
G		X		X		X	X
H	X	X	X			X	X
N	X	X	X	X			X
O	X	X			X		X
X	X		X	X	X	X	

The second grid only contains the missing digrams that are not on the diagonal:

	A	E	G	H	N	O	X
A	-						X
E	X	-		X			
G	X		-		X		
H				-	X		
N					-	X	
O			X	X		-	
X		X					-

Flip it about the diagonal to reverse all of the digrams:

	A	E	G	H	N	O	X
A		X	X				
E							X
G						X	
H		X				X	
N			X	X			
O					X		
X	X						

There is only one in the `H` column, `NH`, so we can delete the other in its row, `NG`.

	A	E	G	H	N	O	X
A		X	X				
E							X
G						X	
H		X				X	
N			X	X			
O					X		
X	X						

Now there is only one in the `G` column, so we can delete `AE`.

	A	E	G	H	N	O	X
A		X	X				
E							X
G						X	
H		X				X	
N				X			
O					X		
X	X						

This leaves only one in the E column, so we delete HO.

	A	E	G	H	N	O	X
A			X				
E							X
G						X	
H		X				X	
N				X			
O					X		
X	X						

This is what remains:

	A	E	G	H	N	O	X
A			X				
E							X
G						X	
H		X					
N				X			
O					X		
X	X						

With only one digram in each row and column it is now possible to reconstruct the key.

Now let’s see an example in which there is an impasse in the elimination process. Take this shorter bit of the text:

... Triangles, Squares, Pentagons, Hexagons, and other figures, ...

Encode with Polybius:

44422411332231154343414511421543351533441122343343231553112234334 3

Encrypt with the cipher clock:

AENXHXGXGXNGOENAHGOXAOEXAEGHGEOXHXAHAEXHXHENHEOAGEONXNGHAEXHEXNGOX

These are the digrams present:

	A	E	G	H	N	O	X
A		X	X	X		X	
E			X		X	X	X
G		X		X		X	X
H	X	X	X				X
N	X		X	X			X
O	X	X			X		X
X	X		X	X	X		

Here are the nondiagonal missing digrams after the grid is flipped:

	A	E	G	H	N	O	X
A		X	X				
E					X		X
G						X	
H		X				X	
N	X		X	X			
O				X	X		X
X	X						

After the elimination process, we still have a 2×2 block that cannot be reduced:

	A	E	G	H	N	O	X
A			X				
E					X		X
G						X	
H		X					
N				X			
O					X		X
X	X						

There are two possibilities: We can keep `EN` and `OX` or we can keep `EX` and `ON`. For each possibility we reconstruct the key, up to a sevenfold choice of first letter, and have to try them. The best decryption of the plaintext wins. If there had been a 3×3 block of digrams that could not be reduced, then there would be six possibilities that we would need to try, since choosing one digram to keep and one to eliminate (three ways to do this) leaves a 2×2 block. By the same reasoning, an $n \times n$ block requires that we check $n!$ possibilities.

Now we are going to work through three examples that use the twenty-six-letter English plaintext alphabet.

example 1: n = m + 2

Below is a ciphertext. It was encrypted with a cipher clock that uses this plaintext alphabet:

`ABCDEFGHIJKLMNOPQRSTUVWXYZ`

and this ciphertext alphabet, before it is mixed:

`ABCDEFGHIJKLMNOPQRSTUVWXYZ◊○`

And here it is . . .

```
XAYIHVBS○URMXE◊○WBQPSTGXECXCXPA◊OCATLXVBEQSZYB◊DTAJSNEKGJDVWEAXA
◊GBNGQWJQXOJEGPXJLFXFESFMWDOWEKEJFPWE◊CABSYUWSKOLZDMH○BIBYTJQYUE
CVAUEOCSPKWLGM◊HIEWNEJKCUBSXMSEQVALMNTUBSNBMD○OJEQGADBPGYPHYQJVH
BHIBTDKTAPCEUMI○WEHNGZIDVQXZENWAQBPFHJKMSYOBUORLGNBLZ◊○QAWPRIEOV
Y○◊RVYUFCHO◊UZJ○DVUNAT○NF◊UPXKOPYRSZ◊STJHKNSMPRDVXTGEKITAPLUEJ○O
VR○SUPJPZKGNGDWCXIPJDIXCPBYOQMSLZUJVNXED○QG○WENUGCN◊OFEPATLMPDWL
CDEOATAP◊D○Z○CFORQLUZPXQ○ZGOKTA◊NKMPG○MQLZUJVNB◊ADBNALQ◊E◊KLND○X
UF◊BL○E◊GOBHDA◊DTAGXPXIYUVBCYOJ◊TQD○WDLIO○NJKB○TDRUZRNG◊H○JW○PSE
◊NRD○XUSLHZSRNXSDXZGKIJEUYXCPVOMHGHSLNULRZPU○UNO◊EGEAPEXUFWEACWJ
YIEMLHJXNOWLIQYXKVXV○GRCMPYKOJDP◊DJZGVPOVUBIQGAZIHUYX◊RLWBSL◊DSG
TBEV○PSKEDLGRLORQYVJQWBGBZMDAWCUQ○IKET◊PISD◊QPVSZGTLTDZODTUO○MEK
O◊FXOMXCOGZIDVUNAXKRENOQ◊FHVAW◊UIVPSKGJVRNGQVMDJYBPQBQ◊FHOMSFECW
ORYQSZLMK○ZFPSIJQCEHXSEDHNJQJGZOBPSBQMGUJOMSFECWORYQSZLMK○ZFPSIJ
QCEHXUJ◊MHMPAEGBNXEIDUWCDK○OQYXKMSIVOPBZKQDRGWZSHBYTAPLXB◊CSF◊SM
AYMEJPTHJWGNBYN○UCZMBL○WCSMTGYJOUBHJQZF○ILIESECK◊FXIVRXMSENJGQJQ
```

```
DVXECSJU○UKSPAUFBOYVJVJMRVJ○BWCUQKXEJPA◊ELVJS○QR◊VSEASOGUIMCGDEU
YXVYJYLNETQEZO◊PQRO○NKMXZOSVASNKLZJFAPWCE○FJVGMKMXDZ◊RXMSNSMJNB◊
XB○JEMNDI◊YSKWAP◊SRU◊H○JABHNQEOLEDASYTCPYOSGMIVGOJQAKUCLXJRNUCPA
IHVBDU◊LMR○OTLGIZUF○KGXNRDLUMIDZLCOBGZIJHGCYPEGTJ◊DUAVAVH○MJRJEY
QDRGLHXT○BVF◊KSREJCXC◊RV○RLGBQNBYTAPDNDKMXSLHUJHKHJBDGIUNTWBV○KT
AJVGMK◊AUYFZSNOGKRHBTXUJNKDT◊JRFEUMIHOBXAEILWTCZSJ○KTAPGIFI◊GD◊F
S○VZLZGJBTCXB○JEMNDUPKXEVL○E○MDWDGHIEWTAPGAZVR○SUGL○EYSWTBI○JZFA
VFV◊N◊R◊FSFPJOWJ◊ILMIXVZOPBPS◊PA◊○KBSZ◊GPELOROXOHYNEHTJZ◊T○J◊TPK
VZDBPD◊OJHEJDHDGQEAWGNBJ◊OPCMKBR○IVSCOHYQJUGQ○XZKDGEGXGAZVR○SUXA
EYGZIULT○XNXBPJNFUI○UTGLGKOYBLJWMKDFJWBVDRE◊FHVYBLSCSZSNK◊XHQDEA
GNBTJDVZUZF○EUVYPKBSDJXAGAYSLUZG○VIAOHESYJT○HIWOXBCLVQNR◊AXUXZNP
BRQ◊XFHIEWDA◊QUDHIXYFZS○JYSPKBDHJSNOACE◊KECUIHVYDNHVJDWXHMKDYPGE
N◊FHJBEVCXUOELYJWMKDTETDSAUCUIECXF◊CXPHSZLV○KVUXBJ○EKPKDHLH○PX○T
GWTK○HBLXECTJO◊VZSVAWQJAUSZKXHSIJOZF○EW◊UNKP○XQJUJEYLMNPSWIWBUSP
FJUBDVZVFTXCFEGWP◊SNBMDWBC○QNPHOBJRTBPUZJHCGQJBZ○DTQYXCKMXDGRMPY
JRLRNG◊DJZVLICVHEJKPAVF◊EM○VYQDUBDXUJWBYDUGYXFCOFEBLGJHUYQJHLT◊D
TETPRI◊PJFC◊VF◊RVXAHUY◊WQEUGWCJ◊EIAGVB◊AIDORQYOQZLGMLVQXYXTXBQYN
RDJ◊BZ○FZSY◊LZOTLMDAPILIPYUI◊HBFSR○RVAV◊L○OMKVUZENUAQBPFHDIVXDJS
UYFOROSXMATLGXPZ◊DSCLUXUIMPKECUGTLRXSTYGZIF○PEWPIFYGOIJVAWDVL○UZ
POWDLHEXZT○EQLXCZDJUVCOPBHYMBZBZHXGEUGPNDW◊HSE◊MXJRBSRNDGZH○JQCP
E○SUIMNDKVEXUFCPTLDHMB◊V◊GW○UAPXNSZPGSAYPVHYXJWMLFZF○◊Y◊○CUZI○AC
VBSMAYQJVUGQMDXBPECUQ○NJVXVGBCP◊LIUWSZYXTQYUJ◊LXCVCWOUZT○IZJBNYS
BDIPSXCZTXLEMNYQJVLIEANOKEBQYXFCXYVQWPDOALM○NGTJFPYBLIPGCKMJDWEA
LHPHOROVGHIEWDYJNDUZSYOS○DLXMAJHJL○OMK○CDECD○KTFPDXCOMPVHC◊YSMOK
SZRXLTHJOE○UPKRUCGLUXBEAORXPHEQSHRXMKL◊IS◊TDUG○PCDIVAI○WZ◊BMJ◊LS
ROUKL◊I◊LOXQTJUGCX◊SR◊CPYTLQDUBZDEVLI◊FHFLFAEBSTPY○UK◊OWJLTDJZF○
QP◊LUEHXVMCWJYQJNB◊○WGBCTQGKUGLPFXOASABRZ◊KMKDZTXSTAPYXJFXAEBSNG
ZIB◊IBFPSIPQUIJBJ◊HFCPNVEJ◊YJHSEXBEQEQ○RVKXYVFSY○U○OQRMVWDQBWQUQ
PY○LNR◊FXEOJWJALFJGT◊SL○TABIE○FNREJKVXPIV○RTXNOHJLSZJBDKVHIBTAIU
QNUIPA◊ORQRZLDUBTHZOXSMJQYXCZPIEOJWJALSXHDHMGCJQZF○ME○BIFECUSYOT
QXAVUZDTAG◊HTBTCUKOQGDLFZC◊SMNEQSHFJMKL◊○XGWJUXSURVKX◊CEGMAUILWO
SGAWOWCENO◊FHJKPYR◊EMZDVZ◊○NOVB◊HIBY○UY○PFNA◊○SY
```

The first step in the attack is to tabulate all of the digrams in the ciphertext. Remember to count *all* of them. The first four letters are `XAYI`; from them we get `XA` and `AY` and `YI`; don't forget that one in the middle. We don't need to count the digrams, just make a table of all the ones that exist.

		CA	DA	EA	FA	GA		IA	JA			MA	NA	OA	PA	QA		SA	TA	UA	VA	WA	XA			◊A	○A
AB			DB	EB	FB	GB	HB	IB	JB	KB		MB	NB	OB	PB	QB	RB	SB	TB	UB	VB	WB	XB	YB	ZB	◊B	○B
AC	BC			EC	FC	GC	HC	IC	JC	KC	LC	MC		OC	PC	QC	RC	SC	TC	UC	VC	WC	XC		ZC	◊C	○C
AD	BD	CD		ED		GD	HD	ID	JD	KD	LD	MD	ND	OD	PD	QD	RD	SD	TD	UD	VD	WD	XD	YD	ZD	◊D	○D
AE	BE	CE	DE		FE	GE	HE	IE	JE	KE	LE	ME	NE	OE	PE	QE	RE	SE	TE	UE	VE	WE	XE		ZE	◊E	○E
	BF	CF	DF				HF	IF	JF		LF		NF	OF	PF		RF	SF	TF	UF	VF		XF	YF	ZF	◊F	○F
AG	BG	CG	DG	EG			HG		JG	KG	LG	MG	NG	OG	PG	QG	RG	SG	TG	UG	VG	WG	XG	YG	ZG	◊G	○G
AH	BH	CH	DH	EH	FH	GH		IH	JH	KH	LH	MH	NH	OH	PH		RH	SH	TH		VH		XH	YH	ZH	◊H	○H
AI	BI		DI	EI	FI	GI	HI			KI	LI	MI		OI	PI		RI	SI		UI	VI	WI	XI	YI	ZI	◊I	○I
AJ	BJ	CJ	DJ	EJ	FJ	GJ	HJ	IJ			LJ	MJ	NJ	OJ	PJ	QJ	RJ	SJ	TJ	UJ	VJ	WJ	XJ	YJ	ZJ	◊J	○J
AK		CK	DK	EK		GK	HK	IK	JK			MK	NK	OK	PK	QK		SK	TK	UK	VK		XK	YK	ZK	◊K	○K
AL	BL	CL	DL	EL	FL	GL	HL	IL	JL	KL		ML		OL	PL	QL	RL	SL	TL	UL	VL	WL	XL	YL	ZL	◊L	○L
	BM	CM	DM	EM	FM	GM	HM	IM	JM	KM	LM			OM		QM	RM	SM		UM	VM	WM	XM	YM	ZM	◊M	○M
AN	BN	CN	DN	EN	FN	GN	HN		JN	KN	LN	MN			PN	QN	RN	SN		UN	VN	WN	XN	YN	ZN	◊N	○N
AO	BO	CO	DO	EO	FO	GO	HO	IO	JO	KO	LO	MO	NO		PO		RO	SO		UO	VO	WO	XO	YO	ZO	◊O	○O
AP	BP	CP	DP	EP	FP	GP	HP	IP	JP	KP	LP	MP	NP	OP		QP		SP	TP	UP	VP	WP	XP	YP	ZP	◊P	○P
AQ	BQ		DQ	EQ		GQ	HQ	IQ	JQ	KQ	LQ	MQ	NQ	OQ	PQ		RQ		TQ	UQ	VQ	WQ	XQ	YQ		◊Q	○Q
	BR		DR			GR	HR		JR	KR	LR	MR	NR	OR	PR	QR		SR		UR	VR			YR	ZR	◊R	○R
AS	BS	CS	DS	ES	FS	GS	HS	IS	JS	KS	LS	MS	NS	OS	PS	QS	RS			US	VS	WS	XS	YS	ZS	◊S	○S
AT	BT	CT	DT	ET	FT	GT	HT	IT	JT	KT	LT	MT	NT	OT	PT	QT	RT	ST		UT		WT	XT	YT	ZT	◊T	○T
AU	BU	CU	DU	EU	FU	GU	HU	IU	JU	KU	LU		NU	OU	PU	QU	RU	SU	TU		VU		XU	YU	ZU	◊U	○U
AV	BV	CV	DV	EV	FV	GV	HV	IV	JV	KV	LV	MV	NV	OV	PV	QV	RV	SV		UV			XV	YV	ZV	◊V	○V
AW	BW	CW	DW	EW	FW	GW		IW	JW	KW	LW	MW	NW	OW	PW	QW		SW	TW	UW	VW					◊W	○W
AX	BX	CX	DX	EX	FX	GX	HX	IX	JX	KX	LX	MX	NX	OX	PX	QX	RX	SX	TX	UX	VX	WX		YX		◊X	○X
AY	BY	CY	DY	EY	FY	GY	HY	IY	JY		LY		NY	OY	PY	QY	RY	SY	TY	UY	VY		XY		ZY	◊Y	
AZ	BZ	CZ	DZ	EZ	FZ	GZ	HZ	IZ	JZ		LZ	MZ		OZ	PZ	QZ	RZ	SZ		UZ	VZ	WZ	XZ				○Z
A◊	B◊	C◊	D◊	E◊	F◊	G◊		I◊	J◊	K◊	L◊	M◊	N◊	O◊	P◊	Q◊	R◊	S◊	T◊	U◊	V◊	W◊	X◊	Y◊	Z◊		○◊
	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○	◊○	

What is important to us now is which digrams are *missing*. Of course, all of the diagonal entries are missing, since double letters cannot occur in the ciphertext for a device with this configuration. But look at the `J` column. The digram for `JI` is the only other one missing. Therefore, we know that `I` comes immediately before `J` in the key. Here are all of the missing digrams, except for the doubles:

	BA						HA			KA	LA						RA							YA	ZA		
		CB									LB																
			DC										NC											YC			
					FD																						
																								YE			
AF				EF		GF				KF		MF				QF						WF					
					FG			IG																			
																QH				UH		WH		YH			
		CI							JI				NI			QI			TI								
										KJ																	
	BK				FK						LK						RK					WK					
													NL														
AM													NM		PM				TM								
								IN						ON					TN								
																QO			TO								
																	RP										
		CQ			FQ													SQ							ZQ		
AR		CR		ER	FR			IR											TR			WR	XR				
																			TS								
																					VT						
												MU										WU					
																			TV			WV					
							HW										RW						XW	YW	ZW		
																									ZX		
										KY		MY										WY					○Y
										KZ			NZ						TZ					YZ		◊Z	
							H◊																				
A○																											

Now don’t get confused, but to make things easier, we reverse each missing digram so that it is written in the way it might appear in the key. Here is the new table:

	AB						AH			AK	AL						AR							AY	AZ		
		BC									BL																
			CD										CN											CY			
					DF																						
																								EY			
FA				FE		FG				FK		FM				FQ						FW					
					GF			GI																			
																HQ				HU		HW		HY			
		IC							IJ				IN			IQ			IT								
										JK																	
	KB				KF						KL						KR					KW					
													LN														
MA													MN		MP				MT								
								NI						NO					NT								
																OQ			OT								
																	PR										
		QC			QF													QS							QZ		
RA		RC		RE	RF			RI											RT			RW	RX				
																			ST								
																					TV						
												UM										UW					
																			VT			VW					
							WH										WR						WX	WY	WZ		
																									XZ		
										YK		YM										YW					Y○
										ZK			ZN						ZT					ZY		Z◊	
							◊H																				
○A																											

Now we look for any digram that appears alone in a row or column. Such a digram must be in the key. We find CD, FG, IJ, NO, MP, QS, HU, TV, Z◊, Y○, DF, EY, JK, LN, PR, ST, XZ, ◊H, and ○A. Once we decide to keep a digram in a row, we can eliminate the other digrams in its column; similarly, once we keep a digram in a column, we can eliminate the others in the same row. We can also eliminate the reversal of any digram that we keep. Here is what is left after we make those deletions:

	AB										AL															
		BC									BL															
			CD																							
					DF																					
																							EY			
						FG																				
								GI																		
																				HU						
									IJ																	
										JK																
	KB										KL											KW				
													LN													
															MP											
														NO												
																OQ										
																	PR									
																		QS								
		RC		RE				RI														RW	RX			
																			ST							
																					TV					
												UM										UW				
																						VW				
																							WX			
																								XZ		
																										Y○
																									Z◊	
							◊H																			
○A																										

Now we play the same game and look for digrams that are alone on a row or column in what remains in the table. We find `GI`, `OQ`, `UM`, `VW`, and `WX`. Eliminate those rows and columns to get an even smaller table:

	AB										AL																
		BC									BL																
			CD																								
					DF																						
																								EY			
						FG																					
								GI																			
																				HU							
									IJ																		
										JK																	
	KB										KL																
													LN														
															MP												
														NO													
																OQ											
																	PR										
																		QS									
		RC		RE																							
																			ST								
																					TV						
												UM															
																						VW					
																							WX				
																									XZ		
																											Y○
																										Z◊	
							◊H																				
○A																											

Now we get RE and eliminate RC. One more time:

	AB										AL																
		BC									BL																
			CD																								
					DF																						
																								EY			
						FG																					
								GI																			
																				HU							
									IJ																		
										JK																	
	KB										KL																
													LN														
															MP												
														NO													
																OQ											
																	PR										
																		QS									
				RE																							
																			ST								
																					TV						
												UM															
																						VW					
																							WX				
																									XZ		
																											Y○
																										Z◊	
							◊H																				
○A																											

Now we keep BC and eliminate BL. One more elimination won’t gain us anything, so we stop.

Here are all of the key’s digrams that we have found to definitely belong:

```
BC CD DF EY FG GI HU IJ JK LN MP NO OQ
PR QS RE ST TV UM VW WX XZ Y○ Z◊ ◊H ○A
```

But remember that AB, AL, KB, and KL are still possibilities. If we chain the definte key digrams together, we get these key fragments:

```
BCDFGIJK
LNOQSTVWXZ◊HUMPREY○A
```

There are two ways to combine them, but it doesn’t matter, since the key goes on a ring. So now we know that the key is something like

```
LNOQSTVWXZ◊HUMPREY○ABCDFGIJK
```

To find the final key, we need to roll this key to each possible position on the ring and decipher the ciphertext. When we find a recognizably English plaintext, then we have found the correct key. It turns out that the correct key is

```
◊HUMPREY○ABCDFGIJKLNOQSTVWXZ
```

and the plaintext (broken into words) is

```
ALL THESE SORROWS ARE PAST MY GLANCING AT THEM MAY NOT BE
WITHOUT ITS USE FOR IT MAY HELP IN SOME MEASURE TO EXPLAIN WHY I
HAVE ALL MY LIFE BEEN ATTACHED TO THE INANIMATE OBJECTS THAT
PEOPLE MY CHAMBER AND HOW I HAVE COME TO LOOKUP ON THEM RATHER
IN THE LIGHT OF OLD AND CONSTANT FRIENDS THAN AS MERE CHAIRS AND
TABLES WHICH A LITTLE MONEY COULD REPLACE AT WILL CHIEF AND
FIRST AMONG ALL THESE IS MY CLOCK MY OLD CHEERFUL COMPANIONABLE
CLOCK HOW CAN I EVER CONVEY TO OTHERS AN IDEA OF THE COMFORT AND
CONSOLATION THAT THIS OLD CLOCK HAS BEEN FOR YEARS TO ME IT IS
ASSOCIATED WITH MY EARLIEST RECOLLECTIONS IT STOOD UPON THE
STAIRCASE AT HOME I CALL IT HOME STILL MECHANICALLY NIGH SIXTY
YEARS AGO I LIKE IT FOR THAT BUT IT IS NOT ON THAT ACCOUNT NOR
BECAUSE IT IS A QUAINT OLD THING IN A HUGE OAKEN CASE CURIOUSLY
AND RICHLY CARVED THAT I PRIZE IT AS I DO I INCLINE TO IT AS IF
IT WERE ALIVE AND COULD UNDERSTAND AND GIVE ME BACK THE LOVE I
BEAR IT AND WHAT OTHER THING THAT HAS NOT LIFE COULD CHEER ME AS
IT DOES WHAT OTHER THING THAT HAS NOT LIFE I WILL NOT SAY HOW
FEW THINGS THAT HAVE COULD HAVE PROVED THE SAME PATIENT TRUE
UNTIRING FRIEND HOW OFTEN HAVE IS AT IN THE LONG WINTER EVENINGS
FEELING SUCH SOCIETY IN ITS CRICKET VOICE THAT RAISING MY EYES
FROM MY BOOK AND LOOKING GRATEFULLY TOWARDS IT THE FACE REDDENED
BY THE GLOW OF THE SHINING FIRE HAS SEEMED TO RELAX FROM ITS
STAID EXPRESSION AND TO REGARD ME KINDLY HOW OFTEN IN THE SUMMER
TWILIGHT WHEN MY THOUGHTS HAVE WANDERED BACK TO A MELANCHOLY
PAST HAVE ITS REGULAR WHISPERINGS RECALLED THEM TO THE CALM AND
PEACEFUL PRESENT HOW OFTEN IN THE DEAD TRANQUILLITY OF NIGHT HAS
ITS BELL BROKEN THE OPPRESSIVE SILENCE AND SEEMED TO GIVE ME
ASSURANCE THAT THE OLD CLOCK WAS STILL A FAITHFUL WATCHER AT MY
CHAMBER DOOR MY EASY CHAIR MY DESK MY ANCIENT FURNITURE MY VERY
BOOKS I CAN SCARCELY BRING MYSELF TO LOVE EVEN THESE LAST LIKE
MY OLD CLOCK IT STANDS IN A SNUG CORNER MIDWAY BETWEEN THE
FIRESIDE AND A LOW ARCHED DOOR LEADING TO MY BEDROOM ITS FAME IS
DIFFUSED SO EXTENSIVELY THROUGHOUT THE NEIGHBOURHOOD THAT I HAVE
OFTEN THE SATISFACTION OF HEARING THE PUBLICAN OR THE BAKER AND
SOMETIMES EVEN THE PARISH CLERK PETITIONING MY HOUSEKEEPER OF
WHOM IS HALL HAVE MUCH TO SAY BY AND BY TO IN FORM HIM THE EXACT
TIME BY MASTER HUMPHREYS CLOCK MY BARBER TO WHOM I HAVE REFERRED
WOULD SOONER BELIEVE IT THAN THE SUN NOR ARE THESE ITS ONLY
DISTINCTIONS IT HAS ACQUIRED I AM HAPPY TO SAY ANOTHER
INSEPARABLY CONNECTING IT NOT ONLY WITH MY ENJOYMENTS AND
REFLECTIONS BUT WITH THOSE OF OTHER MEN
```

You might recognize the text from *Master Humphrey's Clock* by Charles Dickens.

example 2: n = m + 3

For this example, we need to add another character to the ciphertext alphabet. This time, for each character in the key, we should find two missing digrams for the two characters that follow it. In which order those two occur can be deduced from the choices of what characters might follow them. For example, if we have missing digrams `AB`, `AC`, `BC`, and `BD`, then we know that `B` and `C` are after `A`. Since `C` is after `B`, the order is `ABCD`.

Here is a ciphertext for us to break:

```
JEY@VHRIECWIBIRMBSVECN*LOTHSDIZ*DLPAF*LHUN@XCFXLACYUTHBPTCTKVDEN
VWEXBVKMVGK#WG*GJVYSI@WAHOMY@YDZQ#JTERJBLHMRBQCXGUAMUBHAPZG@AV#W
SPDINPX#HEKEHVLSQID#HVLV@WJZMUHYIMTCWH@CXKVDYOWUEQZGRN#CA#QVECN*
ISCVBL@FME#AWOA#TZVMPOZR*CHXKICFSEHXZL@WELWKVW#W@SYOQV*HPIPQAFTX
GJV*#MDHVSWLXLU@PHNKIJPBPXPZDJOTIXLRD#HCEIZAOJEBGSOBOXFYPOMERWYS
TFMKXRLYSP#@PBZQIB*B#GL*IJXGBVE*QYRSAXYJCGQGWOWSWEO#YUAHVNFSA#HD
PQFMRG*DHSKPGPIXGXLGERSCNF*SATJWNHEWXJPJFSI@CUR*VFKQXLQ*#IJUZBCW
DHLXSMTJUXFSQRI#FUPM*BMAXYPQOBUZNCLSMROPRW@WJYAWRGTLPG@RIYUCBYN#
DXFIWUXOYKN#BOXCVOTHUDOIN#DQSDS@AHZLVIUNG*GHYLBPGUEDUEYMANSDFZRS
QIE#Z@BOF*MDR#BXBITBYJSDSHZ#PXUNTRLJOZBCQCLBCFLWZKW#IQEIAWRLUFR*
BCBLENRKN*NX@SMA#VWK@IJ*TEJBO@DTFODY@DYRAVYNHZF*NQYFIJY#FGQR#XQW
EJPQCBJD*NGS@U#HMOADZWXC@QTXGUTYZ#TWN*VRK*MEFCNQYHLYRGBN#*UBXJOJ
EHKQVAFXQA#*W*K@PX#*BEMXHBRKGLBLFWSMEWKRVEOZIBQS#MDV#IYS#D@HWEHE
TUZRB*@DX*XQBKNWKEADIJAY@WNGESGMOBCN#*BXFTCMBXLBIAORIU@XZPLD*EQG
JRLGBQZBPQU@DZCTCTKVIPRV#QZSJWBUX*VCP@V*FHBI*PCWX@RJN@LOD@WJDTX@
JZBPQW*BJSXAVNWSMEW#QZK#PKIPQAYN#MK*OIN#DLYBKWRQIRXFKWPJT*HJSDLW
H#IBWNLKUMNBGIPXTYQT#MYFCMAHQTSDMG#OH*@DPOXFKXKPBLJFSVHRCDYD@GOB
VKMCAIPED@RVENPVA#@AFEA#LZ*N*@WXUTRSHQZKAT@QYJETBRSIXZUVGN@*GQRW
F*TCDSCSFCJAFZSIAILVGPOTZRJBV#VN*#SCUAOFULUBIJQ@WOIUFUWSFXKFJCJO
NZDRNYNKYKCJE#ALQSXJKSHTHURK*VLWSCNFOR#*MPT*@BCUVRHXYP*NOINCUHD#
P#IJPRJBLMPDNW*NSTBUH#CJZV@RLB#QVCIPF*TKZDSDNIESDSYZKNO@TP#LQG*O
HMU#SNQ*JR@CZENOIUIL*OP#PAFJEFXJMSM#YBDOFCLGZJUO*HZ*MWXCLQGVX*BH
XT@Q*JRK*BRL@SCKZTGJOTUNIPXAERSILVSVWJRUYNRNLQGWUT@DJ#WCSKNLDLND
GNEIAXYRVKGAIPQAHEKGEHPHY@LMVSIMUFGOXIUQ*#*UTMC*HLNL@BROKRZQJRGJ
FRK*EDGKTSD#MDPDNFRTYQVLVNFLFTB#CMXQDMGAZMNCNFOQJACAUKYVYIM#BVMF
MKTFJACWP#MFXMAHTZ#J#QCSKGLAGMXTIUBLWOG#MHCMCPOS#MDWIET#FIKEFK@X
ULTYGUTYKCQCSG@MZETUZRSLXKJQVOPBIPXKNSGRUVA#FIP@JMRZT#SY*WX*OMU#
SNWH@CXJZUGVPEIAPRQ#OZGLETLRHXYEGHRNY*AHEMRZCFCILRJBVXMNCMLBJSDI
TRAMEMUYBFWZSEGCOUVP*#MCBRK#PH@JVACLMY@YAWYA*FRGUGKIPNYDMZTRUWCD
YD#QZHKR#IRG@AJLAMYVPYRXINRCNOVDWI*PZFXQZMFUXGVDKFULHZ@QJYABTFGN
EGKRJPDRHVFIZDJNXAWRJZIDXOPYASBQYKMLPMGNBRKVIBQWKIBVMPKSMCBFJPO#
YF@LFTLYHMUN@QKJPVOEBPIFSKNLDYHOIDWNOVOPGEZ*AOTV*J@RFXEHEMRVDR#J
*FYG@#JPQJXZEGLIR@#ZGLWRTBGB#S#WOSNPFVYCMUNRMNFL*KMPZDG*QK#QCXGN
LDENOIUDZARHJCUBEOZJPJFYTZ@TYLHTAWQOFNOQUPLITEAYVM@GQE*NLDMOYKN@
SYA#NC@*EDQ#QOTHJNSNY@QFOUGMZ#P@WJDTX@JDJWKE*RKUCXLUZT*LPTUIQZGI
*BLJLO#OY@SXUIVSIU@AHYZTW*BQIBCZBDVSHTKBDQGSMKH#DX@SYOQGESQVAYOZ
CUP@DZNGRTRZAWJQL@JGWQDJC*B*GYCZEKVIFPANBILZAZBXRTUHMENXG*DETZFN
@AHPTWNFEADPXDMXNULHZH#ABMEJNCN@SYA#SKBY#IQ@E*VLFVY*RJAHVMUGKNK@
TPOPJXPXY#AMSHTARLXRIMTPWRIQE*ANOQLKPHWOWKTLSCSADZCMCXF@YZAZ#DXL
I*GSHZDISNE@UALAON#P@BQWKIYAUNEBGYKRLSVFK*@CVD*ALGWR@XPZATHZE#J#
DLPEGSVHRCIDJV@ATOIR#FSPYKGEQYLTSCVB*DFMDCOQUBO*R@CL*AWCZBDJOYEJ
*RFWLFVLNHQYEIACAQGABLXACVFKNIAHRYGVCWOQLTYGYHKRC@Q@STFVO*LGNEWO
ALOTKJPZTBOISXUIK@VLEYGVBWPTFHRMWFLMGTB*DH#ZKW#JRQJQKBIV#JPHIWOW
```

```
DJCVHPLIDLSISA*NEIUPZDGRGTBP#*U@DZCGSMKVIR*ALVGMZTIQ@EN*SJYVKXIS
DWALWQAHJSRJMVSBM@ECUXHAVYNKDPEIAQIBX@WOI*CBPJMER#YVKFX@RJY#YPDH
NKIUJODGTMKJWRDHUKXKXLENCN@S#QZCJWBUIPKBCBWXSDJMUHXBSQIDZIDJ#ADI
ULJFUTGJMZ@QJ@SMAN#CLHQF@GYTMXHI*LQ#CBORNBDYJACLMZUZNKX*BYMSMK#S
VRGZL@ZEOKHSGINFEQZGRMZTI*TPV#UPYFHLQ#OPHZWRIUIRJBG#J#NF@UCBKUCK
IYEOTGNZV#MDWDFXET@M*NTXY#I*FWEMVSIMUFHOW*VWLQTEADZPOHQZ#JRCNCLF
QRGSBYHCGWZBNTUXYCOQVUKFKTPQDTWY@M*MNWKTCKZEHTSCUAJZADZQH#*STXEM
ENEHNSDVDKCNFHZARVEOZGLQTCBOZ@FVYAMFWATJFZ#*IJ*J#W@BXZ#DJWS@QWDH
CFLESUXFSFQLYOZI*HXK
```

Here is our table of digrams present in the ciphertext:

		CA		EA		GA	HA	IA	JA	KA	LA	MA		OA	PA	QA	RA	SA	TA	UA	VA	WA	XA	YA	ZA	*A	#A	@A
AB		CB		EB		GB	HB	IB	JB	KB	LB	MB	NB	OB	PB	QB	RB	SB	TB	UB	VB	WB	XB	YB	ZB	*B	#B	@B
AC	BC		DC	EC	FC	GC	HC	IC	JC	KC		MC	NC		PC	QC	RC	SC	TC	UC	VC	WC	XC	YC	ZC	*C	#C	@C
AD	BD	CD		ED			HD	ID	JD	KD	LD	MD	ND	OD	PD	QD	RD	SD		UD	VD	WD	XD	YD	ZD	*D	#D	@D
AE	BE	CE	DE		FE	GE	HE	IE	JE	KE	LE	ME	NE	OE	PE	QE		SE	TE	UE	VE	WE	XE	YE	ZE	*E	#E	@E
AF	BF	CF	DF	EF				IF	JF	KF	LF	MF	NF	OF	PF	QF	RF	SF	TF	UF	VF	WF	XF	YF	ZF	*F		@F
AG	BG	CG	DG	EG	FG				JG	KG	LG	MG	NG	OG	PG	QG	RG	SG	TG	UG	VG	WG	XG	YG	ZG	*G	#G	@G
AH	BH	CH	DH	EH	FH	GH				KH	LH	MH	NH	OH	PH	QH	RH	SH	TH	UH	VH	WH	XH	YH	ZH	*H	#H	@H
AI	BI	CI	DI	EI	FI	GI	HI			KI	LI		NI	OI	PI	QI	RI	SI	TI	UI	VI	WI	XI	YI	ZI	*I	#I	@I
AJ	BJ	CJ	DJ	EJ	FJ	GJ	HJ	IJ		KJ	LJ			OJ	PJ	QJ	RJ	SJ	TJ	UJ		WJ	XJ	YJ	ZJ	*J	#J	@J
	BK	CK	DK	EK	FK	GK	HK	IK	JK		LK	MK	NK	OK	PK	QK	RK	SK	TK	UK	VK	WK	XK	YK	ZK	*K		
AL	BL	CL	DL	EL	FL	GL	HL	IL	JL			ML	NL		PL	QL	RL	SL	TL	UL	VL	WL	XL	YL	ZL	*L	#L	@L
AM	BM	CM	DM	EM	FM	GM	HM	IM	JM	KM	LM			OM	PM		RM	SM	TM	UM	VM		XM	YM	ZM	*M	#M	@M
AN	BN	CN	DN	EN	FN	GN	HN	IN	JN	KN	LN	MN		ON	PN		RN	SN		UN	VN	WN	XN	YN	ZN	*N	#N	
AO	BO	CO	DO	EO	FO	GO	HO		JO		LO	MO	NO		PO	QO	RO	SO	TO	UO	VO	WO	XO	YO		*O	#O	
AP	BP	CP	DP		FP	GP	HP	IP	JP	KP	LP	MP	NP	OP				SP	TP	UP	VP	WP	XP	YP	ZP	*P	#P	@P
AQ	BQ	CQ	DQ	EQ	FQ	GQ	HQ	IQ	JQ	KQ	LQ		NQ	OQ	PQ		RQ	SQ		UQ		WQ	XQ	YQ	ZQ	*Q	#Q	@Q
AR	BR		DR	ER	FR	GR	HR	IR	JR	KR	LR	MR	NR	OR	PR	QR		SR	TR	UR	VR	WR	XR	YR	ZR	*R		@R
AS	BS	CS	DS	ES	FS	GS	HS	IS	JS	KS	LS	MS	NS	OS		QS	RS		TS		VS	WS	XS	YS	ZS	*S	#S	@S
AT	BT	CT	DT	ET	FT	GT	HT	IT	JT	KT	LT	MT	NT	OT	PT	QT	RT	ST		UT			XT	YT	ZT	*T	#T	@T
AU	BU	CU	DU		FU	GU	HU	IU	JU	KU	LU	MU	NU	OU		QU	RU	SU	TU		VU	WU	XU	YU	ZU	*U	#U	@U
AV	BV	CV	DV		FV	GV	HV	IV	JV	KV	LV	MV	NV	OV	PV	QV	RV	SV	TV	UV				YV	ZV	*V	#V	@V
AW	BW	CW	DW	EW	FW	GW	HW	IW	JW	KW	LW	MW	NW	OW	PW	QW	RW	SW	TW	UW	VW				ZW	*W	#W	@W
AX	BX	CX	DX	EX	FX	GX	HX	IX	JX	KX	LX	MX	NX	OX	PX	QX	RX	SX	TX	UX	VX	WX				*X	#X	@X
AY	BY	CY	DY	EY	FY	GY	HY	IY	JY	KY	LY	MY	NY	OY	PY	QY	RY	SY	TY	UY	VY	WY	XY				#Y	@Y
AZ	BZ	CZ	DZ	EZ	FZ	GZ	HZ	IZ	JZ	KZ	LZ	MZ	NZ	OZ	PZ	QZ	RZ		TZ	UZ		WZ	XZ	YZ			#Z	@Z
A*	B*	C*	D*	E*	F*	G*	H*	I*	J*	K*	L*	M*	N*	O*	P*	Q*	R*		T*		V*	W*	X*	Y*	Z*		#*	@*
A#	B#		D#	E#		G#	H#	I#	J#	K#		M#	N#	O#	P#	Q#	R#	S#	T#	U#	V#	W#	X#	Y#	Z#	*#		@#
		C@	D@	E@	F@	G@	H@	I@	J@	K@	L@	M@	N@	O@	P@	Q@	R@	S@	T@	U@	V@	W@	X@	Y@	Z@	*@	#@	

Here are the missing digrams, reversed:

	AB		AD		AF								AN																
			BD		BF																								
											CL			CO															
					DF	DG													DT										
																	ER											E#	
						FG	FH																						
							GH	GI																					
								HI	HJ																				
									IJ			IM																	
												JM	JN								JV								
KA																												K#	K@
										LK				LO															
													MN			MQ						MW							
																NQ			NT										N@
								OI		OK															OZ			O@
				PE												PQ	PR												
												QM							QT		QV								
		RC																										R#	
															SP					SU									
																					TV	TW							
				UE											UP														
				VE																		VW	VX						
																							WX	WY					
																								XY	XZ				
																									YZ	Y*			
																		ZS			ZV					Z*			
																		*S		*U									
		#C			#F						#L																		
@A	@B																												

This time, we are looking for rows and columns that have exactly *two* digrams (some are in both), since there are two characters that we cannot reach when we encipher the next letter. From rows we get BD, BF, CL, CO, ER, E#, FG, FH, GH, GI, HI, HJ, IJ, IM, LK, LO, RC, R#, SP, SU, TV, TW, UE, UP, WX, WY, XY, XZ, YZ, Y*, *S, *U, @A, and @B. From columns, we add AB, #C, AD, DG, KA, OK, #L, PR, ZS, VX, and Z*. We can eliminate all non-listed digrams from rows and columns that contain two of the digrams in our list, since we know that any other digrams in such a row or column is accidentally rather than necessarily missing. One such column contains GI, HI, and OI; since GI and HI are in our list, we know that OI cannot be. The table becomes

	AB		AD																									
			BD		BF																							
											CL			CO														
					DF	DG													DT									
																	ER										E#	
						FG	FH																					
							GH	GI																				
								HI	HJ																			
									IJ			IM																
												JM	JN								JV							
KA																												K@
										LK				LO														
													MN			MQ						MW						
																NQ			NT									N@
										OK																		O@
				PE												PQ	PR											
												QM							QT		QV							
		RC																									R#	
															SP					SU								
																					TV	TW						
				UE											UP													
				VE																		VW	VX					
																							WX	WY				
																								XY	XZ			
																									YZ	Y*		
																		ZS								Z*		
																		*S		*U								
		#C									#L																	
@A	@B																											

Now there are some new rows and columns with only two digrams. This allows us to add `DF`, `JN`, and `MN` to our list. Moving onward, we can eliminate a few more from rows and columns that already have two listed digrams, such as `DT`.

	AB		AD																									
			BD		BF																							
											CL			CO														
					DF	DG																						
																	ER										E#	
						FG	FH																					
							GH	GI																				
								HI	HJ																			
									IJ			IM																
												JM	JN							JV								
KA																												K@
										LK				LO														
													MN			MQ						MW							
																NQ			NT									N@
										OK																		O@
				PE												PQ	PR											
												QM							QT		QV								
		RC																									R#	
															SP					SU								
																					TV	TW						
				UE											UP													
				VE																		VW	VX					
																							WX	WY				
																								XY	XZ			
																									YZ	Y*		
																		ZS								Z*		
																		*S		*U								
		#C									#L																	
@A	@B																											

Now we can also add `NT` and `QT`. We have reached the end of this path. The key diagrams we have are

```
AB AD BD BF CL CO DF DG ER E# FG FH GH GI HI HJ IJ IM JN KA LK
LO MN NT OK PR QT RC R# SP SU TV TW UE UP VX WX WY XY XZ YZ Y*
ZS Z* *S *U @A @B #C #L
```

We see that `A` is followed by `B` and `D`, and `B` is followed by `D`. So the order must be `ABD`. Using this logic, we can construct this key:

```
@ABDFGHIJMNQTVWXYZ*SUPER#CLOK
```

We have to roll the key and try each option until we get a readable plaintext. The true key that we find is ultimately

```
*SUPER#CLOK@ABDFGHIJMNQTVWXYZ
```

and the plaintext (with spaces added) is

THERE ARE TWO KINDS OF CLOCKS THERE IS THE CLOCK THAT IS ALWAYS WRONG AND THAT KNOWS IT IS WRONG AND GLORIES IN IT AND THERE IS THE CLOCK THAT IS ALWAYS RIGHT EXCEPT WHEN YOU RELY UPON IT AND THEN IT IS MORE WRONG THAN YOU WOULD THINK A CLOCK COULD BE IN A CIVILIZED COUNTRY I REMEMBER A CLOCK OF THIS LATTER TYPE THAT WE HAD IN THE HOUSE WHEN I WAS A BOY ROUTING US ALL UP AT THREE OCLOCK ONE WINTERS MORNING WE HAD FINISHED BREAKFAST AT TEN MINUTES TO FOUR AND I GOT TO SCHOOL A LITTLE AFTER FIVE AND SAT DOWN ON THE STEP OUTSIDE AND CRIED BECAUSE I THOUGHT THE WORLD HAD COME TO AN END EVERYTHING WAS SO DEATHLIKE THE MAN WHO CAN LIVE IN THE SAME HOUSE WITH ONE OF THESE CLOCKS AND NOT ENDANGER HIS CHANCE OF HEAVEN ABOUT ONCE A MONTH BY STANDING UP AND TELLING IT WHAT HE THINKS OF IT IS EITHER A DANGEROUS RIVAL TO THAT OLD ESTABLISHED FIRM JOB OR ELSE HE DOES NOT KNOW ENOUGH BAD LANGUAGE TO MAKE IT WORTH HIS WHILE TO START SAYING ANYTHING AT ALL THE GREAT DREAM OF ITS LIFE IS TO LURE YOU ON INTO TRYING TO CATCH A TRAIN BY IT FOR WEEKS AND WEEKS IT WILL KEEP THE MOST PERFECT TIME IF THERE WERE ANY DIFFERENCE IN TIME BETWEEN THAT CLOCK AND THE SUN YOU WOULD BE CONVINCED IT WAS THE SUN NOT THE CLOCK THAT WANTED SEEING TO YOU FEEL THAT IF THAT CLOCK HAPPENED TO GET A QUARTER OF A SECOND FAST OR THE EIGHTH OF AN INSTANT SLOW IT WOULD BREAK ITS HEART AND DIE IT IS IN THIS SPIRIT OF CHILDLIKE FAITH IN ITS INTEGRITY THAT ONE MORNING YOU GATHER YOUR FAMILY AROUND YOU IN THE PASSAGE KISS YOUR CHILDREN AND AFTERWARD WIPE YOUR JAMMY MOUTH POKE YOUR FINGER IN THE BABYS EYE PROMISE NOT TO FORGET TO ORDER THE COALS WAVE AT LAST FOND ADIEU WITH THE UMBRELLA AND DEPART FOR THE RAILWAY STATION I NEVER HAVE BEEN QUITE ABLE TO DECIDE MYSELF WHICH IS THE MORE IRRITATING TO RUN TWO MILES AT THE TOP OF YOUR SPEED AND THEN TO FIND WHEN YOU REACH THE STATION THAT YOU ARE THREE QUARTERS OF AN HOUR TOO EARLY OR TO STROLL ALONG LEISURELY THE WHOLE WAY AND DAWDLE ABOUT OUTSIDE THE BOOKING OFFICE TALKING TO SOME LOCAL IDIOT AND THEN TO SWAGGER CARELESSLY ON TO THE PLATFORM JUST IN TIME TO SEE THE TRAIN GO OUT AS FOR THE OTHER CLASS OF CLOCKS THE COMMON OR ALWAYS WRONG CLOCKS THEY ARE HARMLESS ENOUGH YOU WIND THEM UP AT THE PROPER INTERVALS AND ONCE OR TWICE A WEEK YOU PUT THEM RIGHT AND REGULATE THEM AS YOU CALL IT AND YOU MIGHT JUST AS WELL TRY TO REGULATE A LONDON TOMCAT BUT YOU DO ALL THIS NOT FROM ANY SELFISH MOTIVES BUT FROM A SENSE OF DUTY TO THE CLOCK ITSELF YOU WANT TO FEEL THAT WHATEVER MAY HAPPEN YOU HAVE DONE THE RIGHT THING BY IT AND THAT NO BLAME CAN ATTACH TO YOU SO FAR AS LOOKING TO IT FOR ANY RETURN IS CONCERNED THAT YOU NEVER DREAM OF DOING AND CONSEQUENTLY YOU ARE NOT DISAPPOINTED YOU ASK WHAT THE TIME IS AND THE GIRL REPLIES WELL THE CLOCK IN THE DININGROOM SAYS A QUARTER PAST TWO BUT YOU ARE NOT DECEIVED BY THIS YOU KNOW THAT AS A MATTER OF FACT IT MUST BE SOMEWHERE BETWEEN NINE AND TEN IN THE EVENING AND REMEMBERING THAT YOU NOTICED AS A CURIOUS CIRCUMSTANCE THAT THE CLOCK WAS ONLY FORTY MINUTES PAST FOUR HOURS AGO YOU MILDLY ADMIRE ITS ENERGIES AND RESOURCES AND WONDER HOW IT DOES IT I MYSELF POSSESS A CLOCK THAT FOR COMPLICATED UNCONVENTIONALITY AND LIGHTHEARTED INDEPENDENCE COULD I SHOULD THINK GIVE POINTS TO ANYTHING YET DISCOVERED IN THE CHRONOMETRICAL LINE AS A MERE TIMEPIECE IT LEAVES MUCH TO BE DESIRED BUT CONSIDERED AS A

```
SELFACTING CONUNDRUM IT IS FULL OF INTEREST AND VARIETY I HEARD
OF A MAN ONCE WHO HAD A CLOCK THAT HE USED TO SAY WAS OF NO GOOD
TO ANY ONE EXCEPT HIMSELF BECAUSE HE WAS THE ONLY MAN WHO
UNDERSTOOD IT HE SAID IT WAS AN EXCELLENT CLOCK AND ONE THAT YOU
COULD THOROUGHLY DEPEND UPON BUT YOU WANTED TO KNOW IT TO HAVE
STUDIED ITS SYSTEM AN OUTSIDER MIGHT BE EASILY MISLED BY IT FOR
INSTANCE HE WOULD SAY WHEN IT STRIKES FIFTEEN AND THE HANDS
POINT TO TWENTY MINUTES PAST ELEVEN I KNOW IT IS A QUARTER TO
EIGHT HIS ACQUAINTANCESHIP WITH THAT CLOCK MUST CERTAINLY HAVE
GIVEN HIM AN ADVANTAGE OVER THE CURSORY OBSERVER BUT THE GREAT
CHARM ABOUT MY CLOCK IS ITS RELIABLE UNCERTAINTY IT WORKS ON NO
METHOD WHATEVER
```

The text is from *Clocks* by Jerome K. Jerome.

example 3: both alphabets mixed

For this example, we use a cipher clock with the twenty-six-letter modern-English plaintext alphabet and this twenty-eight-letter old-English ciphertext alphabet (before mixing):

```
abcdefghijklmnopqrstuvwxyzðþ
```

To make things easier, we also specify that mixing is done like this:

```
keywordðþabcfghijlmnpqstuvxz
```

Both alphabets will be mixed, so after the digram-counting attack, we need to break a monoalphabetic substitution cipher.

Here is the ciphertext:

```
ienbzuvcuvxþðykyozkpujwajbpblqþdgjhvtxðwitskourixmjtgqwdnahtþtkp
evznyðatzjkfwkxzþaðvtnudhaoþdbþcjuxdmðpkefigyoctdnþpazbifwxhvþpð
ynrdekzdzbðeuhðrxðpinojwzðefwmbmvdujknizpðcucðhlmnlbmdxixjwhdxkp
amþceufbqlcigvjuxzlhqjhticlacxubrcienþdbuzmjsgarsoenupzvsvrtpqwj
nðaizbnvtxqjosotarctauðinzikefgþxnvtxrjþuctðvtðadedofkezhxþvpuko
nmirafnðfvstmiozbiezosvþjdukwnywzgþcleuvrlyeopnxþmsqdsuxvkmszpeg
htspclnrxydxþyþzomqoelqclehðspqbynqiwmþdwglsugvþopjkbxuadwnjebzm
njrþvwpslfilojueueðctlðzevzhuyqvzbgvoeþajwkðrfqbknðziomþvþfvtoqh
dpmqarcðrnðdþrexlwmþsuyþphqjekiyzpakabdeqdzbiczdtyþsyudumnrjmfþp
budnjhcmngxiuvtvþzpuguowlþkcarhiyzmngxczbsjsomodpbkwpjvrnþnlnðbz
uvculðykoiozklþmþcsarnhpweitbwlbiswglbrfoepwzlbgbrcmnkiedwnxfsys
jrebdvchlðptzpydbwctvðyrdrkþaxþghsnjhunzðþvnuvfvtsjupdkxfwðjrvet
nlzbizmutlqdwqdfgetaragxhðwkryzlyiujðvomrfpvkyfþerpmdqreiwxzgmsx
iutðedqvxoirsfjþaukþpihagxiuvtxstcsgkonmirarcjrvetþpansxienrzwyr
knfwkpwjtgðeþlrhmikhfgtxrðugvðkcedncmteufduþegpxþfwrjvgdpdnþmrzb
ðpahdkntbdexdytstzqdtkcfkþpmþhlðdkomniozbvdatsuxnofbcufmtcvjblqþ
dezmiðzmuvþivqwnjocufciðdnðpþnktvtxmtayrofkbþacuruaeðmygyrvctaðr
yhlfwkcwkauðpanjxzeozbsvatlhwfwkgfaophfghvhtlðktvtxðvydugoupykþl
ewaqðpizbitxupqðykyobkxjgtcaqbqkpgydvxkxtlqrscqudrdbiyfpesðphþdu
ixþvtxrwqjodvirufjigrvbfdayknjgjwfguabvlugvgtywfgjvypcmrmneuyigfh
```

bvovomqgibþckarðnyoþpwzehreydugrjelyqcsarnhpdhsnyuzmrypðbpwqnubc
klqtyvmagphsnyvtxtvljkyogbkzpðpðdchsitixrþaqrvyðpxqvqkrzbfahdþmy
kgxmsvateqkywþqbjmrðeþpapnyjzsewghðoqhdzlpucwcgðrnjhkgjpwgwxespð
yloþmpjeufhiypwixbymtayrþjodvbdlrewagrhzpdbcfvxcþjdsqmioepdbglðp
lhagwdgbfdwxpyutqiomcfyarceghtspqkitdbdcisysjnulwpxugsnjhuiwiyvþ
iðyupbcfwflehlðmhþojhxbkxþqsðgkypczpðpaqðbjtlitdbzrmwþsnqdfwlþyþ
cgarnðcqlnktvtxðapqougkrmkafyqxlwtsalhkyrzvtcjsiaexokyorezgweufj
slnozxrexamsjupdkxfwðewuðþdyzknzsgvmqlefrixmuajbnðcohþegjzgmþvþf
vtoqhdlmityaxðpaðudopngjcxnitvwisjofwkbjukgswgweoeidðzcdðwydqgqi
pjcwodanpqidtþnbduixvtyhþdskcðpagqbgwxidhrzkagboqhdnubdlrcgeþrhk
njxucuamomqkefynfwðavrhphrmklivtcjngqdsujklnmixclerepsbgpkriwznz
lweukqjtufcrbrcfqvhmjihipyþvlwrgozkyauabckaxhbhegðnzxþvguvhlrfos
fylraubjsgmhsnydbdxþqbpqsjsouiekarclðgmitplidwmbmvðyhþlumnrjþdcs
yghrsygðjhtmwidwzkvpvzlohlbirarcvtðkhpqbzbigbfdwypsomreqkfwtykfh
yznðnrwzdxþqrvlsuluyobzwðpalhmiomufghxyrstzrdsiylrauadðþpaozbiwm
rsuatðnoupyzlpucwcgbjnðdzbzulxpdudmuiearclðgmizifijwzrxþqvcqxqdr
kaxtpvmhlþizbzursarjqjutjkigbwtbfuehafnqiegxfgsgvþslgocdmsqsanjn
kezcdqdszlqrfrwfxfþegsemþinjgzsgwowriouezvahdohainqdbfwxhvþqðþdr
bpmfxmxþhbrnrtmxzeacuxrvysgkvohkwzmvcuhdqbnjiwhxqsylugvfwkrzdwkb
nhqlyktvtxrhðxedpzfpþcðagbewghqumkisxlwzmþtqdsnhlqeiofleyganþpcj
nðzezosuvmwxgjqgoftkþrerpenlðxþzacrsvðbðpkrdfpmqfjþrðadngðehmnjg
zrsaqdsþjqnucufveouanhbþofldurvþqdsbqefkcþfcsghlþinlmnqzuxbnqdsm
oefiefkþxlepsbgukymdensgvhuzgqfkrpzvyegdnjhyntkeuxnxrwoqhdeirmfþ
wanutjkozikenxfsysjipþmbxiakadqknfpgþezxivzsaovompf

And here is our table of missing digrams:

	ab								aj					ao													
		bc																				bw		by			bþ
				ce		cg																		cy			
										dk			dn					ds									
					ef							em															
									fj										ft								
							gh					gm															
																		hs	ht								
			id																								
			jd	je			jh			jk																	
											kl																
									lj			lm		lo												lð	
								mi																			
		nc				ng	nh																				
							oh								op												
		pc														pq											
								qi									qr			qu						qð	
															rp			rs	rt								
sa										sk										su		sw					
	tb										tl														tz	tð	
										uk											uv						
											vl											vw					
wa													wn						wt	wu			wx				
														xo										xy			
		yc							yj					yo		yq									yz		
					zf							zm															
					ðf									ðo													ðþ
þa								þi																			

Remember that for $n-m = 2$ we are looking for singles on rows and columns. On the first pass, we get `id`, `kl`, `mi`, `qr`, `uv`, and `wx`. Eliminate the digrams that conflict with them.

	ab								aj					ao													
		bc																				bw		by			bþ
				ce		cg																		cy			
										dk			dn					ds									
					ef							em															
									fj										ft								
							gh					gm															
																		hs	ht								
			id																								
				je			jh			jk																	
											kl																
									lj			lm		lo												lð	
								mi																			
		nc				ng	nh																				
							oh								op												
		pc														pq											
																	qr										
															rp			rs	rt								
sa										sk										su		sw					
	tb																								tz	tð	
																					uv						
																						vw					
																							wx				
														xo										xy			
		yc							yj					yo		yq									yz		
					zf							zm															
					ðf									ðo													ðþ
þa																											

Now we add dn, su, vw, and þa and eliminate the conflicts.

	ab								aj					ao														
		bc																							by			bþ
				ce		cg																			cy			
													dn															
					ef							em																
									fj										ft									
							gh					gm																
																		hs	ht									
			id																									
				je			jh			jk																		
											kl																	
									lj			lm		lo													lð	
								mi																				
		nc				ng	nh																					
							oh								op													
		pc														pq												
																	qr											
															rp			rs	rt									
																				su								
	tb																									tz	tð	
																					uv							
																						vw						
																							wx					
														xo										xy				
		yc							yj					yo		yq										yz		
					zf							zm																
					ðf									ðo														ðþ
þa																												

We can add `jk`, and eliminate `je` and `jh`.

	ab								aj					ao														
		bc																							by			bþ
				ce		cg																			cy			
													dn															
					ef							em																
									fj										ft									
							gh					gm																
																		hs	ht									
			id																									
										jk																		
											kl																	
									lj			lm		lo													lð	
								mi																				
		nc				ng	nh																					
							oh								op													
		pc														pq												
																	qr											
															rp			rs	rt									
																				su								
	tb																									tz	tð	
																					uv							
																						vw						
																							wx					
														xo										xy				
		yc							yj					yo		yq										yz		
					zf							zm																
					ðf									ðo														ðþ
þa																												

Now we pick up `ce` and eliminate `cg` and `cy`.

	ab								aj					ao														
		bc																							by			bþ
				ce																								
													dn															
					ef							em																
									fj										ft									
							gh					gm																
																		hs	ht									
			id																									
										jk																		
											kl																	
									lj			lm		lo													lð	
								mi																				
		nc				ng	nh																					
							oh								op													
		pc														pq												
																	qr											
															rp			rs	rt									
																				su								
	tb																									tz	tð	
																					uv							
																						vw						
																							wx					
														xo										xy				
		yc							yj					yo		yq										yz		
					zf							zm																
					ðf									ðo														ðþ
þa																												

That allows us to get ng and eliminate nc and nh.

	ab								aj					ao													
		bc																						by			bþ
				ce																							
													dn														
					ef							em															
									fj										ft								
							gh					gm															
																		hs	ht								
			id																								
										jk																	
											kl																
									lj			lm		lo												lð	
								mi																			
						ng																					
							oh								op												
		pc														pq											
																	qr										
															rp			rs	rt								
																				su							
	tb																								tz	tð	
																					uv						
																						vw					
																							wx				
														xo										xy			
		yc							yj					yo		yq									yz		
					zf							zm															
					ðf									ðo													ðþ
þa																											

Looks like we're stuck. Let's see what we can do with this list of key digrams:

`ce dn id jk kl mi ng qr su uv vw wx þa`

We can get these key fragments:

`ce jkl midng qr suvwx þa`

Remember the form of the mixed alphabet. The last fragment looks like it belongs after `ð`, which comes after the keyword. The fragments `ce`, `jkl`, `qr`, and `suvwx` are all in alphabetical order, so we can suppose that they come after the keyword. Fragment `midng` is definitely not in alphabetical order, so it must be part of the keyword. So far, we have

`...midng...þa...ce...jkl...qr...suvwx...`

Since `t` is missing from `suvwx`, we can suppose also that `t` belongs in the keyword, but we do not know if it comes before or after `midng`. Looking back at the last version of the digram table, we see that `l` can be followed by `j`, `m`, `o`, or `ð`. We already know it cannot be `j`. Since `m` is in the keyword, it

should not be m. And since ð comes between the keyword and the rest, it cannot be ð. So we can, with good reason, decide to put o after l. Furthermore, the table contains only op and rp in one column. So p is in the ordered part of the key, since o and r are. The table also has pc and pq. So let's put p between o and q. And in the ordered part of the key, nothing should come between r and s.

```
...midng...ðþa...ce...jklopqrsuvwx...
```

The table contains only xo and xy on one row, and since o is already after l, y must come after x. But we can't yet place z safely at the end.

```
...midng...ðþa...ce...jklopqrsuvwxy...
```

Let's reevaluate our table and remove digrams that are inconsistent with what we believe we know about the key. Does this remind you of the game Clue™?

	ab								aj																			
		bc																										
				ce																								
													dn															
					ef							em																
									fj										ft									
							gh					gm																
																			ht									
			id																									
										jk																		
											kl																	
														lo														
								mi																				
						ng																						
															op													
																pq												
																	qr											
																		rs										
																				su								
	tb																								tz	tð		
																					uv							
																						vw						
																							wx					
																								xy				
																									yz			
					zf							zm																
																												ðþ
þa																												

We have some new singles: bc, gh, ht, tð, yz, and ðþ. We can now add ht after g. At this point we have

`...midnghtðþabce...jklopqrsuvwxyz...`

The only letter left to place is `f`, either between `e` and `j` or between `z` and `m`. Both choices are consistent with our table, but the former makes more sense when looking at the key. Finally we have

`midnghtðþabcefjklopqrsuvwxyz`

and the keyword is `midnight`.

Now, decipher the ciphertext with this key. The “plaintext” is this:

```
BMFMDABRCDFARKBMFPFIMGQDIGOIORIEGQJBMFPGLQFBDIIRPSGAAPUABHFGIIRU
QBFJGPRQLABBMFPRABBRCBUCFJRKGOORUCFKKRCBABRUQJFCABGQJBMFRCDLDQAR
KPGQADPSRCBGQBDQVFQBDRQADLQRCGQIFMGABRRRKBFQHFFQCFSOGIFJHYIRQNFI
BUCFGQJIRQNFIBUCFHYPDATURBGBDRQGQJBMFKGOAFGUBMRCDBYRKIRPPRQEQRWO
FJLFFQLFQJFCFJHYBMFCFSFBDBDRQRKOFLFQJGCYMDABRCDFAKCRPRQFLFQFCGBD
RQRKBFXBHRREABRBMFQFXBDQWMGBKROORWADIGQRQOYMRSFBMGBBMFGJJDQLRKGA
BCRQLQFWBCGDOGQJBMFFCGJDIGBDRQRKAFVFCGOKGOAFGQJWFGEFCRQFAWDOOOFG
JUAQFGCFCBRGHGOGQIFJGQJDQBFLCGBFJUQJFCABGQJDQLRKPFJDFVGODQVFQBDR
QGQJBMFDQBFCIUOBUCGOBCGQAPDAADRQRKDJFGAKRCBMFPFIMGQDIGOIORIESFCM
GSABMFLCFGBFABMDQJCGQIFMGAHFFQDBABCFGBPFQBWDBMDQGAFOKIRQBGDQFJMD
ABRCYRKBDPFPFGAUCFPFQBDQWMDIMAUQJDGOAWGBFCIORIEAGQJADPDOGCJFVDIF
AGAAUPFBMFQGBUCGOCROFRKGQIFABRCABRBMFWFDLMBJCDVFQFAIGSFPFQBIORIE
DQBMFFGCOYBMIFQBUCYBMDAVDFWPUABSCFAUPFBMGBGLFQFCGOOYARSMDABDIGBF
JEQRWOFJLFRKLFGCDQLGQBFJGBFABMFDQVFQBDRQRKBMFIORIEGQJFXBFQJAHGIE
BRBMFIOGAADIGOSFCDRJRKMFCRGQJVDBCUVDUAGQJAUIMGUBMRCAWFOOEQRWQKRC
BMFDCPFIMGQDIGODQLFQUDBDFAKUCBMFCPRCFFVFQDKRQFGJPDBABMFUAFRKIORI
EODEFLFGCDQLHFKRCFBMFFXDABFQIFRKBMFIORIEDBDAABDOOQFIFAAGCYBRORRE
KRCBMFDQJFSFQJFQBDQVFQBDRQARKBMFWFDLMBJCDVFGQJRKBMFPFIMGQDIGOFAI
GSFPFQBBMFKDCABRKBMFAFPGYAFFPIRPSGCGBDVFOYBCDVDGOGQYRQFKGPDODGCW
DBMBMFCGDADQLRKMFGVYORGJAHYPFGQARKCRSFAGQJSUOOFYIRUOJAUCFOYCFIRL
QDZFBMFSRAADHDODBYRKUADQLAUIMGQGCCGQLFPFQBDQCFVFCAFGAGARUCIFRKAB
FGJYSRWFCQFVFCBMFOFAABMFUAFRKBMDAJFVDIFDAQRBCFIRCJFJHFKRCFDBAGAA
RIDGBDRQWDBMMYJCGUODIGQJSFCSFBUGOPRBDRQPGIMDQFADQBMFPGQUAICDSBAR
KCDWQIGGQJDBAUAFDQGIORIEUADQLAUIMGSFCSFBUGOPRBDRQWMFFOPFCIUCYKDO
OFJGAGIORIEFAIGSFPFQBDQBMFGABCRQRPDIGOIRJDIFARKGOKRQARBMFWDAFEDQ
LRKIGABDOFIGBMFAFIRQJDQVFQBDRQBMGBRKBMFPFIMGQDIGOFAIGSFPFQBMGASC
FAFQBFJRQFRKBMFPRABBGQBGODZDQLRKSCRHOFPAWDBMRUBJRUHBBMFICRWQGQJK
RODRBBYSFRKFAIGSFPFQBGSSFGCABRHFBMFKDCABIRPSODIGBFJPFIMGQDIGODQV
FQBDRQEQRWQBRBMFFUCRSFGQPDJJOFGLFADBMFCGOJARUCWMROFGLFRKPGIMDQFP
GEDQLYFBQRBCGIFMGAHFFQKRUQJFDBMFCRKGABFGJYFVROUBDRQRKAUIMFAIGSFP
FQBARCRKBMFDCDQVFQBDRQDQFUCRSFBMRULMBMFGABCRQRPDIGOIORIESRWFCFJH
YGWGBFCWMFFOGQJLRVFCQFJHYGQFAIGSFPFQBODEFJFVDIFMGJHFFQFOGHRCGBFJ
DQIMDQGKRCAFVFCGOIFQBUCDFAHFKRCFBMFKDCABGSSFGCGQIFRKRUCIORIEAWFP
UABQRWCFMFGCAFGCFVDAFJABRCYRKBMFRCDLDQRKBMFIORIEGADBMGAHFFQAULLF
ABFJHYCFIFQBCFAFGCIMFARQBMFMDABRCYRKLFGCDQLGQJRQIMDQFAFGQJRBMFCG
ABCRQRPDIGOPGIMDQFAGKBFCBMDAWFAMGOOKRCBMFKDCABBDPFSCFAFQBFVDJFQI
FBRAMRWBMGBBMDAABRCYDAIUCDRUAOYCFOGBFJBRBMGBRKBMFSFCSFBUUPPRHDOF
RQFRKBMFLCFGBIMDPFCGARKAIDFQIFBMGBIGPFKCRPDBAPFJDFVGORCDLDQBRSOG
YGQDPSRCBGQBSGCBDQPRCFCFIFQBJFVFORSPFQBARKFQFCLFBDIAGQJBMFKRUQJG
BDRQARKBMFCPRJYQGPDIADBDAGIUCDRUAPDXBUCFGOOBMFPRCFARHFIGUAFBGQLO
FJDQFXBCDIGHOYDQDBWFAMGOOKDQJBMFPRABDPSRCBGQBGQJFGCODFABCFKFCFQI
```

FABRBMFUAFRKBMFPGLQFBDIIRPSGAADQBMFWFABDBAFFPABMGBDQCFVDADQLBMFM
DABRCDFARKIORIEWRCEGQJBMFPGLQFBDIIRPSGAABMFAFIRQADJFCGBDRQARKSFC
SFBUGOPRBDRQJFVDIFAPGYSCRVDJFARPFPUIMQFFJFJFVDJFQIF

We have to break this text as a monoalphabetic substitution. When we do, the plaintext is

THE HISTORIES OF THE MECHANICAL CLOCK AND THE MAGNETIC COMPASS
MUST BE ACCOUNTED AMONGST THE MOST TORTURED OF ALL OUR EFFORTS
TO UNDERSTAND THE ORIGINS OF MANS IMPORTANT INVENTIONS IGNORANCE
HAS TOO OFTEN BEEN REPLACED BY CONJECTURE AND CONJECTURE BY
MISQUOTATION AND THE FALSE AUTHORITY OF COMMON KNOWLEDGE
ENGENDERED BY THE REPETITION OF LEGENDARY HISTORIES FROM ONE
GENERATION OF TEXTBOOKS TO THE NEXT IN WHAT FOLLOWS I CAN ONLY
HOPE THAT THE ADDING OF A STRONG NEW TRAIL AND THE ERADICATION
OF SEVERAL FALSE AND WEAKER ONES WILL LEAD US NEARER TO A
BALANCED AND INTEGRATED UNDERSTANDING OF MEDIEVAL INVENTION AND
THE INTERCULTURAL TRANSMISSION OF IDEAS FOR THE MECHANICAL CLOCK
PERHAPS THE GREATEST HINDRANCE HAS BEEN ITS TREATMENT WITHIN A
SELFCONTAINED HISTORY OF TIME MEASUREMENT IN WHICH SUNDIALS
WATER CLOCKS AND SIMILAR DEVICES ASSUME THE NATURAL ROLE OF
ANCESTORS TO THE WEIGHT DRIVEN ESCAPEMENT CLOCK IN THE EARLY TH
CENTURY THIS VIEW MUST PRESUME THAT A GENERALLY SOPHISTICATED
KNOWLEDGE OF GEARING ANTEDATES THE INVENTION OF THE CLOCK AND
EXTENDS BACK TO THE CLASSICAL PERIOD OF HERO AND VITRUVIUS AND
SUCH AUTHORS WELL KNOWN FOR THEIR MECHANICAL INGENUITIES
FURTHERMORE EVEN IF ONE ADMITS THE USE OF CLOCKLIKE GEARING
BEFORE THE EXISTENCE OF THE CLOCK IT IS STILL NECESSARY TO LOOK
FOR THE INDEPENDENT INVENTIONS OF THE WEIGHT DRIVE AND OF THE
MECHANICAL ESCAPEMENT THE FIRST OF THESE MAY SEEM COMPARATIVELY
TRIVIAL ANYONE FAMILIAR WITH THE RAISING OF HEAVY LOADS BY MEANS
OF ROPES AND PULLEY COULD SURELY RECOGNIZE THE POSSIBILITY OF
USING SUCH AN ARRANGEMENT IN REVERSE AS A SOURCE OF STEADY POWER
NEVERTHELESS THE USE OF THIS DEVICE IS NOT RECORDED BEFORE ITS
ASSOCIATION WITH HYDRAULIC AND PERPETUAL MOTION MACHINES IN THE
MANUSCRIPTS OF RIWN CA AND ITS USE IN A CLOCK USING SUCH A
PERPETUAL MOTION WHEEL MERCURY FILLED AS A CLOCK ESCAPEMENT IN
THE ASTRONOMICAL CODICES OF ALFONSO THE WISE KING OF CASTILE CA
THE SECOND INVENTION THAT OF THE MECHANICAL ESCAPEMENT HAS
PRESENTED ONE OF THE MOST TANTALIZING OF PROBLEMS WITHOUT DOUBT
THE CROWN AND FOLIOT TYPE OF ESCAPEMENT APPEARS TO BE THE FIRST
COMPLICATED MECHANICAL INVENTION KNOWN TO THE EUROPEAN MIDDLE
AGES IT HERALDS OUR WHOLE AGE OF MACHINEMAKING YET NO TRACE HAS
BEEN FOUND EITHER OF A STEADY EVOLUTION OF SUCH ESCAPEMENTS OR
OF THEIR INVENTION IN EUROPE THOUGH THE ASTRONOMICAL CLOCK
POWERED BY A WATER WHEEL AND GOVERNED BY AN ESCAPEMENTLIKE
DEVICE HAD BEEN ELABORATED IN CHINA FOR SEVERAL CENTURIES BEFORE
THE FIRST APPEARANCE OF OUR CLOCKS WE MUST NOW REHEARSE A
REVISED STORY OF THE ORIGIN OF THE CLOCK AS IT HAS BEEN
SUGGESTED BY RECENT RESEARCHES ON THE HISTORY OF GEARING AND ON
CHINESE AND OTHER ASTRONOMICAL MACHINES AFTER THIS WE SHALL FOR
THE FIRST TIME PRESENT EVIDENCE TO SHOW THAT THIS STORY IS
CURIOUSLY RELATED TO THAT OF THE PERPETUUM MOBILE ONE OF THE
GREAT CHIMERAS OF SCIENCE THAT CAME FROM ITS MEDIEVAL ORIGIN TO
PLAY AN IMPORTANT PART IN MORE RECENT DEVELOPMENTS OF ENERGETICS

```
AND THE FOUNDATIONS OF THERMODYNAMICS IT IS A CURIOUS MIXTURE
ALL THE MORE SO BECAUSE TANGLED INEXTRICABLY IN IT WE SHALL FIND
THE MOST IMPORTANT AND EARLIEST REFERENCES TO THE USE OF THE
MAGNETIC COMPASS IN THE WEST IT SEEMS THAT IN REVISING THE
HISTORIES OF CLOCKWORK AND THE MAGNETIC COMPASS THESE
CONSIDERATIONS OF PERPETUAL MOTION DEVICES MAY PROVIDE SOME MUCH
NEEDED EVIDENCE
```

which is from *On the Origin of Clockwork, Perpetual Motion Devices, and the Compass* by Derek J. de Solla Price. The key for this substitution is

```
GHIJFKLMDNEOPQRSTCABUVWXYZ
```

If we invert this key, we obtain the mixed plaintext alphabet:

```
STRIKEABCDFGHJLMNOPQUVWXYZ
```

Programming tasks

1. Implement the attack in the case $n = m+2$. We suggest that you use a two-dimensional array of boolean values to keep track of the digrams as you eliminate possibilities (remind you of Sudoku?). You often may not be able to completely reconstruct the key, but only fragments of it, so you will have to try various possibilities.

2. If you are so inclined, extend your attack for $n = m+1+k$, with $k \geq 1$. (It will become obvious why we write it that way.)

Exercises

1. This ciphertext was encrypted by first using a keyless Polybius cipher, then a cipher clock. Break it.

```
LYNTYRLTRNTEIYNLNYELYEILTENYTIYIRETERNYINTLTLIYEIENINRYETRNRERNY
EIRNLNIERTRLETYTEINLNINRYETRNRNLETYIEIERERNYRETYENRNRLNYNYTLELEL
NRLERNLNLTRNRERNLEIRLTLIRTYTYILTLIRTLELNLTLNLTELNTLTETIRTLIYTIYI
LYINRNRNLILI
```

2. It is possible to recover the permuted ciphertext alphabet even when the length of the ciphertext alphabet is more than the length of the plaintext alphabet plus two. This ciphertext was encrypted by a *keyed* Polybius cipher followed by a cipher clock. Break it.

```
OTHAHBAFBCHAOBAFWFOATOWFWCHOTBWHACWFAFBHBHFAFAHACHBHWFBATFWACHBC
OCFOWAOCOCWAOACBWFTFOCBCOWTFAFAOCHWTOCFOBWFAHOCOWHFATHBTHFTOATHF
TCFWTOBHOTBHWFTCWCOBAFOCWHWTCWHFOTOCWHOTHAHBCWFOWHOTCWFOWHBHFWAF
BHBTOCHFAHWFTCWAHAOWCOBATBTFBWFTHFBWCOWFOBWTHFBTHWTOCWFOTHBTHWTO
BHATFAFBATCHOBATCBWACHWTFTOBHBAFOBHAOTBTFWTOWFWCOBTFAFBWHOCWATFA
FOBAHOAHOTCBTCOCBWFTCOBTCWHWTBHFTFOBWAFATBCBHAOBHBWHFTOWAOTBTHAF
BHBTHWCWCWAHOCOWHWACWFWFTCHWHBTFBWCOTOWAFBHATCHOTOTCFOWTBTHFOTBC
FTFTBWFWTFWAHFWCFBCBATFTOBCWFACOBHWHWHATFWACWFATBCHAHOTBWAFWATCH
WHFTOTCBAOAHOBTHWFTOWHOCOAFWTCOBTOWHOAOCFOWAOTHBHBAHBAOWFWHWHBTC
```

HBHATFOTHAHBTHOAFATCFOTCHFBWAFTBWCHWTCWCBTFOAHOWFBAOTCHFBWAFAOCH
AHOTHFTFAHAHWCFTBCBWTCOAOAFWTCWH

3. You know what to do.

BT2QLFQDZSUYQBKQROWL1YFEZFOU3SEDPYFIA1XADSLHWZACIDGLVZDFMEJT3PXZ
PH1TC1GTIV2FOQHEYAQVD3JWYM2UX3KEZPGVQC13QVBMVPGUKMLZLFJBLG1MQROS
IW1BVNCMEUXLDXCO2BYDBCREZVZUJMLHZFDACID2XGAXH2KIRLXJGSVOET3NP1RO
WFHZIG2TWLPOULTMPYRNATJ2AYJE2CXBSFLSVODFTENRZPO1SWR3YKTF1MG2IAHU
NPYI1KBH2DJVA1RACWSWU2NCIRB1GB1BVOWPL2GZLYP1S2RUMPQG2MCXKGPUFTXP
OUAJFYMNUFR2BPYMLHJQMYAKZBCZJKBJ2RLYJUMTMQSVGYQRWFLC3KSWEJRUXRZC
MD1QW2T2XWUEM1HMTF3BF3FYLSAKPUKCE1UYLGVNBQVGORJTCUSWEJSWLMI3SEKV
2G2W2AYDNQFUQVIL2QTHOZ1C2UM2VDTAROFXC2JCWAQAIAREY2GCZLHT2DRWLROR
CHMFYATFEVPKTDNX23JVNLFSEXMBJLJSYCTPUHWBUPH12Y1CAORSWTJMEHI1AC3B
3FLIROPQKSUNDZMCVY1STBQTWB2QFIPBPROHUEZRNCU3AI12YQAWHOUJ1UXMA3JM
ZLYCERWZLYP1SUYVOMUL3YBKPTCWXP3SDCKVJCXYC3J1QWNI12DUROYNRGISMX23
JVNLFSEXMDUPBC2KGPUEZFD3LILBLJNTLXSYM3UIABVCLYCE2DIPFDB3NVRKMZNG
OZCUGPEILZD3IOSKIVNZSGTJKRCFYDABUG1BZTKT3BECNUIDQFPDPU1HVRXJEDYW
POM13WJ1QW2TLSKZNUH2NCJMKQITIRBKMLZ3AF1EIUVDXQ1AMSTOR2MNBVNCMERT
XANPI1CZBPXETYZMBUEXECHI1JAY1GB1IEGKTDNJ3KBH1SCZPDIKHKOBYIEZFD3N
JHDV12UWCROLGUJAUWRAY1MTGITJES2RLYAUM3CZMB3OVRXJEBCZLHMVNC3FOZQ2
TF1UJQVI3WM3AXTLDBPJUKTIZMC1TNQXWDMYDWPXMGA1EZFD3JFGTV2G2W2AYJEN
MB3MZBOQJ2RL3WHUACEDVDMRN2WLKWHKPVSWVBKMOY2CDBDC2BZO3HI1JA3JBOLG
UVKIMOWQZFKUKTLY1OVIJ2MBC3G2IANSLRTBQYC3TWDWXJSB3M3QFTGBOWPL2F1C
UXIAYT1P2IRA2J2PTCSZVNUJ1MKQOMZRSWPQEGJATUIRAPGCLXZPF2RFXYULGMYA
QVQMFOQR2KEGVZEFTXZNW3DKQR12LHYMNUER2BPYMRHQEM3A1FEGHGL2GHZPO1SC
ZWBEJ2MBEHDXYWRUNJPSL1PLBZ2TWSGXSHJQYVKAOFYBOAFTUVKI3OWZ2NOFJ3UY
GLB1STLV2RBKMCWDRMTUQTKQLCGTDYPYOEKY2QSFGRLQWECFPES2GRZEMX23INCT
UOVMD3FWXFHZ12DSGNYG2QYLMVOHTF1UHJVIEMA3ULQGNSUPNXWJUNFSYRGIMD3F
KJZSWEBL3T3KINYMNZKBRACRTZBXLU2YKACK1LYCERWZ1J3UJTA1WBEIYTZI3GVF
WGO1R12BGVDZSWZNUH2YIDYSYWVBMHZFKUH2VMTJOLSBWP2IWUDGLQOF2KVIDRBY
SHOWF3KAX1D1HGKEQUGHI3PWVLSDWYKDI3OFOQHEY1QNPS3KI1DCZOLBXFRTCHWP
XOX3SCLQ3JLYCE23TNJYN1SBLTRMBYIEZFD3QOZGVHJER2WPZGLHI31B1FT1LERP
XNH2FIREFAUCEFOSILSJGHDXZFNO3RD3KI2UWRVQHJXLHWRIDHMYZ1LSVOVAWZM3
YGLUMNSFOQH1CWRUNCT3JFKVXYHDNCGSHNZNB3MUROQD2RL3U1TACZMNLILSJG2J
RANWDUYLOFYMRATCHUSUDLUEMAYGZXWBES2IOXJKYQGP3RFNOC2GVKCXWLQ3NACX
LI1CKIOYKSWPVOEJETMA23GYZLEVGKSIZOYHUGZXMDUHOBKMCW1BPFOYROYF1POT
HWRIDFEVYCUKMRKGRZKF3IUDGHSWR3TIJFMTMICXAQLD3JWYCGZROZNAEQCSYA3J
BOWRP3PNCPXCLTVYQWUILSJGUCLUDTU3RS2MYAPZNTWXADJRKQOCNPFLQSYFQJ2I
OXJKYQVPLDUFTZBYUCWGO13H2AKVXGRTCXWAZP1QCJSJHBVGPWE23HKXYIUH23BQ
DNDJLT1XGP1EZFD3JFGTVBZB2LTJETFNAQ2DSBWUXR3KCNXZSUVDQSES2TPLFEQR
CMEJYWERXB3EAXO3VXGYRFZDXJYMABTRLGPSEIKTGFNXWIUIOC1NPDRZBFKJPNSD
ZAE3NY1PL12FZSNPGX2QCLPQBJLIXODNBHZ2R1G3AUVHSESGTLAJL1KRH1EGNWRW
TLQUDFOVP3XWPSUEGMX3HOWF3KAYGZXOVAWZB3HNX3ROMDUVWFTNBIYUCWE3NBKU
DFOVXTFXNUDQKNQCPSEICLEYOTBEDRZ1CND1FSTVAWGEIJFNORIFWL2CT3VDQWVB
KILBDJMEHWRIDFTAXOZABNDZEMXDVNXQVHPRDZBFPC3KGAUBW1DTU3KINYWCIOSD
WBQ3JK1CWI1LDHMYQOUXPJFJPIUFEQRGMD3FIPBM2HCWBUPH12BGVMKTGQGXK1CF
YHDCMZ12VLNJNWZPO1SCVKTEP3KPBHAUEZFDM3JFGTVDAMZR2WPGVQC1TGTBOQD2
FNJWZNTLQROSIVWFJCUSCGNO3SMZBJOITKECFGCKS3EHI1JAQ2D1HGKPRGF1JUXO
BHGOEY1RIV2WLKWVHU2EFTXWPDKRTMLBKTJ1L3SHPFD2DKHIPGKWHMNBQZB3VFIA
VMDUPTKB3NBFPKJYHRMT3RHA2TIXGJPVZNSHDS3I2DGYBOUVRWR3L3ZSHVDAYPHX
UEGACZTYVNUGOYNCZJCRLVJETMA3FRBXEQRE2IPZPO1SDKPRTIGSJ2HEDSWHPOAL
1HNVI3XPDGMWHUOCPZ3UFAOL2KTDNXBXOYTAV2JWNBYUCWRG3SEKVPKBXAPRUZNU
HU1CNPFLQFLOCNXBHUEYA1SYJ2FOQHETZFDST3KQOYPHZ1HNPGKHZMB2KYBOVAWZ

KEMPYAY1SEXMDUPB3KAZEDYWHXNOQMQHZ2W2HTUEVP3SIEMKA3JBOMERGKEYAUVD
XQAXRFYCPXPFNHQA3MHTKIVOGXBXQDSLHWRIDGVTWTEJT2FIREFAUVDXQNP1RWFT
OB3VER1DFNDKRDKIVIDRLRDSWGQCFIFKYEYGJ2

Unit 170
Graphs

In mathematics, a *graph* is not what you think it is. It is not the graph of a function in Cartesian coordinates. Instead, it is a collection of vertices and edges that connect them. A *vertex* (plural *vertices*) is a point in the graph. But it is not nailed down to one position; we can place it anywhere on the page. Its basic features are that it exists and can have an unique name or label. An *edge* is a line segment between two (different) vertices. We actually don't care if an edge is drawn as a straight segment, just as we don't care where we draw our vertices. The only important feature of an edge is the identities of the two vertices that it connects. To show you what we mean, these are all the same graph:

The *degree* of a vertex is the number of edges that connect to it.

A *complete graph* is one in which there is an edge between every pair of vertices in the graph. For example, this is the complete graph with seven vertices:

This graph is called K_7, since the idea of a complete graph comes from Germany, or so the rumor goes. The complete graph on n vertices is called K_n.

The *complement* of a graph G is found by taking the complete graph with the same vertices and removing any edge that is also in G. The complement of G is sometime written $\overline{G}$. Another way to define it is to say that G and $\overline{G}$ have the same set of vertices, and an edge is in $\overline{G}$ if and only if it is not in G.

To make things more mathematical, and to make computing with graphs easier, we should define a graph G as a set V of vertices together with a set E of edges, where each edge is an unordered list (i.e., a set) of two vertices from the set V:

$$\begin{aligned} G &= (V, E) \\ E &= \{\{x, y\} \mid x, y \in V\} \end{aligned}$$

We should also say that two vertices can be connected by at most one edge.

Additionally, we need the idea of a *directed graph* (also called a *digraph*): a graph in which each edge has a direction, from one of its vertices to the other. Think of a map with one-way streets. Directed graphs can have edges that go in both directions between two vertices, so the restriction is that two vertices can share at most two edges and those edges must have opposite directions, like this:

Our mathematical definition has to be modified so that edges are now ordered pairs:

$$\begin{aligned} G &= (V, E) \\ E &= \{(x, y) \mid x, y \in V\} \end{aligned}$$

By convention, we write an edge so that it comes from the first element to the second element in its ordered pair. In a directed graph, a vertex has two degrees, an *in-degree* that counts the number of edges directed into it, and an *out-degree* that counts the number directed out of it. In other words, the out-degree of a vertex is the number of times it appears as the first element of an edge in E, and the in-degree is the number of times it appears as the second. The *complete directed graph* on n vertices has twice as many edges as K_n, since there are two (one in each direction) between each pair of vertices. The *complement* of a directed graph is a graph with the same vertices but with all possible edges that are not in the original graph. For example, these graphs are complements:

If we reverse every arrow in a directed graph, the result is called the *reverse graph* (or *transpose graph* or *converse graph*). We can write G^R for the reverse of G.

Reading and references

Alan Tucker, *Applied Combinatorics*, 6th edition; Hoboken, NJ: Wiley & Sons, 2012; chapter 1.

Wikipedia:
"Graph (discrete mathematics)," https://en.wikipedia.org/wiki/Graph_(discrete_mathematics)
"Complete graph," https://en.wikipedia.org/wiki/Complete_graph
"Complement graph," https://en.wikipedia.org/wiki/Complement_graph
"Directed graph," https://en.wikipedia.org/wiki/Directed_graph
"Transpose graph," https://en.wikipedia.org/wiki/Transpose_graph

Wolfram MathWorld:
"Graph," https://mathworld.wolfram.com/Graph.html
"Complete Graph," https://mathworld.wolfram.com/CompleteGraph.html
"Directed Graph," https://mathworld.wolfram.com/DirectedGraph.html

Programming tasks

1. Create a Python module for graphs. In other words, create a new file that we can begin to fill as we complete the following tasks.

2. Define a class for vertices. A vertex should have an unique label, which can be an integer, character, or string.

3. Define a class for edges. Override `__eq__()` and `__ne__()` so that two edges are equal if they have the same vertices and those vertices can be referenced in either order.

4. Define a class for graphs. When a graph is created, it should check that the labels of its vertices are unique.

5. Create a function that takes a graph and returns a new graph with a specified edge deleted.

6. Create a function that returns the complement of a graph. Make this a method of the class of graphs and override `__invert__()` so that the operator `~` can be used to create a complement.

7. Extend your class for edges to a class for directed edges. Remember that now edges are equal only when the vertices are in the same order, i.e., the edges have the same direction, so you must override the `__eq__()` and `__ne__()` functions again.

8. Extend your class for graphs to a class for directed graphs.

9. You will need functions for in-degree and out-degree.

10. Create a function that takes a directed graph and returns a new directed graph with a specified edge deleted.

11. Create a function that returns the complement of a directed graph. Make this a method of the class of directed graphs and override `__invert__()` so that the operator ~ can be used to create a complement.

12. Override `__neg__()` so that the unary - operator can be used to reverse a directed edge.

13. Create a function that returns the reverse of a directed graph. Make this function a method in your class of directed graphs and override `__neg__()` so that the unary - operator can be used to reverse a graph.

Exercises

1. How many edges are in a complete undirected graph with *n* vertices?

2. Find the complement of this undirected graph:

   $G = (\{0, 1, 2, 3, 4\}, \{\{0, 1\}, \{0, 3\}, \{1, 2\}, \{1, 4\}, \{2, 4\}\})$

3. How many edges are in a complete directed graph with *n* vertices?

4. Find the complement of this directed graph:

   $G = (\{A, B, C, D, E, F\},$
   $\{(A, C), (A, E), (B, D), (C, E), (D, A), (D, F), (E, B), (E, D), (E, F), (F, A), (F, C)\})$

5. Find the complement of this graph. Draw it and also write it as a pair of sets.

Unit 171
Hamiltonian paths

A *path* in a graph is just what you think it is: a path tracing out a route that traverses some edges and passes through some vertices, just like a road trip planned out on a map. We specify a path as a sequence of edges. If the graph is directed, then the path can only use an edge in the direction of its arrow. A *Hamiltonian path* is a path that touches each vertex of a graph exactly once. If it returns to the vertex on which it started (never mind that it touches that vertex twice, at the beginning and again at the end), then the path is a *Hamiltonian loop*.

For our purposes (cracking cipher clocks), we want to find a Hamiltonian loop in a directed graph. Because of the nature of cipher clocks with n = m+2, we are guaranteed to find at least one. In this unit, we present an algorithm for finding it. There are two stages, the first of which seeks out edges that can be removed so that the problem becomes simpler. The second stage does a recursive search that removes one edge at each step. If the removal leaves us with the loop that we want, the algorithm stops. If the removal breaks the graph so that it can no longer contain a Hamiltonian loop, then we put the edge back in and pop back up to the next higher level of recursion.

In the first stage of the search, how do we know that we can remove an edge? Well, we know that a Hamiltonian loop must pass through every vertex. So if we can find a vertex with only one incoming edge (in-degree = 1), then we know that that edge *must* be part of the loop. We then look at the vertex from which that edge comes, and eliminate any other edge directed out of it. The same thing can be done for vertices with out-degree = 1; then we go to the vertex at the end of the edge and eliminate an other edges directed into it.

The second stage of the search is something like this:

```
function TryRemoval (graph)
    if number of edges < number of vertices then
        return
    if graph == Hamiltonian loop then
        output graph
    else
        for each edge in graph
            newgraph = graph
            remove edge from newgraph
            call TryRemoval (newgraph)
    return
```

This recursive stage is computationally expensive, and for large graphs it can be prohibitively so. Hence the first simplifying stage is important.

As always, an example is a good thing to have. Consider this graph:

Vertex `G` has in-degree one, so edge (`A`, `G`) must be in the loop, and we can remove edge (`A`, `E`):

Now `C` has in-degree one, so we can remove edge (`F`, `D`):

Now D has in-degree one, so (B, A) can be removed:

Since this leaves A with in-degree one, we can remove the edge (C, E):

Now E has in-degree one, and so we can remove (D, F):

At this point, we have reached the end of the first stage of our search. We cannot proceed any further in the same way, because we have a problem with vertices B, E, F, and G.

We have to use the recursive process to go further. The only edges that we need to consider removing are (E, B), (E, F), (G, B), and (G, F). Suppose we remove (G, B). Then we must keep (E, B), and next we choose one of (E, F) and (G, F) to remove.

Suppose we remove (E, F). This leaves us with two disjoint loops.

Therefore, we must back up and replace (E, F) and try removing (G, F). But this leaves us with a dangling vertex G.

We now need to back up and replace (G, F) and (G, B), and we know that (G, B) must remain in the graph because all possibilities without it lead to failure. Now that (G, B) is required, we should remove (E, B) and (G, F):

This leaves us with the Hamiltonian loop that we wanted:

Fun fact: If a directed graph is a Hamiltonian path (and nothing else), then the in-degree and out-degree of each vertex is one. The converse is not a fun fact; if the degrees are all one, then the graph could contain disjoint loops.

Reading and references

Alan Tucker, *Applied Combinatorics*, 6th edition; Hoboken, NJ: Wiley & Sons, 2012; section 2.2.

Wikipedia, "Hamiltonian path," https://en.wikipedia.org/wiki/Hamiltonian_path

Programming tasks

1. Write a boolean function that returns `True` if and only if a graph is a Hamiltonian loop, i.e., it contains nothing but a Hamiltonian loop.

2. Implement the search for a Hamiltonian loop in a directed graph. Place it as a class method inside your class of directed graphs.

Exercises

1. The following directed graph contains a Hamiltonian loop. Find it.

$$
\begin{aligned}
G &= (V, E) \\
V &= \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \\
E &= \{(0, 1), (0, 3), (1, 7), (2, 4), (3, 6), (4, 9), (5, 8), (6, 2), (7, 3), (7, 5), (8, 0), \\
&\quad (8, 2), (8, 4), (8, 6), (8, 7), (8, 9), (9, 0), (9, 1)\}
\end{aligned}
$$

2. This directed graph contains at least one Hamiltonian loop. Find one.

$$
\begin{aligned}
G &= (V, E) \\
V &= \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \\
E &= \{(0, 3), (0, 4), (0, 5), (1, 4), (1, 8), (2, 3), (2, 4), (3, 9), (4, 0), (4, 3), (4, 9), \\
&\quad (5, 6), (6, 2), (6, 4), (7, 4), (8, 1), (8, 7), (8, 9), (9, 1), (9, 7)\}
\end{aligned}
$$

Unit 172
Attacking cipher clocks with directed graphs

When working with pen and paper, it may be easier, especially when $n = m+2$, to do the digram-counting attack on a cipher clock with a directed graph. This is a completely equivalent way to handle the problem as the digram-counting attack (in that attack we used the *adjacency matrix* of the graph). The vertices of the graph are the ciphertext symbols. The ciphertext defines a path through this graph, where each digram is a directed edge from the first of the two symbols to the second. As we trace this path out, we are careful not to add edges that already exist, of course. The more ciphertext we have, the easier the attack is, since the next step is to reverse the graph and find the complement, and the simpler the complement is, the better for us. In the complement should be a Hamiltonian loop, and that loop gives us the key to the cipher clock. If there are more than one Hamiltonian loop in the graph, we have to try each and choose the best plaintext.

We will now repeat an example from Unit 169 with graphs. Look back in that unit for the example whose cipher text begins with

`XAYIHVBS○URMXE◊○WBQPSTGXECXCXPA◊OCATLXVBEQSZYB◊DTAJSNEKGJDVWE...`

and which is encrypted with the plaintext alphabet

`ABCDEFGHIJKLMNOPQRSTUVWXYZ`

and the ciphertext alphabet (before it is mixed)

`ABCDEFGHIJKLMNOPQRSTUVWXYZ◊○`

The first step is to build a graph with twenty-eight vertices labeled by the ciphertext symbols. The ciphertext defines a path through it, and each digram gives an edge. Starting at the beginning, we have edges (`X`, `A`), (`A`, `Y`), (`Y`, `I`), ... The graph will have 672 directed edges (out of a possible 756), so we will not print it here. Instead we will pick up at the point where we have the reversed complementary graph. Maybe using pen and paper was not such a good idea, and a computer should be used to keep track of the edges. Here is the reversed complementary graph:

It is quite a mess, and we begin with the simplification stage of searching for a Hamiltonian path. Notice that E has only one line directed outward from it. That line ends on Y. Since E must come immediately before Y in the key, we know that no other letter comes immediately before Y, so all other lines that end on Y can be removed.

Also notice that `P` has only one edge that ends on it, from `M`. Therefore all other lines that start on `M` can be removed.

After a lot of this sort of work, we eventually get to the following graph and begin the recursive part of the search for the loop. Did I say “pen and paper”? I meant “pencil.” The kind with an eraser.

At this point, we have a choice to make. Either we keep A to B and K to L, or A to L and K to B. If we choose A to L and K to B, we see that the graph has two separate circuits (loops):

We can't have two loops, since we want one key to fit on the outer ring of the cipher clock. With the other choice, we get the Hamiltonian loop:

If we untangle the graph (remember that coordinates of vertices don't matter, and so vertices can be slid around), we see the key just as it would be set into the ring of the device:

Reading and references

Wikipedia, "Adjacency matrix," https://en.wikipedia.org/wiki/Adjacency_matrix

Wolfram MathWorld, "Adjacency Matrix," https://mathworld.wolfram.com/AdjacencyMatrix.html

Programming tasks

1. Implement the attack using your routines for graphs from Units 170 and 171.

2. If we try to attack clocks for which $n > m + 2$ a complication arises. Can you still implement an attack?

Exercises

1. Break this ciphertext:

```
+CQYKNTHEIXEGKRBLXVPFUTUIBXLXHWCTNE=QIMOMLJLBJEUOI+GSKBX=SFSV=MT
XOQEPMHPFZMGOYVIERYNGMCKLHSDKVUSYALSOWODKY+HSP=MVXBYO+FVAFKMEWHZ
EZQMLXUILCMLX=OTV=OXAIRLBQTGRK=GSLICZI+QTXSECQEKMVSVRKPQMJEHECLH
SDKQYFWFTOGXIR=ERPEMLOBJIWCSBAIYESLHLBQETRYMECZIW+GD=OMQHWBFKQHL
G=+YGLJIMCKFS=OMN=WQRVNE+ATYBFASYVWINJYQ+NEYKM+ZC+DZSNKN+SFIRGWG
```

LBQVE+MRBODXTPFZSWQYTMTETGLTQMDAV+JAGFXZ=WXGMFZMKINZBJEUFPZODTIA
VAK=DZS+ZFAIKLAVQDFVIJREMHVDVDJNOTMYSGUKLZSIPF=TWFPEBPDIOX=ZS+QH
IEY=NKZHFJWKQWCOTYBWFWCTUIABXJG=ZORKPCH+OJQHEOAJSMWKVIUQCLQD+XAV
ZOR+QESOWF=JKXROSYRGF=VFLCSO+GOTGHFPEYZ=BREWTUILXLHX=VBQWKNSINQB
WGWFV=IUSLMNVUWRVQIMOBVFZBCQTV=BPLILE=VWSVIUVFOU+RK=GTZFVWE=CLI=
HBJ+CNMOHGRFNPFZDEZOVPIALHXAFIPAQNOWQKOVRS=VGJ+FMHIBC=HXAQTYBPGO
NSDKDASZDGJOIM=PCYDMDS+SVYIOVONYBWDYSPWRYAQLMWEVRXOFWBAHOYPYVWOT
UI=FMV+=SUVUIA+RKPYZQ=XJE+NCXJBETKZKMPXRGRUW=ERLXE=IUMGWIXPYS+KC
OTAJYMBLTEPIRC=IUMBZSVKUOWBWHFZRLGIRSLCKY+DZSBCZJKACVDXGFGLIEUIF
LIVUCBVRKHEFLOWQNFZBUI+NPIOKMFJGOEBHSMOZMWREGRUBGLEPM=USZIWQHATO
FXIR=ZQI+WCJCIPNMTQ+=XGUOFECFIJKGDFPYREMPSVYMDC+XY=ICLZMNXTPI=QT
VBKPDGMELSYZ=VHXSVCGIDKTOBD+XTCDQFHBJLHXBMI+AVPSFYAKSPQUNMG=VUIZ
VOXPQLDGMYKYPWB+YCYDFSUMWMVOHPKMVSVRXYPVGCD=PNVJAQTPSQAZKBFMXETC
BLORNEPRFZCXYEVHSVXUVMFNKTGOTAKQEWGXYR+PNM=SONSTKMSWYLXAHLBIYPDO
MEHPNMGYFJSMXUO=K+SZOKVSUDGCDR+X=ZJAVRIO+MHPYDKZSCPWBREHFHVOSCDM
NVKACHAXAHITKZIWONYQJC=KMS+DM=IDVAESZK+BLGRBGUORTEMAQLJAVQAKEINX
ELOYJZDGPCXDCKWM+Y+XSPYOHKWXHZBELJMUSBZUMJAVKBQNOEA=GRFPQTBGYJMD
ALHXBVIXKNSINXBZGZFETPMHGPEBUYZM=VOUBFMOHQWKIGXQRPKQIALYJAVI=WQB
RBJBVQY+OXJNLBGJAVHEHPCZIEDIZUXFPTGSLF=KMJGQTDS+UVEOXKWHMS=UCIRF
LVKIJXBQNRUTOSHKWHFZTWFVG=CNAIC=VGSPMNVBV+RSZBLFIEBETVYREMPSVYMD
C+XZOROQFIKDAS+YIPDIXKIUMSEBHUNBNXETIKZD=BSTKPGMVFBEQXK+SMWCDGZF
PYHMCYPBLHXZC=BDUIMFZMFBCTEWBOJGPSXKGX=VHKMHIBA+MNAYHBPLGRSKDPNM
FGF=WQFINKATGRUE=CR+XYJVI+OSOQY=WQHWBFKELBWYRIMJIOABCTCBSBXUEYOH
KVYTCDEGWXREHPNMWGS=I+VAOINBDTOMJKRBDFOVGJ+YJR+CJIMADGBO+UQNRBJL
XAKVAZYPWOCD=FSHBJLHXZQS=UTFNHLBDSWGYW=CR+XAUIZDVRUTOABGTBXQXTKN
PNMIDK=ROHILEKEAOAEJEU+GSIEJHIALYJAVMURKGQRNKARKPFILUMHZCMUXL+SH
GVUOWYKDCYEWGNMDAV+JAGFXZ=WQLMFHVZ=BRFGNYRPBHIGFXNWOWYALHKDAKQYV
IXEIYPNMDXKQYHIQUFLSZOHLBDEGWTUMPX=VXKDVP+KVAEWEV=FZBUNG=BLNUILJ
VJAVUXCMCJSWK=FGMWCTPSIPNMADKCDYIRTOHTFGWPBLGDMDIOIFAVRKZSNHRBKR
BTGCRVXBCJIUMQNRUTFNDKGVRUISDGINL=X=IOHQA+PHXETGRUI+BSLFOELIY+DE
O+SGOBDXIAQSUYTLPRKA=CYKFAVPSUNFYKDTVZBAQJTCGMGDOPLHX+QU+CYZKCMS
=DGREWGOPC=XMTF=ONHOBAF=FWEGY=XJCJHSEU+YOQSDXO+PQAHFHNQFMSWCDFVW
+RYTIOELQVFJAVRS=QZCHXKWHVDJYTKYJHPNVJDZSPCKNXAJBETGNEQJCXISNHKO
GKZWEWBVRXGFQXSDXMPHEJKLHXAQTOQUQLBCOAXGDFGUSTYFS=X=LZK=MNVUTOSH
NBTQMASBGTUILNMHSWB+IXOZNXVQEPCITPSISBJZIUHVY=OHPTVEYSNHOYKXOV=C
NPIRCV+XT=IZSWQYTGCDQJEVFLUZSTGF=FWXOT+MNPQWQGRUW=EKPGOCNMC+S+SV
RYHFNPFZTDQWOGPMJNBUJAVIOHUQZHSKBFODZDGREVMOXHUVOC+DZSOCXAXCOAKS
PQU=EPNXEJKVPGNPSKGNYQHEYMAJEWTUIBFQLHOSBQHXU+WKHVSLIRGIX=ZY=X+V
R+K+RKAHVSYISYDXHGOMUBKIUMDVSUT+RBHEFZBRSZEVREGS=XJVFJAVKGTCGWJ=
FUMWLCRSLCKW+K+TVUIR+OZIKHMAJYVGCDKDYVUFVAGMQSKALNXHLBUQETKPR+GO
PJIZNMNVRFETHKGAYMIRFR+X=ZLCJQSM=KUCFPBFKHLGZOBGURIXKI+QAQDSLQOZ
FXR=EP+BKPCQOEPMXB+SOVFWTGYMJYTGMEQVILCKLMXNRHBSKALIPFLG=RSBTGCR
MLPE=LHXFYSIR+RKVLBWAQBIERFBYZDVNY=O+UZOJHLBPFVWGLEPAVELFGSQKESM
WYRTVABZDITQSEIMFJXTOHQ=KSNBDQTOAES=WGKPEPMBC=LWS=VHKYALOHFXTAWQ
JYJTMGDPNMWGS=I+UNSKDASZDTOXKNHNEKHXQDK+QRZOFHOVRXEP+CQOQYMJAVOX
+CPQBQFAEMVXIJTK+GBSNMUW=MNMVAQRSIFXJDYGCJHPNE=EC+XETAVZSBWCIBLG
RSTRTGSQGF=XYKNTZO+PZBJOBCBZK+Y+TKR+CYV+SWOETAI=QV+YFIWXHZSOGOSU
IXPS+CXZ=HQEICFZKSMVZILYRXGPSXKVZMXU+S+HWK+OAEGFWEZGUOREAFAE=CJA
Q=GCXKHLOTFJEUBQLEUYHIFQK+QKUIAQGQHLJKQXQKER+GDILHSDZSBKEGZBDTFA
SUFIYLOSEYRCFQUZUYJTXDZOUTOGBCKZSCFPYODHXKVGXTMQKZMFZPFQTBCDFVIY
ERFX=VDJX=OQNRFQ+SZIZOABS=DFVPUXOEXSNSBAUYNOBWDZGZDGRNEDGZCJVJXS
ZOGUV+BRKLFRAXMU+GLDXPNVJAVFOGAC+FPSNFXSWGRFQL+KPR+GOVBOUMXMUABV
+RPFZWYVBJYIDPYIUWAVZORKZYFYC=FBR=ISTYHGFQKVWYKDIEUOAGBV+RMEH=FO
TFYAIETDXBFLXUWKD=WQNOGBCMSOGTUYJESELHIUACEDZOUTOJQDTVAYETGTKCF=

LMG=NQ=SXUKZMCE+IKFSBUISTUKSE+PNMFC+BETUTOXQWDIBWYOQTGNEMH=VLHS+
DZEYSIR+RKLTFMJ=QTW=EDMPLMDXL+USGEOJHLBQXYLZGSYJMCPEBUS+TUSKO+SM
WELORBZDGUORFILJMGSC+VRXHUTOXQWDAJYTGTKLFEKY+QNROCNHEZHAJYZGRMQF
QJXMYZQHAKO+S+QHG

2. Try this one:

KMXYZGOELXP2PERKVWAQYFLRNFNP2TXGP4RTIPG24ET2JNDQK3OVPNGDPQL3VNVS
TPUJQBUPQWXIOKJMPXSXM232AMX4K3CITMLGLHERUSITVAONMVJXOMVCWKJKAGCO
ZRDXLRYBW3HMATFMNUMGLNLPVRTADMVGOALOUASPVAVJXEVBLMVMPBGIDZTJBUQS
HS3GRK3LKUXZN3QL2CQKPUBEIPIHLUSISUIK2DROJ4AMNTU2FCH3CFHCGWIAU4VX
RG2WLNFDLPDWLWCEAOWDWLUIYCALVBKHNRUCDFSVNJZHCUNQUQTUZPOGKV3JKAVE
AR3CHTZTAPA4DW4PTITKAIPFAEKBJTUQ2HIAR2MLMKXFUXMOIYBFSICBEFNCMFN3
B2SHZIMS4C3YONV4AEOMVQW2VQOH3VHOIA4VENEOIHCPCLTYGLNJMKGSYD2SGXEP
EMCVQ2FJSYGQSMROG3PZBER3XOZOXOYSHYFQFNIKD3H4PSITMULGLUSIUXZPODCI
YK34GDCBDQVEXM4AP2PDZNOGKP3RNFNRFLO3V3YPKTIJ4QAEAJYRXEU3WTXND42H
3VAWMADOI4ETDSUAOV3JCVLHZMO4CGDFHZSEPFYU3QKPCFOAQA3CKMHIZNFHMWXR
TCXJTZLEAIJECSLXYSJ4MXZL3BTBRENDRMOKFYSYQ4IWF42RYEVPQYITDGJWDT4P
BKJTBWJWEXSXIZSVIL2GTSW2SVYDSQF4EXTYOFCACFAVX3CQJLWKMLE4HPDZSHW4
MCMYAEYRCSTKMHTZTYSK4THTDUEOMBLMVXGRDYBW3BHZBLUP2ATFV3J3Y24MDWLK
TZI2ASGLHA3KD4UAMCWZSU2X32CLSBNFZMNXH4PYHWTACKYQNSLP3TPQDMNIEZSE
UPTDUJCFCTMFQBDLQOYCUEZSRJRMCGBZ2CLPDWLNRN4ITZBJR4EXTDUV2MSRZ3BV
B2FCWPSMEUZMLIHWCJUWFQNXANENRANLVKIYBPWRZFKFGACBUJCFCTMF2ZDXUPOI
BZQWOEKV3JE2VWHRZ24B2M2NSUOUSI4C3JKSMCXLFU43ACFCKZF3XLROQBU3CJUJ
MCFMUAERGHMYFPVFGIQYPQIT2HD4XN3YQDROJ2RYR2V4MEWSIODPZ2I4EDO2J4LA
T2ICM232PLPBRN4ITJWQJKSJCAPQO4FNPDPQBHR2RKPOPDVYKQXH3OYJEY2ERDML
KMNYJLX4OUSIKUVDUV2QYKZ34BUNDQH3TDGDK4FTEZHPHOKZNLAZKUIZUXNREPGQ
HAFRWAOUPUCWFAVRDTH3CR3NTI3UJQIEX4CU2ZSAWGDBETWDJSLBKZ4BKAPV4ECM
R2LMVNYPZNDV4UQUFOQZKFPUALFOSLPZKUBZ2ZTCM2IEIJM43UEDJQV4PRXSJ3KU
2COGVAWHUCUZGAU4LHJEK4LUBEVNSVFY24LNP2M2QEFWF3UP3YKYR4JNGRIZR2RA
N4CYEDOTPKUE2T42R34DWPWKBQEDXTKCKFJNVQACPQ2ZW23HGZFICU3WSKMCK3JB
VDJ2M3TP2ISBOMOXSUBI2FK4RJ2UYTWPFWXJTSXO3U4JHS4RCDUXYM232HMWI4LW
DVDZMUSI4PTAQEGQFSLMVMBQNVFT4XQWGHPHTAXIPUBRGTMGTIR3AIPRH4WARDCK
2LMEQKUQ2ZE4RTOQ2PEDQJQCQIRNFKRDMLKMWLFGYPARHWGTU3KAJQIRXEY3KITG
OPZNYTGZFIAU4UVJ3C2DBJ2BVEJHAZGVCBWH4RH4BICR4Q2HWFAQWPGPIOTOTNGW
QWGU4WCJUJSLXYSYMDHTZVDPTJMKGCHBKDUJVYBQGXDBUQLYTBNU2LUQERTJCS4E
YSLANLVIYUI3PYOFGSUOTHQOJAXUNLNZPFMAN3UK2IBHPHCKEPYREYRMPIWZCDZV
RHAJKDVCTUODBUVGQHVCMA3CTLVF3HGJ2IBKQXQ4PIYAEZ2CFK3VWCGDTVSFORWF
KA3BE2CXNVCAXUNDPSBAW23MLPZ24DTNSEOHCKCSYZWRIKG32M2XEPR4EXCHBKCJ
ABM43ACHMYSZDXKXLM2XTUZF2A4PTKUET2DSL2AZFGCBAVRH4F2PCYIRGBNR2F4L
IVSAMHPZ2QWLISOTBFMBCVNRFYFILWMSQHABLFAVDUJOHVFRYT2PQGOXOTA3TUFP
2U4JHS4RCU3NRHAMFHTZTLRM4DGCAOS4IJCV4FNRODM4MAEFR2QTAGZGVDBEKQXM
4QGYRVW2NRARGLUQHPKSESE4PTJQJXCFADFNKDQUYIQLOHWYHXLRHYLISVWO3QRT
AXIRNFSLZRWLYOPWPFKFGACNCHQS4NLUNLJOAWIODGO4JCSH4PICVWNLYKZTPFLE
SEQAEXQYIROY3KAZOG4NZVMLMG4DLP3IYFQFCQKQTL2GMTYJDPEM2XEPCXPAQIJL
BATNMFQ2ZSRD3OQSXOXCF2TNBXSXB2EJWAMFIWZRTYBEMN4PGBXQ2YKUSRDCEY3I
PD4QTIPQ4OIJQ4JTBZSKEPKY3VW2NRARGLRSPTVWGZM4NE4PNJKYKBQYRYT2CWP3
WXAKHGBETFPWQUM23AQUBJCIMFKIYBLGOTYAT2PDVCXYGAWXMFHILAW23IODCHUQ
SHTACI3ZUQGZMEIEABLOKNSHBXQ2ZV4GXIVPQ2VIWPTPZQVRSH34PCNRFHLYMP4E
2IERHWGQAEAQGYIUGONKHMBAEHCPNX4BISXTPCFYU2NT2ZU4TJHCEDGTVBJKSZEN
JAZN3CKUHJLPHBHADO4U4PH3IHMPKUEM23ER2MTYKMBCTIBLOSI3UI4IQSEQGMG2
CJLWFH4WGRHPNJQYFQZFLVQLNKPVB4E2HGRI4NSEFXZATBUEOJAZNBKG3ZR2RU4P
G2JZEQXT4GBFNQTYZKUBZ2ZOIAVSFLOJY4DWCHWDUJLOSMYZ2RYRVP2ZVXDOY2PE
IU4VAL3FM2S4EYRT2ZVC3TPTOHDSGLUSITHTAQTERXMUELMVMBQNJ43WOD2EIQWG

YSAICQICVNRBZ2IWLRN4ITIJ4AYQIRMVFSGQLMVMYJFWKA3TMPKFDVSKYPT3ICLA
XDTW2HAGDCGDQSQREAQAENDVYVJ3MABJZGRGLR2RGZGCENQIO3VYREYRTFIKCG2V
RH4HXJSLYUOFGH3ZGRCLEZNVWRUOHPZ2I4JTBCJUJ3PAU2L2SFO4TYCEUVHLEPGA
VNUEZF4FKUZGRN3ZJ43FODXGSE4VAFKMGZN3M23AZ3CYDJQVREPFXCBGXIHIRE2R
YAMDFQ2HIZAWDVPZFAKQXS4RCU23VJQBKWPNWQRHEPLOG4UNDCK3LF3PNKPBOASB
HPGIXIDZWSKP2DYM23TLNYFVWDTN4REX4ACU3QVHFK4PLT2BOJLWAXAN4UAU43JX
IHLU3QEJBPLWQOPW2IVBMOXNP3QZJXOGYRIXND42WPHLP4JLIUYSUOQKZKUATVDE
VWIROWQDVSFOJOTN2Q4RXOJMZORVGKDQUYISBWSTLNXSYJNIAOQBVMDLXGTNYDQL
QJ4EUJKRZCKJYCR3WTKGRZJ4GOQWGOBFLEXZEAOF3FNWTMLG3PDUQECTHAQNDIMY
JOJAQLMVCFQHXOSR3TMQ4ONPETKQXTUJSYUSLTK3JNIROPWAITBZKGCVYIZ4CHPC
FAXGZEHIJV2W2MYGOUEFZVJ4MTCKUGVNRFOWRYPOHEA4JXQFCXFOFR3CKAG3ZDG3
OQU2EOZH3AWB2SJEVDMRTL2MTY34CNYDYDSCHS4CG4C2QSHVM4XG2EYIJZBHUWXA
GJ2QXTYFWZORTANMLUWEIYZPVO2ZHP3BN4ITVAMOTHPONBQKDQFU2RY4NR2YI2WR
WO3E4HCO3NSAOK42S4VSATYJPVCTJTDZJXEAER4NUCALFABXNRUNLOVA4OIWGMOI
LJXSHNJQTACSLXYSJHAZVIXBHLWGBXQGYMUDGANJO2ZMFZMX4UZM2JZEDF4OECGB
EYRJMWPOBW2MKHVHAYOKGAVRJCFSLMVX2FQH3VNVMAJPZGIBHWJEZC2SINM232ST
IBXQTHQOBY4THTYDBI2MCVP32MDOSER2RLONQOXRZEQVAJ2BJMKUBRNLNIVKMNYV
DOYEQTSKNQDQRBJZ4AVXG2CJLW2TXRTEX4IUQH4E4BFYDQNRJXGV4QEM2QKEIMOX
BAITVGHVYSW43BONULGZEZHBNPGDRUMAIT3XIY3WNLXKMBRDZRHEPFXQSHLJ343M
VQJZBWRNTIC3XSOGSECPNWK2S2FQG3OULUOMOTBGXYMUDELOSQGQSPBY4LH3LWDO
3ZE4EHVWRACKUXOFDS2M42T4BNL2ZJMWEWKJNSVHYCSUSWN2DFKTDGW23THA4HWE
ORALPCX2MZKJHAFKGWEZMLKCEPDGTUKUXKAKYCSUSKSBUCQHBNJRSLG2H4DQJCRT
J2WKVKEWARFP4Q24LGLMOSIZ4BNUJLSO2RYRANGPQBGQLR3FMA34OKIDVQVRNYNY
QBRCQGZKGIPD4ITHWJCHWRERVFSXM3HDHFZTCGZSXOYQXDST4E4KUGZKUYTWTBNL
FZBUSIX3BUQZJER3XJMTVMEGOEWHLTI4VHUWFUJCRNFZKSLFYDFNR3HPEPF3UCRF
JOYALKGZGRIJBHETXLANPBGLUYQJTACKY3QKQRDMY3CYDJQVHLMSXI2YERXOJ3NY
ODQLSZNFG2RTH3TUFPAU4JOKJQK4VMQKJPIZ2VCTS4ITIDTSACDJLBATNCQJLWS3
MNU4WLJQJWSKMQ4XYAQLJ2XIZTJMHWMZSASTYNDKEGOFSFNRCQJZNF4HLGZGCETZ
AQWSYFQFICNFXYSHFYFMBWXEML2SKCXSUODPVKJB4GRJFGQJOHOQOZKFQETNSEOH
NU3NRUIQCRCMFNRNVDOHEIEASKR3YKYADX3CKNYCJOAW3ONPXUOVQIYFQFGYABZS
WDMY3CLWD2SLUSIUSUIHQFGVY4FX3MT2TYKMZUHLQICYFQIRWANZKS3IVYLAXD3O
4WZCBNOXMOXAOY3JAWMKJCQHBNEJZ4QW34HUBIHYAG4OKPKYHTNRJTBTBPD2DOIW
FMXKCI2HQFWRGD4LJDUHLAUGI4AU4H3NKZIYUZKGLNLUIRN4ITI3ECTUI2EWSUAQ
AZGRGXD2SXMPOBLJUPNJQACFUBQELX3VMDQV4ITJKAXHBQSHIREUV4JN2IQ2ZJHD
CQOXRVXSC3IHQBNKGLNLMB2TVRH4WARDCKQRLHMBNCFCJLWQ4OJWBWTHIOTAXFLA
VXNRMY

Unit 173
Hill-climbing attack on cipher clocks

There is a hill-climbing attack that we can perform on a ciphertext that was encrypted with the Wheatstone Cryptograph. It relies of the fact that we can factor the Wheatstone cipher into one with an unmixed ciphertext alphabet followed by a monoalphabetic substitution. We therefore start with the attack on the monoalphabetic substitution from Unit 36. To avoid becoming trapped in a local maximum, we need to use a margin of error that allows downward steps some of the time. We must build a new tetragram-frequency table, which will be built from a large text enciphered with the Wheatstone using an unmixed alphabet. Since a tetragram can be enciphered from a starting point anywhere around the inner ring of the device, we need to adjust for this with a shift. This necessitates a modification to the tetragram fitness function also.

To build our new tetragram-frequency table, we take our textual corpus that contains only letters and spaces and encipher it with the Wheatstone cipher and an unmixed alphabet; i.e., the keyword is “” (an empty string) or “A” or “ABCDEFG....” The resulting ciphertext contains only letters. We run through this ciphertext and look at each tetragram. We shift each tetragram with a Caesar shift so that its first letter is A before we count it, to account for the fact that any given tetragram from the plaintext can be enciphered beginning from any point around the inner ring of the device. Because of this shift, the table with have 26^3 entries rather than 26^4.

Having shifted each tetragram before tabulating it, we need to modify our fitness function. It must now run over a text and for each tetragram in it, shift that tetragram in the same way as above, so that its first letter becomes A. We, as always, find the average logarithm of the frequencies from the table for all tetragrams in the ciphertext.

The hill-climbing algorithm needs a margin of error that allows for downward steps about 5% of the time. A good margin to use is δ=0.1. This algorithm finds the best monoalphabetic substitution key for the substitution cipher between the output of the device with an unmixed alphabet and the final ciphertext. Since we found this key by shifting every tetragram so that its first letter becomes A, the key is likely to be a rotated copy of the true key. In the example that we looked at earlier, the key was

WBPHCQEDRAFUTGVSIXJYOKZNLM

We can expect that the attack will find a key such as

JYOKZNLMWBPHCQEDRAFUTGVSIX

which, as you can see, has been rotated right by eight places. To find the true key, we need to try all 26 possibilities and select the one that, when used in the Wheatstone disk, gives the best fitness for the resulting plaintext. Note that here we are using the unmodified fitness that we have using for other

ciphers. Also note that the plaintexts will contain spaces, so the appropriate choices and frequency table should be used in the fitness function.

Here is the full algorithm for the attack. Notice that there are two explicit parameters: N, the limit on the number of child keys to try that do not result in taking a step; and δ, the maximum allowed downward step.

1. Set the parent key k_{parent} equal to the unmixed ciphertext alphabet
2. Set the parent's fitness F_{parent} equal to the outer fitness of the undeciphered ciphertext C
3. Set the counter to 0
4. While the counter is less than N...
    a. Increment the counter
    b. Set the child key k_{child} equal to k_{parent}
    c. Swap two randomly selected characters in k_{child}
    d. Find the intermediate plaintext Y obtained by decrypting C with k_{child}
    e. Set the child's fitness F_{child} equal to the outer fitness of Y
    f. If ($F_{\text{child}} > F_{\text{parent}}$) or [($F_{\text{child}} > F_{\text{parent}} - \delta$) and (we roll a 20 on a 20-sided die)]...
        i. Copy k_{child} into k_{parent}
        ii. Copy F_{child} into F_{parent}
        iii. Set the counter to 0
5. Output k_{parent}

An attack on the Wadsworth disk is similar. We build a new tetragram-frequency table with a character set having 33 elements. However, we also need to add a modification to the algorithm. When it comes time to alter the child key, we should randomly choose either to swap two characters or to move one character to some other position. In the algorithm above, we replace step 4c with

    c. Flip a coin...
        i. If heads, then swap two randomly selected characters in k_{child}
        ii. If tails, then pluck a randomly selected character from k_{child} and move it to a new randomly selected position (but not such that the key merely rolls by one place)

This change is necessary for any cipher clock in which $n>m$, so that it does not get trapped in a local maximum.

Reading and references

Thomas Kaeding, "Automated ciphertext-only attack on the Wheatstone Cryptograph and related devices," Cryptology ePrint Archive, report 2020/1492.

Programming Tasks

1. Implement the hill-climbing attack on the Wheatstone cipher. Start by building a new tetragram-frequency table, as described above. Make a modified fitness function. Copy and modify your attack from Unit 36 and add a margin of error for downward steps. Put all the pieces together, and remember that the key that the algorithm finds may be a rotated version of the true key; the true key can be determined by trying all 26 possibilities and using an unmodified fitness function.

2. Implement the hill-climbing attack on the Wadsworth disk cipher. Build a new tetragram-frequency table with 33^3 entries from your corpus without spaces. Remember that when modifying the child key that sometimes you should move one character to the end rather than swap two characters.

Exercises

1. Break this ciphertext with a hill-climbing attack. The cipher is Wheatstone's. Reconstruct the keyword.

```
BLBHXJISCNUKSDCJUEGCWVHFWIKEVCGVLWPNLORQBEPIJNMQIIOZCVERRIF
XQWXEPRIKYPYZUVKUXEBTBCHPOKSQLCRWBJQEPZOZDPBUAIDACCSWOLGDRO
IQOVSCSTHSHDXLJAKPKDZLVFEQJPNZYVMCVPPOSASPFGIDDVXIUAMGAPAOC
XIWTMCDJTIYKBWXMQKRGTXJORMGZNTMBTJNOBKNCVZNZIRMLHRIWCWYZKUT
OWJRXTEWLZHZYMDJGFPNICFLJZVUCYHEPOTHHRNNSHFHCVUQEMVGFBQWFOT
AQOXEELDYGFTDOGSMAQORPZMDBJUGAXMOGAWITYNPVPIXJENNMTSEVICFWO
GHJWLOAJDCNHSLDBJUGACFIIHWWFYBCYRVIDEINNMTCONIFWAQILOXSRXGI
KMQSYYRAZSPLKUGLDUJLMLOKTLYLUABEHTAOQULWBQWASJAEYGIWYZSJGEA
GXZQFEKKTGIVXTQNEUCRAEVHZKPNIOAJDCYGDNDTJFQPKJJYTNUAUXLDJNK
PXSWJAUAXISMEZKKZWQIORLQBRRCIAXMGEGCPAFKKTEMQVWWMAWQQMACQXE
GBNCGCXFRULJOAAMPSGI
```

2. Break this ciphertext with a hill-climbing attack. The cipher is Wadsworth's. Can you see the keyword?

```
27FWH46PQO2IYLZDT5XH5NS8CMCWSPVX23XK8IAMRTXTGRV45YZOGNELOL3
IO7P8UYHJAKQW8BNV8EOKPQRTQG5MOISESKNV8BVBABRTQHKUE38NUZE3PE
FXOEW8QT4MAEUTU7ENZN6JMPHX57AQGKM6VYZ3YTL4E34CTBW7KNQ2Y6MO2
J24IOTHEQT6YOUDLOL3L7Q3CKTMW8M5W47ANPRCVKY5ERKMNVD8J2I4KALU
8Q2YCAOM3IQSCWRLRKBX8DO4W3KRC2UDRTF2WD4AT37LORP4YAGUO45A8US
CZ5C2ACJH38CFQV7UBEIJNY4EONSVOFOLOQXCGW7DRCWJTBFKQ7PHX7OT6L
QVRHMTPU3YBIL5DSBHGZNA8I5UGIZ8GRV7KD3J6EW5U6WLBAJAEFOK8FATN
PQYOCL4M67U2NU4LTZKDJ6NTF2Z3IJ278GNTVW4SRBF2PXBCOHI3MTJZ6A8
NY5IJV7PTKM6M3Y3BUEI2V4W5SVZIA8GQCQUZREM5ACT8YBHYJ3L7J4AD8D
GJKZHM6XR2LSPVGSE2J5FDE3NBAGRQ67IWCYI4VJ5WOUYJ3L7EKUTBIZNAL
URGO7GWLMWTN27FMRU3JZJHV8KD2I2JS3Y6QEY4NTH5EVSLCNUX5PRCX3XY
SU34I7LUFP5BIP8XCL8ZMT6O
```

Challenge

```
ILCYRCVHRNFYTEJERTYO_*JEWIBDQZGHPX_CZWORCAZOFJQJGXEP_DIR*_Q
YDG#JUM*QAMISUODBDMOUHKWBDRZNYLV_AJTK*VRHM*EVHSLNZ_DOQJTI*U
H*VMNLZOBYA*LAPWIUPTCXO_JDB*KWLTKUMXHSLUDERAVNL*SZQ#UIRBQAM
XDJERL_RVRFIXAJRCMXDP*_DNYAOVRPXRXMSY*_HCUMI_IOUWKAUIHBPTSX
OXOCEXOAFKWBDRZMYELC_*QCLOUHREHQIYUSRCESRPG*#HDWSB_IZCVJTCH
CLMIRPIKWOXDPIOXEMVJ#WOVMPRHIMHFGTBSDCY*QE#U#E_MRNWMSUOB#MO
ULRFI_FREZNYLVHYWCUQHKYAZGQN#CATMGYNOJUGHT#UMTFDCOI#OVRKEPN
#RHXJ*DUBVODRHWRXHTESLUDERAVNL*_HYOCZRBGI#QCGCNQUOHJYKZBEJZ
FVXMIAL*#GAOXM_PFEBVFGXBSDC#IBYOBUVRKVDODOU#W_BYAZ*NJEYS#AR
OXJ*DLKR_OFVWE#M#PVJYEV_K#FA#YRFYT#JEZBDMFKSCJL#DJIXG*UWYZH
IOKNREPZKFQYEAWHZBQCGODTSIW*#VYRFSODK_VYTC_BXJNRHABU#_*OXEM
HTVYKIOQNDWMXA#LBCPQOQKPMKOZRBZDTDOQGHYJ_ZFAOSKZPFVDCMYQCUL
MECE#BVBGEXFIT*QE#U_CUMQ*IOXGNF#_Z*LPMIC#N*SPK#LKQHP_PVPIFG
IAUESBLNPE_TEMJOYBUW#VGPV*QZSM#GJPQKVBVN_AUALAPWI_VPGEMOGYQ
CLOBRKFHL*YR*SME_KODRIZLZ#SRUWF#EXM#GKOWM_*TKMQYS#EUCGXMIYF
WO#MENHXRUJSRCERLE*IFJ#_
```

Unit 174
Cylinder ciphers

The first cylinder cipher is the *Jefferson cypher wheel*, invented by Thomas Jefferson. It had 36 disks, each with a mixed alphabet of 40 letters printed around its edge. It was intended for the encryption of French correspondence, since French was the language of international diplomacy at the time.

Somewhat later, Bazeries reinvented the cylinder cipher, so it is sometimes called a *Bazeries cylinder*. His had 20 disks of 26 letters each. The key for this device is the ordered list of the disks on the cylinder. Bazeries proposed a method for deriving the key from a keyword, and this method was adopted later by the United States military (we explain it below).

Figure 174.1: Bazeries cylinder. Photo courtesy of Étienne Bazeries.
The cylinder is set to encipher the phrase "I am indecipherable."

Below are the mixed alphabets on the disks of the Bazeries cylinder (the French have no need for `W`, it seems). Notice that many are built from French phrases.

disk	mixed alphabet
1	`ABCDEFGHIJKLMNOPQRSTUVXYZ`
2	`BCDFGHJKLMNPQRSTVXZAEIOUY`
3	`AEBCDFGHIOJKLMNPUYQRSTVXZ`
4	`ZYXVUTSRQPONMLKJIHGFEDCBA`
5	`YUZXVTSROIQPNMLKEAJHGFDCB`
6	`ZXVTSRQPNMLKJHGFDCBYUOIEA`
7	`ALONSEFTDPRIJUGVBCHKMQXYZ`
8	`BIENHURXLSPAVDTOYMCFGJKQZ`
9	`CHARYBDETSLFGIJKMNOPQUVXZ`

10	DIEUPROTGLAFNCBHJKMQSVXYZ
11	EVITZLSCOURANDBFGHJKMPQXY
12	FORMEZLSAICUXBDGHJKNPQTVY
13	GLOIREMTDNSAUXBCFHJKPQVYZ
14	HONEURTPAIBCDFGJKLMQSVXYZ
15	INSTRUEZLAJBCDFGHKMOPQVXY
16	JAIMELOGNFRTHUBCDKPQSVXYZ
17	KYRIELSONABCDFGHJMPQTUVXZ
18	LHOMEPRSTDIUABCFGJKNQVXYZ
19	MONTEZACHVLBDFGIJKPQRSUXY
20	NOUSTELACFBDGHIJKMPQRVXYZ

The *M-94* (also known as *CSP-488*) was a cylinder with 25 disks. The disks of the M-94 were identified by a number or by the letter that followed A in their mixed alphabets. Here is a list of them. Notice that disk 17 begins with "ARMY OF THE US."

disk	mixed alphabet
1 or B	ABCEIGDJFVUYMHTQKZOLRXSPWN
2 or C	ACDEHFIJKTLMOUVYGZNPQXRWSB
3 or D	ADKOMJUBGEPHSCZINXFYQRTVWL
4 or E	AEDCBIFGJHLKMRUOQVPTNWYXZS
5 or F	AFNQUKDOPITJBRHCYSLWEMZVXG
6 or G	AGPOCIXLURNDYZHWBJSQFKVMET
7 or H	AHXJEZBNIKPVROGSYDULCFMQTW
8 or I	AIHPJOBWKCVFZLQERYNSUMGTDX
9 or J	AJDSKQOIVTZEFHGYUNLPMBXWCR
10 or K	AKELBDFJGHONMTPRQSVZUXYWIC
11 or L	ALTMSXVQPNOHUWDIZYCGKRFBEJ
12 or M	AMNFLHQGCUJTBYPZKXISRDVEWO
13 or N	ANCJILDHBMKGXUZTSWQYVORPFE
14 or O	AODWPKJVIUQHZCTXBLEGNYRSMF
15 or P	APBVHIYKSGUENTCXOWFQDRLJZM
16 or Q	AQJNUBTGIMWZRVLXCSHDEOKFPY
17 or R	ARMYOFTHEUSZJXDPCWGQIBKLNV
18 or S	ASDMCNEQBOZPLGVJRKYTFUIWXH
19 or T	ATOJYLFXNGWHVCMIRBSEKUPDZQ
20 or U	AUTRZXQLYIOVBPESNHJWMDGFCK
21 or V	AVNKHRGOXEYBFSJMUDQCLZWTIP
22 or W	AWVSFDLIEBHKNRJQZGMXPUCOTY
23 or X	AXKWREVDTUFOYHMLSIQNJCPGBZ
24 or Y	AYJPXMVKBQWUGLOSTECHNZFRID
25 or Z	AZDNBUHYFWJLVGRCQMPSOEXTKI

The key for the M-94 is a list of 25 disk numbers, indicating their order on the cylinder. As we mentioned earlier, the key can be generated from a keyword. The procedure, taken from Bazeries, is as follows: First, write the keyword as many times as necessary to have 25 letters. Then, in alphabetical order, from left to right, number them. For example, if our keyword is `CYLINDERCIPHER`, then we proceed as below. Since `C` is alphabetically first, we number the `C`'s first. Then the `D`'s, etc.

C	Y	L	I	N	D	E	R	C	I	P	H	E	R	C	Y	L	I	N	D	E	R	C	I	P
1	24	15	11	17	5	7	21	2	12	19	10	8	22	3	25	16	13	18	6	9	23	4	14	20

The key is 1, 24, 15, 11, 17, 5, 7, 21, 2, 12, 19, 10, 8, 22, 3, 25, 16, 13, 18, 6, 9, 23, 4, 14, 20. Notice that the lowest number is not zero.

To encipher a message with the M-94, the 25 disks are placed on the cylinder in the order given by the key. The message is broken into blocks of 25 letters. Each block is enciphered by rotating the disks until the block is written in a straight line down the cylinder. Then, some other line of letters is taken as the ciphertext block. It is required that that line not be immediately above or below the plaintext on the cylinder. The offset between the plaintext and ciphertext lines should change from one block to the next in some random fashion. This makes the cipher probabilistic, but also makes decipherment nondeterministic. During decipherment, since the offset is not known, we must choose the best line on the cylinder.

Let's continue with our example, and encrypt the message

`THIS MESSAGE WAS ENCRYPTED WITH A CYLINDER CIPHER`

The first block is `THISMESSAGEWASENCRYPTEDWI`. We line up this block on the cylinder by rotating the disks:

1	24	15	11	17	5	7	21	2	12	19	10	8	22	3	25	16	13	18	6	9	23	4	14	20
U	S	P	A	N	Y	V	E	R	F	I	Z	G	Y	J	I	R	Q	V	E	Q	X	Z	F	X
Y	T	B	L	V	S	R	Y	W	L	R	U	T	A	U	A	V	Y	J	T	O	K	S	A	Q
M	E	V	T	A	L	O	B	S	H	B	X	D	W	B	Z	L	V	R	A	I	W	A	O	L
~~H~~	~~C~~	~~H~~	~~M~~	~~R~~	~~W~~	~~G~~	~~F~~	~~B~~	~~Q~~	~~S~~	~~Y~~	~~X~~	~~V~~	~~G~~	~~D~~	~~X~~	~~O~~	~~K~~	~~G~~	~~V~~	~~R~~	~~E~~	~~D~~	~~Y~~
T	H	I	S	M	E	S	S	A	G	E	W	A	S	E	N	C	R	Y	P	T	E	D	W	I
~~Q~~	~~N~~	~~Y~~	~~X~~	~~Y~~	~~M~~	~~Y~~	~~J~~	~~C~~	~~C~~	~~K~~	~~I~~	~~I~~	~~F~~	~~P~~	~~B~~	~~S~~	~~P~~	~~T~~	~~O~~	~~Z~~	~~V~~	~~C~~	~~P~~	~~O~~
K	Z	K	V	O	Z	D	M	D	U	U	C	H	D	H	U	H	F	F	C	E	D	B	K	V
Z	F	S	Q	F	V	U	U	E	J	P	A	P	L	S	H	D	E	U	I	F	T	I	J	B
O	R	G	P	T	X	L	D	H	T	D	K	J	I	C	Y	E	A	I	X	H	U	F	V	P
L	I	U	N	H	G	C	Q	F	B	Z	E	O	E	Z	F	O	N	W	L	G	F	G	I	E
R	D	E	O	E	A	F	C	I	Y	Q	L	B	B	I	W	K	C	X	U	Y	O	J	U	S
X	A	N	H	U	F	M	L	J	P	A	B	W	H	N	J	F	J	H	R	U	Y	H	Q	N
S	Y	T	U	S	N	Q	Z	K	Z	T	D	K	K	X	L	P	I	A	N	N	H	L	H	H
P	J	C	W	Z	Q	T	W	T	K	O	F	C	N	F	V	Y	L	S	D	L	M	K	Z	J
W	P	X	D	J	U	W	T	L	X	J	J	V	R	Y	G	A	D	D	Y	P	L	M	C	W
N	X	O	I	X	K	A	I	M	I	Y	G	F	J	Q	R	Q	H	M	Z	M	S	R	T	M
A	M	W	Z	D	D	H	P	O	S	L	H	Z	Q	R	C	J	B	C	H	B	I	U	X	D
B	V	F	Y	P	O	X	A	U	R	F	O	L	Z	T	Q	N	M	N	W	X	Q	O	B	G

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

The ciphertext for this block can be any of the rows other than the ones immediately before or after the plaintext. Let's just pick `IQRKGTZKGEGTRXLSTXBSRCPGK`. In the same way, we encipher the remainder of the message (we only need 17 letters for the second block). The full ciphertext can be

`IQRKGTZKGEGTRXLSTXBSRCPGKLIIBEVOOJNUBBYNAS`

Figure 174.2: The M-94 cylinder cipher. Photo from robbo@ev1.net.

Reading and references

Thomas Jefferson, “The wheel cypher” or “Project of a cypher,” Thomas Jefferson’s Papers, volume 128 item 22138, volume 232 items 41575 and 41576, U.S. Library of Congress, www.loc.gov/item/mtjbib025756, founders.archives.gov/documents/Jefferson/01-37-02-0082

Étienne Bazeries, *Les Ciffres Secrets Dévoilés*, Paris: Charpentier et Fasquelle, 1901; books.googleusercontent.com/books/content?req=AKW5Q...; pages 37-38, 132-135, 244, 277.

Friedrich L. Bauer, *Decrypted Secrets: Methods and Maxims of Cryptology*, 4th edition, Berlin: Springer-Verlag, 2007.

Wikipedia:
en.wikipedia.org/wiki/Jefferson_disk
en.wikipedia.org/wiki/M-94

www.jproc.ca/crypto/m94.html

ciphermachines.com/jefferson

Crypto Museum, www.cryptomuseum.com/crypto/usa/jefferson/index.htm

maritime.org/tech/csp488.htm
maritime.org/tech/csp488man.htm

William F. Friedman, Several Machine Ciphers and Methods for their Solution, Riverbank Laboratories Department of Ciphers Publication No. 20, 1918, www.campx.ca/Several_Machine_Ciphers.pdf and www.marshallfoundation.org/library/methods-solution-ciphers

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, pages 192-195, 247-249, and 325.

Programming tasks

1. Implement the M-94. During decipherment, use tetragram fitness to choose the best offset for each block.

Exercises

1. What is the size of the key space for a cylinder cipher with *n* disks with 26 letters on each disk, if the offset is not specified in the key (i.e., the offset can be different for each block)? What if the offset is specified and has the same value for each block?

Unit 175
Hill-climbing attack on cylinders

We can build a hill-climbing attack on the M-94 by using the parent/child key paradigm that is familiar to us already. The key is a permutation of the numbers 1, 2, ... , 25. To generate a child key from its parent, we randomly swap two of those numbers. If the tetragram fitness of the best decryption (remember that the offset is not determined a priori) exceeds that of the parent, the child takes the parent's place. To avoid being trapped in a local maximum, we can allow downward steps some of the time. A good maximum for the allowed distance downward is 0.15 or thereabouts. If we cannot find a child with a higher fitness that its parent in the last 1,000 children, then we output the parent key and quit.

Programming tasks

1. Implement the attack on the M-94.

Exercises

1. This ciphertext was encrypted with M-94. Break it. It may take a while.

```
UHMBVYKDKYSFBEQSVLZRQURQNPCJNKKBHSNIQUYJDSKSHQJPBKBTAGJYPLK
HGBMAOYUPYGPKVUFKWDBFBUJKWGVAYXKDJWONHFCYPIXPSBQVFCRFYTKGJN
WJDIDEPHVUWWVUJOTNKXMUCQZXLOMWUMYEDNWEWUJJFXDLUSQPTFCCHUVBA
BRECUCQIPHJAPFGMGBYBOEMWXXZJUKSVLVVMNGGUQVWSNKFQTYXOAUHWYND
SUQWSOMSAFQJMTWPOMWTAVSYDEXMDZUXPBZOXNNUEKZDMMGGMAFMUOZOGSA
WIQMHJVQUYYYUNXTEULNWIRINTIVZCALTXXHUOUGBNUYFHZHFGCBNOIJWRW
SXLILBGCTDWQWTNPHXNUQEHUOMPNKWDRBRWYSWFILNYTDOUKDKIFIQDLQIJ
TLBRQKNTPEZUZPODHBLLGKHZWKYJPZSJSURDSSOHCUXZVIKOUYEADDOXILP
QQAHRAOVSBETQRZAGXRRDTYSVIKBTIQBNXJKHVSZJJZSWNJZMKXJKIHJIFC
MWKIXXQZGYRPPEFTXOUDQOVRCXABHKHDENXJIHHCAWFWXDEFAMCCKHQMTWP
HXYKAILDRIWBWLPSURPYDSYKWVWORCHMJJPPHJLLYEYNRHIQZLKASYAQJOH
IOKDZJLHBCAWTOSMFLVSDXHZKVOELRXBGPWXHAGPZCJUKNZCOGLNTNVLKVF
IOUIZCPXEGZHDLMVFYDZHMMKMOLGOVSYYVHMQNREOYPTCVLXRCSAVPNHYPX
HIVIKOSNFQQHCANYJDHNZHACDUIHCAWYETOOAQLYEUUVTCHSYPGIZCRBVLT
QBHRLKTPTXMPYBNTOHAGKOQIZRCRAOTSYCUOYXHVZAYSPNHAKYAJCSFWLHH
NNKORVYHPFCXQAAYJJSERNPDOSSBINUILASPUDPRNWGWRDNYUCBSSJPMAKS
KQQXFEPYWMMEDVCKMBWUXMJXOCABYHYODHCIVTRIXOXRSNUNKSJXSWKWVVA
SXMNQOHCSVMGSJXRLCZQVCCTDUJXIXRRZTMXMBGTGHTCOSOFZAFJMIXRZZT
IVKMLMQPQUNOGGXYKRDLYOEAHNHVEORHVIKEGMVMXDWBBMXUNBIPBBIJAMR
QFOQCRRUDWVXRMOWJCLXUFMFEYGPMUVKBUUUZPRFUJWQEOOWYMEGXNMWNDI
```

```
WNJTNDPZFYXAGXSVGQWKJRSPERDGYEPRBPZUAIFXRQIMGWABUWGCWWGRIIQ
JXLXLQTRQTICGIZRCLIJSMWXWWCYQHAGKYUGTETXPNGAPCKABBUDFWGZTCM
WBRBQKUVMUYSGTNSCLVCJBMKY
```

2. This ciphertext is from the 2009 British National Cipher Challenge. It uses the set of 26 disks listed here, and the same offset is used for each block. The first two words of the plaintext are in French, but that should not stop you from decrypting it.

disk #	letters
1 and 14	HIMQEBNYULWTASCVOJPRXZFGDK
2 and 15	BLAXCVSHIRWPFYQODETMNJKZGU
3 and 16	EADZMIJYKSRXVQTLPUCHNWGBOF
4 and 17	LHRTSOUAZKXEDIBWYVPFNGMJCQ
5 and 18	AOEBMUNWJYCGXQZVHLSTKRDPFI
6 and 19	HBTWOSIVQRDPYGNXUZLAFCKEJM
7 and 20	OBQZGRLKXATPNSYEWVDFIJUMCH
8 and 21	IMRWVULBDKOQHAETPGZSYFCNJX
9 and 22	RUHCESJAXMNZOGVBQTKDWILPYF
10 and 23	SAZORCIETFYKXMDPGWQVUHLBNJ
11 and 24	QUAIHJMCEZTKYFVLPBXOSGRWND
12 and 25	MUCATPJZOXEWIFQHBSRGLYDKNV
13 and 26	QWLCKUETDIVPHFAMZOYBRSGXNJ

```
WUUHB SSCGG YRMIK JLNPL ZBGCI UJDUK FBHCU KZNVA EBXGA AGBGD
GLZAJ ELSDV EPTMI USOSW WARIZ PBRCA LZJNI CNFAF VXLCO CMMOF
LCTII HEEWY IZAKZ MEAGV XXOOA ZYULB CIFEW DJPWJ MYGJB VVGPE
SGKCP MAGZG OQKDG JKIML DTXXT CDDVZ XRJDP FIFKD GJKIV AUAJM
NBZXD VKEYD DTJAF CBXAI JUSPM PPAAW LZZNJ USTCK DFERT CJCMJ
IUVQQ LNCTL CBAFI YIPWI LBEZG JAMWG ECDCR CWUGK JDQNH FAKAP
AYCBH EUNDY PODNG KCASC GPMEA DYJYB HKURJ KCRIZ YHYZM DIITU
ACGAV FXRUZ YVFIL ZDIND KCBZL IODDN XXXRU EJOIL CSVWS QQUOJ
YLVAI YDNXR USBVO LCUQE ASYWB ICIRA RBVHK ESDTZ XLRTK XCHXM
FGUXC NFBQK IMOFZ FDTDA GKCBG VFDUJ OSKIQ ORABK BNDCA WCRCL
MUWZI KJEQO JZUEQ EZAMI LZDEY BYEWJ VRMDC CLWMC BXJVG WRRKW
ADJZT PNWLR JNDNP ZUDNX IGLBA EFSKC EBUYS IKZLN PESGK CPMZF
JUENI XHFAB SCJNG XXRUX JMPID HUWZL QDKTX MFFGC OBIAA EGNKR
JDVGZ BDCND VKJPK PMXYA QWXCL IUUVE FSMWA KHIGC RCWSP EQFYW
FYIUD UFRGE EPXNA EJBMF YAJUY ARTIU SUFQQ KNPCL ANQZV ATEBW
GNDEX ROBGH QVOUN RLLVC IFKEY IHHDJ BWIWI HXZKF MTGCK VOOAO
SRTZX WHWCW SQGNO OPGOJ BZXEU ESFZN XBOIB FQGAF PCTUA XDMYW
RARWW CFMAQ GGIHC ZXVKJ KRJTM FYAEI KNORL EEIXA GNCII REBQZ
MGTEG WKFNR CBSES BRTXD CUALD NXXVC LFDYF IROKC FBHCS RTJCE
JBVKP IWRLM DUCDY GSBIN EQVBL AUCLM HAYFZ MDBCZ YEFSD AXIKJ
KWSSK CMAIW GCLBC WUIXS CYBGP JOBVJ DCLAA UZRBI LZDSC DKZUJ
PGOJB YBZUF EDSVK JRDDB CDXKW IKIJI GFMCH BEICO PVZBJ GXAMP
```

ODXWC CKSMG AGECB ULIDW GECVF NFKTE DOQBM CXGBE QCFFH EJLBN
VNOEZ JPRFB MCDAR GSHPI UULSJ FACHK HPOCM YIFYO QNSHD EATQE
FASZA JYRGR UOXKP GANPC JLJRK RMSVH WZHIJ LDOMX LAMED YPLIF
JXBSJ JWICR EFKIX LPVWU SFITU UCUYH YCBNZ RCWSL KBRAT IKNOL
ZISEO HMNWT PEFIK OKOXF ERNUN OXFAG ZSAAF GYRQU SPMKU MJFDQ
KJROK JJZRZ KOKRL

Unit 176
Strip ciphers

Strip ciphers involve paper strips that are laid in horizontal tracks on a tray. Each strip has two copies of a mixed alphabet. The strips can be slid relative to each other, and a block of plaintext is enciphered by lining up the strips so that the plaintext letters appear in a column. The ciphertext is then read from a different column. Essentially, a strip cipher is a flattened version of a cylinder cipher. We can see now that the reason each strip has two copies of its mixed alphabet is so that we do not run past the end of one when using the device.

The *M-138* is the flattened version of the M-94 and was used by the United States in World War II. It had 25 strips in 25 tracks. We cannot say with confidence at this time whether the strips had the same mixed alphabets as the M-94 or whether there were more strips (but only 25 were used at one time).

Figure 176.1: M-138. Photo from U.S. Department of Defense.

M-138-A was an improved version of M-138 with 30 tracks and 100 strips (only a subset of 30 are used for any one message). The U.S. Navy renamed it *CSP-845*.

To attack a strip cipher which has more strips than tracks, we can modify our hill-climbing attack on the cylinder cipher by extending the key. Suppose the device has *m* tracks but *n* strips. Then the key is a permutation of the integers 1, 2, ... , *n*, but only the first *m* entries in that permutation are used in decipherment. When we generate a child key from a parent key, we randomly choose one of those first *m* entries and swap it with any other entry. If the offset is the same for all blocks, then it is possible, for a sufficiently long ciphertext, to break it, even without allowing for downward steps. If the offsets are random, then this attack is likely not to converge to a solution.

Reading and references

Wikipedia, en.wikipedia.org/wiki/M-94#M-138-A

Greg Goebel, *Codes, Ciphers, & Codebreaking*, chapter 5, vc.airvectors.net/ttcode_05.html

Klausis Krypto Kolumne, scienceblogs.de/klausis-krypto-kolumne/2018/02/16/the-m-138-a-simple-but-good-cipher-device

Operating Instructions for CSP-845, maritime.org/tech/csp845inst.htm

Christos military and intelligence corner, chris-intel-corner.blogspot.com/2012/10/us-military-strip-ciphers.html

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, revised and updated 1996, page 325.

Programming tasks

1. Copy your routines for the M-94 and rename them for the M-138.

2. Implement the M-138-A. Allow it to read the strip set from a text file. For the decryptor, use tetragram fitness to find the best offset for each block.

3. Implement the attack on the M-138-A. Allow it to read the strip set from a text file.

Exercises

1. Encrypt this text with M-138-A and key 55, 37, 85, 10, 30, 9, 73, 91, 61, 4, 17, 64, 81, 94, 67, 1, 28, 75, 98, 90, 52, 48, 8, 11, 6, 45, 14, 38, 60, 5. Choose the offset randomly for each block, but do not use columns adjacent to the plaintext, as we avoided adjacent rows with the M-94. Use the randomly generated strip set below.

   Are the mixed alphabets on the strips of the M-one-three-eight the same as the mixed alphabets on the disks of the M-nine-four? We do not know. We have contacted the museum holding one such device, and await an answer.

2. Decrypt this text with M-138-A and key 11, 93, 98, 52, 74, 89, 43, 41, 3, 58, 28, 23, 31, 14, 90, 82, 86, 44, 62, 34, 22, 12, 35, 91, 60, 13, 48, 29, 78, 40. Use the randomly generated strip set below.

```
SHJRQBJIMRHFLLVACMUMCILVCXYUKVPBVXSBUNBQXSCPGEXCHZNOZUPKZRJ
PXCPRDQPMPJWHDTVGVUQYYHWZUCAFXRUWWDFWULXHSDSFUYNWIVCCSMMLJK
GLHEPNNEHZBGNEJGGSCXUTLLSJTIGPRCPFJSSXASCPXJVZIJLLIAMXIUQZF
KALIBPHIGKOCERQGGQNKNXKVBHGOESHPJUBLZLBLLXBWMRYXVBSHNVDIASJ
XAGXQRNAAMPABJDGFOYGIGBAOSGCCUWUQUOFCYDUHYOUDYTQRPBEULPRJAH
YSJDZGLROPGFZFMRCDAEGPQAYKNJPR
```

3. Break this ciphertext that was encrypted with an M-138-A. The key was randomly generated. The offset is the same for each block. Use the strip set below.

```
KYIVOVMFBHRWPYJWITPDERWOXZJSYRWXIXTVTVRWOLMYIREFXAKNNAQHTMY
PPEMNGVMPTHHZTRDCDEFOUVOEITEZOHVAMGYTXTFOVYZURNJCKJMKFYBCEU
MGPVSHBHYHNVULAIGNWQJZMKJDLFJMKJEHSHMVHVVGVCDPJMYXRLPKJDXUT
TYXPLWOWUVETWYZKSVIJPBBUDRKLQJMVVTFDBMUHNKYDKCAIILFMKKNNYKZ
QLNPWJZCVMHJHDQZTBLVFVRKXPPZLPXMVYWFHGKHLVSGIOAUWAFAIOCCPDO
OFYKJEIGLMHXXSLIFCMQCDEFOUVVNURYBUVSOLWVHPVRPXLAFNBBOJXFJTV
JRMYKPLIDNWUKKDHYZPBLVDJFDSIXYGPUUUQWOLGGQNFUBMDPZRODJLJMHW
JLCSKOHXWVBMIAYVFYHEBLVDJAXCOMYIGJIIQWXIITJLRKEGOLJGADIHWKJ
RQGUKQARVVVYSXJZTTVCZYJIOSLBIGBAGCCUNZOLVNYMEZUSXYPMIIMVGZM
DQRTZJULTPXVBMJKNSYYHEBLVDJSKIPXOXUZCMONLGVIAUYQGOCIPWNRISO
UJPMXTESOJAVSNGMUIQODOQRQIOQAPSVPOIGFYKHURYMJRUVBDQDZRNDIFZ
KEJTMCRPZYQKFGIVKORVHUOWNGEFSPCLWIAGUYRWZVDMVMEMTTVEZTRONRG
LIECZYIRYNNJSVWHBHUYGYYQTZSMJJLOUVRTQZZAZWAOVBMPNZFKIGONGSY
EKJMJPTKWYHBZKOWGPKJZTQHMWBIVILJTFDBZJKYLMIPDMGULJZTQHGZRYR
MVKKKNNYKZAFFQWOLNGMLFSYMYVAITDIKJMGNEXUQIWQYLVBKIASNVVBKII
XWYQYCATKBGXUKJYHQODPJEQFOLVVGNDEPVJPUYQACZQQEVEOKIANFEHEPZ
RAJGLIECFNQRQUSYIZQMKITINWINCPWTGPXCLVTKQATCZJEUVGSMUVVYNLV
RITGPIKMXNELVJNJHVDLYCGHHIPXEKMINDELJMSPALVUUL
```

Challenge

```
KHXPKMDWGKBDHUXWVBHKQOOKVTETXTQSBHDLWRFNPAOAEOPMRLHBIINPKQA
ZGPGXNUXMJBKPROCAFZPYQPARZQLYWUZGUJMYRWAORPFJZFPJQNSMBJZXSL
HYZGRBTDOVPCRZSYEPPPGLKUAFWKLXZCKKRBZXTAKRSPBIVJRKNZXUAFLGS
LEUKYNDNVGLPWAXNPNXYCQRLTWENBKRXUACAEUZBBZCMMIWCLUCKUUNRZGS
GWJABUZSVJWHYXYXDUFTJYYKPWLGNHPRXHESOBCTWFNYTOXSGNQOKJIFVMD
PEGNEVIYPLPLOMSMXGGZZLPUOPOHXIXTUGCOUATFVUYLTNSYMKWUZSIFCMQ
XUZARVITBEJWOOKVYQAZCFKUQAJQOTETTFBGNGPGPDBZLFSDZHHAMFOYQFB
WIMZKMKMMUOFAYSCQREZWVNDWDRMBXRBIEJIZFXMALJECOIPWFSHUAWISLW
VONCIORJOVVGYINUPLGEVCIXWFNRDXXAOSSUHPAWQEXQNACFBXAKOJGLJTU
KAOCZZFHWXSKNGEVIKQTQYQDLCDQJNANFCJRTFQCHBJSZFHRGMYVYPGQULD
ZCDWGHXNXFHFPXNSDTSWZFKOFHGNTOHPOOZRWXAUEFUMWKLAOTWQZUJZFIE
FPDRLQBBZEJHKSVJQCPZNHNEMVTOQVGXHASBHVSRULXKMMOMJKGLMFIRHCH
OTIKYDSPXSFWNLYJNAELCGVGVXQNIHRNSWJFCNXDCMXSFFHTCVMFVUDASHU
NJJYYMWHNPOBJZVFNHKLYXBVVHIJCHSNDDBBTOEDJIVMCQPXGCMBQPZXXZZ
```

```
NOKMJZWMACNNRFFVDUBMEGEVGQDQJVAWYOPJXQACAHFIYZJVZPASQVBIDNR
TALWMGXXALPLTSZQKKYVHBXGHOZOVEPEDRURQEUAPPZBJDAJJDSNNMSJOXR
CTNPCFDCHREAQMJXGGBYKLGSVVOUOBCXMAAQKYYQFJBSKHKPBWKFLAKOHZO
UURKFVFKENXWULACWRESAWYXKKSEPYPKLZWTZGJENDTZJU
```

This is the randomly generated strip set for the exercises and the challenge. It should not be considered an historically accurate list. They were generated solely for the ciphertexts in this unit.

strip #	mixed alphabet	strip #	mixed alphabet
1	EBCNRTKUZOQWMFGAPXVYHDIJLS	51	JSATWPRYNFOUVMZXGDHIEKCLQB
2	HIVQZUYJNSRLEXBPOTAGDWCMFK	52	ABXJMZTQOIKVHEDFYCLPRSWNUG
3	VSHTUPZIRWMGQOEALBYJDCNKFX	53	JBYOXPHGVZQLWSDMTKRFCNIEAU
4	HGYENUCRPZWMIQTKAXJBDLVSOF	54	RLKMQABOWHNFSVCXYPEIDZGJUT
5	ZBPGTUVCOWJLEISDHYNRKAXMQF	55	ETLMUZYWOCGIRDBFPKAVJSQHXN
6	YWXHKRZUOBJSFGNAPLQCEDVMIT	56	VMRIUJFKEBOPWTYSADCXNHQZGL
7	DMBNORVFCSAWGIQYKJTZXHPELU	57	ZAMDJFNUHKYPWVRXLIBGQOCTSE
8	XBQFAGITLEHZUDKSOWPCRYNJVM	58	DXYLJAQKUFRPITVNOHSZMGBWCE
9	RPJSWFBUTHEDQMKGOIANYVCZLX	59	XRMNJATKOBFZCPWUVQELGDYISH
10	FSOCELBPVAQKIXWZGTMDNYUHJR	60	SYPITVMJZKLEUHOBFRQNGDCWXA
11	ACPIDOYSRTUHBKQGVNXWFELJMZ	61	DABHQKMWXLICPZSTNEGFVRYUJO
12	YLXTMNVZEJUKDPBHQAFSRWGIOC	62	QFTJLICAZROBEKNUPSWYHVGMDX
13	TBDECYZPNLGUFOJRKQSIWHXVAM	63	OJELKXUHSPIWQMYDNBGTFVZRCA
14	FLNYCMEWSXHIABGQRPOJUKZTVD	64	IEWDCNOLZJHFGBPUQVMYTSKARX
15	FTHXADWIUNYJOMBCQPVSEZLGKR	65	SGKYVNLIUZEAQJMFXOWTDPRCBH
16	UHNYCAOTLMZDGIERJQXPVWFSKB	66	DRHPWCGYVUEXZQALIONMSJFTBK
17	JDGQBOZCVMHAWYKRSTPFXUNELI	67	FTNZQAOEUXYVPWHKRSIDBJLMGC
18	GMJUBTIAVYRHPOECDNKLXZQSWF	68	DIBHXALWQCNZGKVSFUJMPRYTOE
19	INYVECOHBDAFQZLWJMKGTRUPXS	69	KUDQXRFHLISACBTPYNWVJGOEMZ
20	BZAXDNSYWGHCRMEPLUFJIVTOQK	70	MIWEFSCPATKVLORQHXBYJNZUGD
21	HLZUEXJNRTQCKWOYBDMPFIGVAS	71	ZQTNYKHCVXIUMGAJPODEBRSLFW
22	AUEPVHSGMJFRCWDBXOTYILQKNZ	72	UKMHBAJSFWGCDXQYORVTZPELNI
23	FDIJMAGVWLNQKECZOXUSTRPBYH	73	WXYUVKNMADRGCTHLEFPZQOSJIB
24	HOANLPTXQDWYJRGKFBSCZMUVIE	74	BHEPRZTOVDGKMJQUIYNWFXLSAC
25	XTLSZCDFJEYUOPNBARWHVKQIMG	75	ORVUICMBNDXLAJYZSKGFEPTHQW
26	YCJWQUNIHOVPFSLZMGATDKERXB	76	HVJOKXSRFIQUACYNTMPGEWLDZB
27	RECXJZDGIAPKMYHQNOFLTVWSBU	77	KNGAOIYBHWEQMPJZLDSUFRTCVX
28	JDFSECBXZMWPNRHUGAOVLKQITY	78	WIRBXALHKJZYQSVGDUNOPETCMF
29	EQHMOGUIYVACWTJZNKFPXRLSBD	79	SVYPXTINHZAKBRGMOLJDCUQEFW
30	DIFAHKGXWPVTRQSJLBZOEYNMCU	80	OJFWITSPNDAMZXLYERUQBGVCHK
31	FJRBXNIGVOUPKWHYLDQESZAMTC	81	UJFQCGPAMTLEKRXBOYSDIVWZNH
32	AKEFBDJPUIGVNWZHOSQRYCXLTM	82	QTEWPORKXCIZYJVBMFULANGDHS
33	ZMPKSAUBLWVQDNTGOFEXHCJYRI	83	TLCGDRXFSZIAJUPMEVOHKNQWYB
34	JISEFBDYAMRTLPCQHXKGUNVZWO	84	XKTJDPWLVOEZGRUCYANQFHBIMS
35	CNDTUSILPVFYHMZWRBAKGOJXEQ	85	TBUFMIHRAGCWPVJLXZQYDOSNKE

36 MATXNJLHVZUKCDBIPQFEWOGYRS
37 IMVEJAFZHQRGLKUPWTNCSYDOBX
38 NYJQLMDARHGOFKBPETWZXUVSIC
39 ZAVEPUWSOKCHNLRTQMJDYFXGBI
40 MFKTUREJYOCGXIZLQPNBVHDAWS
41 RQICZOHVMNWELSUGXYTJKABFPD
42 CPJMODWQXIFYLRNUKBGZVTESHA
43 LZMACJPOSQTKGINXVFWRHUBYED
44 VNTAYRIPFEDMGKWBQJUXLSOHZC
45 ZIWKNJMSVXFEUDGQACHLYBTRPO
46 FEWRDMLBNATKGVCSZJYOQIXPUH
47 NVEHYGLSTBIFXUZPCQRDAMWOJK
48 ALEGPMUOSDFYCHTWQIVNJXKBZR
49 DJNGOUHQYAELRWBMTXSVCIPFKZ
50 VQSEGULNCBMZJOWRYXPAKDTFHI
86 WMLHSAGZEPKYJUIXVQBCRNFDOT
87 VLNTRGHDUIKEFMSWZJQOYAXBPC
88 YUVBSKNOPEIQDMCWAHJZFGLTRX
89 LYPFDHWBZNCTXROGKQIEUSJAVM
90 EKAQCSILMFJYUZBTXWDPGVHRNO
91 LRQJSDPTYWGBMAINKEVFOXUHCZ
92 AJNRXTMZKHBGOSLEFVCUPDIQYW
93 TCEWPVDIQBJONKYLRZGMFUAXSH
94 BMAOCFRZDWVTGXSHQJEINKUYPL
95 XKVDFJOPIWZQNHAGLERBSCYUTM
96 HZNUGFQRWCEADVSPMOBJYKLITX
97 BPTLKUHZMFADEVOXGSRQIWNJYC
98 MTCISYWNVKOZJELUFPGQBXRDAH
99 BANGXEZSMJPTDWCOHRYFIVLQKU
100 YVTSFOXPBRMKDIELGAQHWCNUZJ

Part XI
Cipher machines

Unit 177
Enigma

The *Enigma* is a cipher machine that was first invented for commercial use, but was used by Germany (and others) in World War II. No book on mechanical cryptography would be complete without a description of the machine and some of its variations, but for the history around the Enigma, we refer you to Kahn's books cited in the references.

The significant innovation in the Enigma is the use of *rotors* (also called *wheels*) to generate permutations of the alphabet in a very-long-period polyalphabetic substitution. This innovation spread beyond Germany and was incorporated into cipher machines in America, Russia, and elsewhere; in Units 179 and 180 we will look at two such machines. A *rotor* is a wheel with a pin for each letter of the alphabet on one side. On the other side are the same number of connections. Inside, the rotor has hidden wiring that connects each pin on one side to one connector on the other. The result is a permutation of the alphabet, whose letters are sent through the rotor with an electric current. When a rotor is rotated to a new position, the permutation changes. By putting several rotors in sequence, and by rotating them separately, the alphabetic permutation can be different for every character in the plaintext.

Figure 177.1: An Enigma with four rotors. Photo by Greg Goebel

The Enigma was originally configured to use three removable and replaceable rotors, and later the German navy added a fourth. There also was a removable and replaceable *reflector*, so that the signal for a letter of the plaintext left the keyboard, passed through the three rotors, the reflector, and the inverse of the three rotors, to emerge on a wire that lit an indicator representing the ciphertext letter (see Figure 177.2). The machine is laid out such that the signal enters the rightmost rotor first, which happens to be the one that we list last in the key. The rotor listed first is leftmost and closest to the reflector. The reflector by its very nature must have a self-reciprocal (involutory) (degree-2) permutation. The overall permutation of the machine is therefore also self-reciprocal. In the following equation in operator notation, we use U for the reflector ("Umkehrwalze" in German) and W for the rotors ("Walzen" in German), to distinguish them both from the symbol R that we use for the Caesar rotation. The current positions of the rotors are denoted with r. From the equation, we can see that the overall permutation is a conjugation of the reflector's permutation; therefore the overall permutation is also self-reciprocal and has degree 2.

$$c = E\,p = W_3(r_3)^{-1}\,W_2(r_2)^{-1}\,W_1(r_1)^{-1}\,U\,W_1(r_1)\,W_2(r_2)\,W_3(r_3)\;p = X^{-1}\,U\,X\;p \qquad (177.1)$$

Because of this property, a letter is never enciphered to itself; this weakness makes it easier to place cribs and helped cryptanalysts break the Enigma during the war. See Tables 177.1 and 177.2 for the permutations of the reflectors and rotors that are relevant to Germany and its military. For those models using four rotors, the reflector must be one of the thin ones, and the first rotor must be β (beta) or γ (gamma). In such machines, the first rotor is placed in a position specified by the key and does not rotate. Reflectors are always placed in the same position.

Figure 177.2: Diagram illustrating how the signal passes through the rotors and reflector of a three-rotor Enigma machine. The three rotors are all in position 0. *The path changes when one or more rotors advance.*

The β and γ rotors are thinner than the others, so that one of them with a thin reflector occupied the same width as one of the normal-sized reflectors. Furthermore, there are configurations of thin reflector with thin rotor that mimic the thick reflectors so that the four-rotor machine is backward-compatible with the three-rotor machine. In terms of alphabetic permutations, for this to occur, we need the thick reflector to be the conjugate of a thin reflector:

$$W_{\text{thin}}(0)^{-1}\, U_{\text{thin}}\, W_{\text{thin}}(0) \;=\; U_{\text{thick}} \tag{177.2}$$

The choices and positions of the thin components that satisfy Equation 177.2 are B thin with β in position 0 and C thin with γ in position 0, where by "position 0" we mean the position in which their effective alphabetic permutations are those that appear in Table 177.2.

We would like to point out that the rotation of a rotor is equivalent to sandwiching its operator between a Caesar rotation R of the alphabet and its inverse (look back at Unit 20):

$$W(r) \;=\; R(r)^{-1}\, W(0)\, R(r) \;=\; R(-r)\, W(0)\, R(r)$$

Then we can rewrite the encryption operator in Equation 177.1 as

$$\begin{aligned}
E \;&=\; W_3(r_3)^{-1}\, W_2(r_2)^{-1}\, W_1(r_1)^{-1}\, U\, W_1(r_1)\, W_2(r_2)\, W_3(r_3) \\
&=\; R(-r_3)\, W_3(0)^{-1}\, R(r_3)\, R(-r_2)\, W_2(0)^{-1}\, R(r_2)\, R(-r_1)\, W_1(0)^{-1}\, R(r_1) \\
&\quad\; U\, R(-r_1)\, W_1(0)\, R(r_1)\, R(-r_2)\, W_2(0)\, R(r_2)\, R(-r_3)\, W_3(0)\, R(r_3) \\
&=\; R(-r_3)\, W_3(0)^{-1}\, R(r_3-r_2)\, W_2(0)^{-1}\, R(r_2-r_1)\, W_1(0)^{-1}\, R(r_1) \\
&\quad\; U\, R(-r_1)\, W_1(0)\, R(r_1-r_2)\, W_2(0)\, R(r_2-r_3)\, W_3(0)\, R(r_3)
\end{aligned}$$

In other words, we can think of the rotors in the Enigma as remaining stationary, with changing Caesar shifts inserted between and aside them. Including the ring settings would have the same effect as changing the rotor positions in the above equation.

There is a further complication regarding the rotors. At first, the letter labels on their edges were printed directly onto them, but later they were printed on rings that could be fixed onto the rotors in different positions. This does not affect the way the alphabetic permutations are applied, but it does introduce offsets into the key (see below for how the key is specified) and does affect how the rotors advance between the encipherments of the plaintext letters (also see below regarding rotor advancement).

We must now discuss the rotation of the rotors in the Enigma machine. When the operator presses a key on the keyboard, the advancement of the rotors occurs before the plaintext letter is enciphered. The rightmost rotor (which is listed last in the key) advances one step every time a key is pressed. There are one or two notches on the rings for each rotor (except β and γ) that control the advancement of the rotor to its left. See Table 177.3 for the locations of the notches. Since beta and gamma can only be used as the rotor next to the reflector, they do not have these notches. If a rotor is in a notched position, then when it advances it causes the rotor to its left also to advance. The Enigma machine has a bug (which programmers now call a "feature") known as *double-stepping*: When the middle rotor reaches its notched position, then on the next key-press, it advances again along with the rotor to its left. For example, suppose that the rotors in use are I, II, and III. Rotor III is notched at position V, and rotor II at position E. So the middle rotor advances after the rightmost rotor reaches V,

as in ..., AAU, AAV, ABW, ABX, ... When rotor II reaches E, the double-stepping occurs, as in ..., ADU, ADV, AEW, BFX, BFY, ...

Later models had a *plugboard* ("Steckerbrett") that allowed the transposition of up to ten pairs of letters before they entered the rotors. After leaving the rotors, the plugboard applied the same transpositions again. This had the effect of adding another alphabetic permutation (S) to Equation 177.1:

$$c \;=\; S\, W_3(r_3)^{-1}\, W_2(r_2)^{-1}\, W_1(r_1)^{-1}\, U\, W_1(r_1)\, W_2(r_2)\, W_3(r_3)\, S\, p$$

We did not write "S^{-1}" on the left because S is its own inverse.

The encryption key of the Enigma consists of these things:

- the choice of reflector
- the choice of the three or four rotors and the order in which they are placed
- the ring setting for each rotor (one number [1-26] for each)
- the plugboard connections (a set of up to ten pairs of letters, with no repetition)
- the starting positions of the rotors (a sequence of three or four letters)

Typically, all but the last of these were configured each morning as the internal daily settings. The machine would be locked and only the starting positions of the rotors could be changed during the day. They served as the message key and were, ideally, different for each message during the course of any given day.

Figure 177.3: The two sides of rotors used in the Enigma. One side has pins, the other pads. The advancement notch is visible in the ring on the rotor to the right. Photo by Ted Coles.

Let’s try an example of encryption with the Enigma machine. Take this plaintext:

THISXMESSAGEXISXENCRYPTEDXWITHXTHEXENIGMA

and use the key B, II, I, VI, 5, 7, 9, GELATINOUS, GOO. On the first press of a key, the rightmost rotor (VI) advances to P and the keyboard emits a signal on the line representing T. The signal passes through the plugboard to emerge on the line for I, since (I T) is one of the transpositions specified in the key. Next, the signal enters rotor VI, but since the ring setting is 9 and the rotor is in position P, it enters the P pin (8−8+15 = 15 modulo 26, and we must take 1 less than the ring setting because we consider A to be zero). Rotor VI then emits its signal on the pad for R (see Table 177.2), which becomes K after we add the 8 and subtract the 15. The middle rotor (I) has its ring at 7 and is in position O, so we subtract 6 and add 14 to K to find that the signal enters on the S pin. The signal leaves rotor I on the S pad, which becomes K after we add the 6 and subtract the 14. The leftmost rotor (II) has ring setting 5 and is at position G, so we subtract 4 and add 6 to see that the signal enters on the M pin and leaves on the W pad. Adding the 4 and subtracting the 6 gives a signal entering reflector B on the U pin. The signal leaves the reflector on its C pin. Subtract 4 and add 6 to see that it enters the E pad of rotor II, so it leaves via the Z pin. Add back the 4 and subtract the 6, then subtract 6 and add 14, to find that the signal enters rotor I on the F pad and leaves on the D pin. Add back the 6 and subtract the 14, and subtract 8 and add 15, to find that it enters rotor VI on the C pad and leaves on the X pin. Add back the 8 and subtract 15 back into it to get a signal on the Q wire. The plugboard does not affect Q, so theis is the ciphertext letter. This painful process continues until the entire text is enciphered. The complete ciphertext is

QXRYIASMEJFZDGZMHYNBBRLFNRDEQDVPNMHAEJAYH

If you work through the details, you will notice that a double-stepping occurs when the D of ENCRYPTED and the following X are enciphered. The positions of the rotors go from GPM to GQN to HRO.

reflector	input ABCDEFGHIJKLMNOPQRSTUVWXYZ
A	EJMZALYXVBWFCRQUONTSPIKHGD
B	YRUHQSLDPXNGOKMIEBFZCWVJAT
C	FVPJIAOYEDRZXWGCTKUQSBNMHL
B thin	ENKQAUYWJICOPBLMDXZVFTHRGS
C thin	RDOBJNTKVEHMLFCWZAXGYIPSUQ

Table 177.1: Alphabetic permutations of the Enigma's reflectors. Later versions of the machine have a D *rotor whose wiring is changed as part of the key.*

rotor	input ABCDEFGHIJKLMNOPQRSTUVWXYZ
I	EKMFLGDQVZNTOWYHXUSPAIBRCJ
II	AJDKSIRUXBLHWTMCQGZNPYFVOE
III	BDFHJLCPRTXVZNYEIWGAKMUSQO
IV	ESOVPZJAYQUIRHXLNFTGKDCMWB
V	VZBRGITYUPSDNHLXAWMJQOFECK
VI	JPGVOUMFYQBENHZRDKASXLICTW
VII	NZJHGRCXMYSWBOUFAIVLPEKQDT
VIII	FKQHTLXOCBJSPDZRAMEWNIUYGV
β	LEYJVCNIXWPBQMDRTAKZGFUHOS
γ	FSOKANUERHMBTIYCWLQPZXVGJD

Table 177.2: Alphabetic permutations of the Enigma's rotors in position 0*. In the forward direction, input is on a pin and output on a pad.*

rotor	location of notch(es)
I	Q
II	E
III	V
IV	J
V	Z
VI	M, Z
VII	M, Z
VIII	M, Z

Table 177.3: Location of advancement notches on rotors in the Enigma.

Reading and references

Mark Stamp and Richard M. Low, *Applied Cryptanalysis: Breaking Ciphers in the Real World*, Hoboken: Wiley, 2007, section 2.2.

Wikipedia has many pages on the Enigma. Start with these:
https://en.wikipedia.org/wiki/Enigma_machine
https://en.wikipedia.org/wiki/Enigma_rotor_details

Philip Marks and Frode Weierud, "Recovering the Wiring of Enigma's *Umkehrwalze* A," *Cryptologia* 24:1 (2000) 55-66, https://doi.org/10.1080/0161-110091888781

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, pages 420-423 and elsewhere, https://ia600606.us.archive.org/30/items/B-001-001-264/B-001-001-264.pdf, https://archive.org/details/B-001-001-264

David Kahn, *Seizing the Enigma: The Race to Break the German U-Boat Codes, 1939-1943*, New York: Houghton Mifflin, 1991.

Programming tasks

1. Simulate the Enigma in Python. Allow for the possibilities that either three or four rotors are used. Implement all reflectors and rotors that are presented in this unit.

Exercises

1. Decipher this ciphertext. The key is B, I, V, III, 5, 9, 15, `PS`, `AY`, `IJ`, `ER`, `VN`, `MAX`. An `X` has been placed between words in the plaintext, and `XX` between sentences.

```
LZJWKMOPGWGKESNUONOICEWIAULJIQTLRUZKAANKPWWGVQHRCWVZZERYGGZBBLCY
BSZWPLWGSVUWFRKGLHPRJDHJYOSTDPKRGAMJEZCNFMQGBAHYBALYMUJCPOOJGEDN
NKHBHMAFMUGLQDCYPESQZDIYOQWQSTZSUMDCMQSSXKHGYKMLTZKFDXYZCFXKWJZA
EPSVIZVUIQAVWGRJAW
```

2. Decipher this ciphertext. The key is C, II, IV, VI, 13, 25, 7, `DECRYPTION`, `ABC`. An `X` has been placed between words in the plaintext, and `XX` between sentences. This exercise will test your implementation of double-stepping.

```
TFELTJWHTSFJBHTRBBOWLHNRLMNDWHGZZIGNYHWKKWJQTNCKQQTNWNJRSWDJPGDM
ILQOIYPPYJUZFWCYZKPJZSWISAISUJECOQAHCFVUDJTQQRYEHVUGEIYHRMDBXNNB
IFOZDIARLAOXTFYMIWAPRACSFXZDXADRKJNJUVVVNZMLYZZOYAUCTMHIVVPYCXVI
DTQDEZGVEGMMTCXZUIGRUCRCDWDXEYHMSVVVKKLJWJWAWYONPYBUKWKBIESHKYYF
RGQAMDIXEBXWQSXLPG
```

3. Verify Equation 177.2 for U_{thick} = B, U_{thin} = B thin, $W_{\text{thin}} = \beta$ in position 0 and for U_{thick} = C, U_{thin} = C thin, $W_{\text{thin}} = \gamma$ in position 0.

Unit 178
Attacking Enigma with the index of coincidence

Analysts during World War II used every bit of their computing power to break the Enigma. In the modern world, we have so much computing power that we waste most of it. A statistical attack like the one we present here uses a lot of computation. We are going to use the index of coincidence (IoC) (see Unit 12) to help us find the configuration of an Enigma machine that was used to encrypt a ciphertext. In this unit we will restrict ourselves to a three-rotor machine, but the method can be generalized to four-rotor versions.

The attack has these three stages, which we will discuss in more depth below.

1. Try all choices of reflector and rotors and (provisional) initial positions and keep the configuration and starting positions with the highest IoC.
2. Keep the reflector and rotors from stage 1, but vary the ring settings and initial positions and keep those with the highest IoC.
3. Try plugboard wirings one pair of letters at a time and maximize the IoC.

In the first stage, we try all possible combinations of reflector and rotors. Remember that a rotor cannot be used twice in the same configuration, since each machine was outfitted with one of each. We also try all possible starting positions for the rotors (the message key). In this stage, we fix the ring settings to be at 1=A for all rotors (we could take any arbitrary ring setting, but it should not matter much). There are three reflectors, eight rotors, and twenty-six possible starting positions for each rotor, so we have to try $3 \cdot 8 \cdot 7 \cdot 6 \cdot 26^3 = 17{,}716{,}608$ configurations, and therefore we recommend a compiled programming language such as C, rather than a slow interpreted one like Python. The configuration whose resulting plaintext has the highest IoC is kept for stage 2. The winning margin may not be large, so we should take some care; if there is no clear winner, perhaps the method has failed and we need to try a more exhaustive search (see below).

We consider the initial positions of the rotors found in stage 1 to be provisional because the true ring setting can affect whether the rotor acts more like it starts in one position or more like in a neighboring position. So in this stage we will give some leeway to the rotor positions, and allow it to have a starting position one place earlier in the alphabet. The ring setting on the leftmost rotor is irrelevant for a ciphertext-only attack because it only affects the locations of the stepping notches, which aren't used for that rotor. This ring setting is important for cipher clerks, because it affects the message key. But for us, we can fix the ring position to 1 and not allow it to vary. As we increase the ring setting of a rotor, the starting position increases with it. If, for example, in stage 1 we found the best starting position for the middle rotor to be G, then we should try these possibilities for that rotor (remember we give it some leeway):

ring setting 1	starting position F
ring setting 1	starting position G

ring setting 2	starting position G
ring setting 2	starting position H
ring setting 3	starting position H
ring setting 3	starting position I
⋮	⋮

We need to also give leeway to the starting position of the leftmost rotor, even though we do not vary its ring setting. So there are $2\cdot(2\cdot 26)^2 = 5{,}405$ possibilities to try. The one that gives a plaintext with the highest IoC is kept. In this stage, there should be a clear winner.

The action of the plugboard causes more apparent randomness for the cryptanalyst. But for each pair of letters swapped by the plugboard that is correctly identified, the randomness can be reduced. Hence, every time we account for a pair of letters cabled on the plugboard, the IoC of our plaintext will increase. To find the first one, we try $26\cdot 25/2 = 325$ pairs (divide by 2 because AB is the same as BA, etc.). We keep the pair that gives the greatest increase in IoC and include it in all future decryptions. To find the second pair, we try $24\cdot 23/2 = 276$ pairs (we don't include the letters of the first pair in the search). When we reach a point where adding another pair to the plugboard reduces, rather than increases, the IoC of the plaintext, then we stop. At this point, the attack is finished.

We must, of course, look at an example. Consider this ciphertext that was encrypted with a three-rotor enigma. It is long enough that we are confident of success.

```
NNBOFZRDRLRSYRRVJPSXPSZMWHUHESXPFKKHYRNUFYMYXQJPMOLTPMRTKKCVJQNT
HAPVQBFHJWVRXWDOMQNWRJOJXIPBBHZVLPSGZJCTVUMMUUEFHEKNYHWIXVCJQKSJ
IEZLSUXZLRLKJBOYCZHNYUCKWPAIYCMUTKUFHMCOXWLEMSYWKZYBOFZCPTQZKNZQ
DDGTPHRMFREBNMQUFMRELRGAFXVRWKAOEVKBQIMGMTIBKBGLRRAYBOIRXEQLFOZA
QTDFDNUAGDHVATDTBSOXAFTHZAHWKAWACQKLRKCJTNQEWODQJEUMNQXKCAYQVVBL
KMDQNIMEHKVMHENUTIPTAPNCQFPHVKEJLDEIDGXGRYJAZJILHLJUEJNYBFTUPEMP
FXUXBICDFBUDYLMJMDKTUHWKKLOKJNAXRIUDIHVSDJTSTRRNNZKIAVEYASORARRS
TKUWJNTFYCSOSQEFGZKJPFDDHPXSUJTRFSCZPJKPZVQLNNTGODWCTCKEUBJWXNNA
WFBJUQXEDYIVTYJMNLSZQQMQVKVNNORUFPZPXTHZQIDHTFDYYXVJMAXLWDWHMUWS
BKTFYMJPKGNGILAUPPVXNWLUDAISQPLZUZDGXRFOHUYJRRBHWIFHVBJQXBGVAFTT
IZGXXZVGOEOUUUXSUXJKMBSEPTLOBZITMNIKYQBHMVIIHAOJNILNNIKCEKERRAOY
YNPWLCYQRGUALYOZFUZVSHXDGKLSDRLFZVKPDZVMKUHMJNMZXBSWLZGIURVJIQZX
STUVXIXLDHOKNEDJLXCKMEMDWVWAMPZZZOBZYVHMDNCJHXCOJHZUHEPYWHKRJMVH
STVUPGRZEJZBTOJRCTQFYLOQFCWMDDBKHRKUXXUOVYBOQZTGJBQKDBQLENNNXBOV
OTSOVMSXGIMIGXFIPMEFFUAWUYZBQBEAXJFDWBDUAAPRYRXNYBMIH
```

In the first stage of attack, we try all possible reflectors, rotors, and starting positions. Ring settings are all set to 1. We sort from lowest to highest IoC and find that the best is when the reflector is B, the rotors are II, III, V, and the starting positions are KEZ. It's a narrow victory, but we will run with it. If we find that we cannot solve the ciphertext, then we can always come back and try the second-place configuration.

reflector	rotors	starting positions	IoC
⋮	⋮	⋮	⋮
A	VI IV VIII	IZS	1.0603

C	VIII II V	QHT	1.0607
B	V VII VI	NIK	1.0625
B	II III V	KEZ	1.0635

We keep the choices of reflector and rotors, and now concentrate on the ring settings. For the fast rotor (on the right), we found our result above for ring setting 1 and initial position Z. So we try 2 and A, 3 and B, ..., 26 and Y. But because of the uncertainty we have in the starting positions, we try these plus or minus one step. At the same time, we try varying the ring setting and starting position in a similar fashion for the middle rotor. For the slow rotor, to be safe, we allow the starting position to vary by one step, but still keep ring setting 1. After the search, here is the end of our list, sorted by IoC:

ring settings	starting positions	IoC
⋮	⋮	⋮
1 17 7	J U F	1.2585
1 16 8	J T G	1.2659
1 17 9	J U H	1.2833
1 17 8	J U G	1.2943

The best is for ring settings 1, 17, 8 and starting positions JUG. We continue with the assumption that we have found the correct settings for the rotors and now turn (pun intended) our attention to the plugboard. There are (26 · 25) / 2 = 325 ways to connect two letters. If we decrypt the ciphertext with our favorite rotor settings and each possible choice for one wire on the plugboard, we find these IoCs:

plugboard pair	IoC
R V	1.1833
K R	1.1899
⋮	⋮
C F	1.3799
P T	1.3888
B X	1.3912

The last few are all good choices, but to follow our method, we keep only the best pair, BX, and repeat the procedure assuming that this pair is correct and look for a second pair. This time, we only need to consider (24 · 23) / 2 = 276 new pairs.

plugboard pair	IoC
R V	1.2652
I V	1.2686
⋮	⋮
C F	1.4838
A L	1.4876
P T	1.5027

So we keep the pair `PT` and continue to find the next pair to be `AL`, giving an IoC of 1.6081. We are now in the range of typical English text, so let's pause for a moment and look at the resulting plaintext:

```
ZEENTRMAGEGINMAOCMOGSIEURBOURDOGRSMAIGHYXGOBJEFTIVEIMPARTIHLAGSI
MPERSOAALSTUDYAGD...
```

OK, we're not finished yet. We can see some words peeking out at us, but it is clearly not correct. So we find a fourth pair, `CF`, with IoC 1.7200. Let's pause again and see the plaintext:

```
ZEENERMAGEGINMAOFMOGSIEURBOURDOGISMAIGHYAGOBJECTIVEIMPARTIALAGSI
MPERSOGALSTUDYAGD...
```

There is marginal improvement. Searching for a fifth pair reveals `GN`, with IoC 1.7960. Trying to find a sixth pair only *reduces* the IoC for this ciphertext, so we may be at the end. When we check, the plaintext now looks correct:

```
THEGERMANENIGMAOFMONSIEURBOURDONISMAINLYANOBJECTIVEIMPARTIALAND
IMPERSONALSTUDYAND...
```

We have found the entire key. The plaintext is from *German Problems and Personalities* by Charles Sarolea:

> The "German Enigma" of Monsieur Bourdon is mainly an objective, impartial, and impersonal study, and the author has been careful not to obtrude his own private views. It is only in the last chapter that he attempts to draw the lesson and point out the conclusion of his own inquiry. And his conclusion is an eloquent though restrained plea for a Franco-German *rapprochement*, and in favour of the only policy which will bring about that reconciliation. France, he argues, does not want a revision of the Treaty of Frankfurt. She does not want compensation or revenge. ... A French statesman, on the eve of the Treaty of Frankfurt, made the rhetorical statement that France would never surrender one stone of her fortresses nor one inch of her territory. Animated by a very different spirit, modern French statesmen do not claim back to-day one inch of lost territory. All that the French people demand is that the claims of justice shall be heard, that Alsace-Lorraine shall cease to groan under the heel of an arbitrary despot, that Alsace-Lorraine shall be governed according to her own laws, that the Alsatians shall be treated as a free people, and not as conquered subjects.

In the case where too many cables are used in the plugboard, it may be impossible to succeed in stages 1 and 2. When that happens, we need to try all configurations of reflector, rotors, ring settings, and initial positions of the rotors. For each, we try to maximize the IoC in the same way as in stage 3. Only when we find an acceptable plaintext do we stop the search. This can be a long and difficult process, so it would be best to use a fast programming language, such as C.

The index of coincidence is not the only statistic that we can maximize to break Enigma ciphertexts. See references for refinements of the attack and the use of other statistics.

Reading and references

James J. Gillogly, “Ciphertext-Only Cryptanalysis of Enigma,” *Cryptologia* 19:4 (1995) 405-413, https://doi.org/10.1080/0161-119591884060

Note that there are 325 possible two-letter plugboard wirings, not 625.

Heidi Williams, “Applying Statistical Language Recognition Techniques in the Ciphertext-Only Cryptanalysis of Enigma,” *Cryptologia* 24:1 (2000) 4-17, https://doi.org/10.1080/0161-110091888745

Olaf Ostwald and Frode Weierud, “Modern breaking of Enigma ciphertexts,” *Cryptologia* 41:5 (2017) 395-421, https://doi.org/10.1080/01611194.2016.1238423

Programming tasks

1. Implement the attack on a three-rotor Enigma in which it is possible to do stages 1 and 2.

2. Implement the search on an Enigma that has too many plugboard connections for you to succeed with stages 1 and 2.

Exercises

1. We can start off with something easy. This ciphertext was encrypted with a German army Enigma without a plugboard. The army used three rotors chosen from only I, II, III, IV, and V, and reflectors B and C. To increase the IoC, X has been placed between words and XX between sentences.

```
XBCWHIYYBMIUFWJBFMBNCYRABCJABKUHIWUTWUYTIDYMWYHZQWOVFGUQELLAZDGY
BGNYEIWXTHISOPYETEASUBYSVJRDBPEAIWSBBPCQGRRHZDNFMQCOCPISCFNIXZCJ
LCEBZFKDDBOQRHWAEGOFOKHAXVCSVWTPQHYBVTAEIMXHWMUZZLUYRKNCTJEEZXBI
KAUCSBGLDDTPVLFWQXTZRTYBVOYJVXXGWJDZXOEHYPTKTOIPSEGCLWFMOKSABZWD
ESGNSPEFGKNYFIPPJFWTRTKYOSLLGPHLWRJUBVLEFDRDCUIWGYURLWNUPTSHDPOV
MIYDWFXLOPDFAZRPQGWEBEILBDNDCDKDRLHTCIYNOMRYLVZZFNWDHVTBWDXMRHUE
SEEBPOYLJVZXVCACJYZJYSHYAUPMOOHBFYBDSTSOXUVETTSXLSSPLRHRTARRIAVO
QKCRKGWEGXQAUAVSZEDPIAJUNSARRQNJZUWQHOBSTYAILRLECLWUABQJHPROOCPE
ZUJYWIJZMVQGGCEBVHOVGHVVPCBNVVNNXXASQHMKPSZPWLRMMW
```

2. Break this ciphertext that was encrypted with a three-rotor Enigma. The plaintext is in English. To increase the IoC, X has been placed between words and XX between sentences. There are five cables in use on the plugboard. The rotors were chosen from all except beta (β) and gamma (γ). The reflector is B or C. There may come a point where you have to try two forks in the road.

```
CCFZIAPYHFQWOWJULWBHWFGIXPYDRGEDDSBXCGVXLQCPWMXEMYRWJKFUQPFGDESI
PRJIIWBISATECJCFRNAXWCDKQPDDRSPMHACXVJYZLZKABALXTSWISWVCLJNIZOAK
LVLNGHCHFPMLBGGCPKROXBWDUFCRHBNLNJHJJEGVFQNSAOQUZESPDGMHJVCRMAUX
KQRMWQQFBOKDNOANLHTGGHEGVDJGERNNQMFBVPNSWTYPRLPADUCYPNGWBCCQRDRK
MFAZTXICVIAGNBXKZTCWATGRERОIDTDENCZVEUTJIIGGXBEVPBAUEMEKPUWNRLNF
MIJXCAYKVKCKEHKMFKJADUSMCYBTVNLHWFKNPXEHQOGUHMJDRIOBQEWRSMCGSVCO
VYPJLMJCNCEVXBAPKTERFWAKDPVVCZBGXYLUHKTEIJCYRQZZLCPCNFLFHGMSDAUK
```

```
UIWJIVNBRHKRUMNGTBUMOGKJFZTNZMOUKFKCJVKAXNRPVRLDQUMLVMIMFVKGTPRK
TPPZXQUHXKOFDQFZPRBHWOFBUYJDPDQDHLGZKEHRSVRTEGJBWPMRTTZYZHAUAGTR
EOUPMTGLHOHMAKIAUQPUOOIWIGBMGECRQZQVWVBBCLGDDNSAMYMUAAERFVNPVINT
YGSKTYXTCPTBURVRGIXEKIQQMWLWYDOMAAKBUNPEZYMPLMPZVAKKAAFBGGMYQRRA
PDUNFPROCZBBUDJVHGYYHXOAWOZCVLFUVTELHKFGWBKZPTCTXJFLUKVDHXVKOWTK
BNEUAEYMHLGEPILRCVPWEGLIOBIUUZWMEJPPIDJTCRXJVBQUZRHQTBYDEVEFCOLH
JWVGBBSJDEDLGKQSIVVLSTZWBNHLKCNUKSUSFEWIHEDGYTFWWJKHNZKCEMYOZGKV
QUIPGGPSDZALLMDMJUYWIGUSFQUQVPQYZCTMUQTYBAZGTUJFFWQTHJGHUIAGRGSN
ZNBYNLNTZPJJGMDFTWBHUQTYFGVMVPYBNTAODQFSQPYLLDZGYHAGZJIWKTDEORYP
AIHTVDOIZYXSTUWKBLIGCCQTZJUQVEPIEUTTVPLKZJZNYOSTUXYEJJFKZFLOUAWN
NWGXCMYCCRDAGONYQBYBGOJMQPPGIYTRFCVXSZVOTFKNGGCPVLVSPQRBFUJFZKPE
ZHYKTPVWVFBUBIFLQJMLTHBTKE
```

3. Decrypt this ciphertext that was encrypted with a three-rotor enigma. There are ten cables in use on the plugboard. The reflector is C and the rotors are II, I, V. The plaintext is in English; believe it or not, the word “enigma” does not appear in it. Remember that the first ring setting is irrelevant.

```
UOPIIKUDEZGWQFXUVYSTROHVNAYLAKHPJVRIVEPNPTTMHXGJYFWJQPAWMZVQFPYB
FZYMYJKOFROAABKIXSFBOYFWTHVNYMWEJKYWVCHCJMFHORYLNOLLIUYNZETGGZOQ
UUANZXKQWLNYNZEXDHWAKXPXLMDULPTANBGAYTWOBLYVGVFSNQDEQQGVZAMEGADB
JPNYRFVUPEJTZKMKUUJUHOHFJNTBLACTDJFRQTQFPTGWSXMVSRIAKVOJGOQTRZGX
ACYYRZQMSRBOMNRDZGDKYQZJMLOOWGIIBIUMHYBMBXSFSLYQGBWXYUDAPWDGYARW
JGMTDXNDYBYRNZFWTAPLHBZPZADJDBZODAYGTKQCZIZQLKYUFXRDXZJEVYEUCZCF
MUGYMTRCQXNMJGHWYDYLERSXUPNCZICUZHSAPPBTZMTDUWNOKEKRRESQJIKKLQVP
VAUJHXTQJAIOVBOSPENPMWHFERFAWJDLISNIFFGTBBFYCBFJSXOFOZYMWDMPEBIY
VRTDTHKYSXDYGRALISAJBWFQXKVYWTYFMORTYDVPKYXFXWYEOSEWGPIZOQNBPYQH
DVNDNQYRJCAVXZKZIXZEXDSGBEVAVQGTGPSJUUXNTISCYPRMNZWCXVWLRUTBEXYG
UNLBKASRUPIFXKWXCPWQTVIMRHBCMNRSOVGWZZQBFPEVRWCVRGSRMAJBQXLVGEGL
HQRULPWLXYOHPKZMLIKPCZASULNRWJRFMYVVTFBXBPICXHZCOGZQEXPSSOLUIIXD
ZM
```

4. Decrypt this ciphertext that was encrypted with a three-rotor enigma. There are ten cables in use on the plugboard. The plaintext is in English. This could take a long time.

```
HQWSEQGSTBGBHDBGVNYJMZHFZNIFOCTADLAHXYJXCHMQHWMEAXVZGHBMLDJGELOG
DUVVJDHWUHIYAEVCWHQSCLYHVOTKYXMOLYRIDAVVJXYGEBJUYCBKREATBYVWRBQI
EJRARBSXELJJFVJOMSJUDTKIKGQQTUINRAZLGXRTEFSDGDMNFYVAGARQNXLMAQVH
TFNVHCGJXVQQIRGKCGUWHVFZXSQRHKNJCLTKDXZZOJFWXUNLJEAPDSUXNOLDWNNR
MQMOZOOVLKRMGZVTMMKNHZIZXUIQKJPDRQSCBUFRKKDPPPIRPZBDFUOLNOPBOMGV
RFIZXWONCNFKYXPGBDOPKBQDIPLBFSDCLVMAVDFGQPMKYPCSDVKUQBHZPQRHBUZG
LMNTXVGZFFQWPOUFMCUCHLDMGLIHYIBKBDRHBNZDVZYTJJRIGYMDBEHGKCUZPTQW
IDJTZVMWXODOKUWBBGKRWPXHEWJOLEWECGCPDKATIYTJCENQKKTOLCJDIPIZCZPA
CEVIDGSSXLIQMGHLOLT
```

If you enjoy breaking Enigma messages, there are some challenges (with ciphertexts in German) at this site: https://www.ciphermachinesandcryptology.com/en/challenge.htm But note that in Enigma messages, the Germans put `X` between sentences, `J` around proper names, used `Q` for `CH`, tripled letters in designations such as U-boat `U235` becoming `UUUZWEIDREIFUNF`, and, worst of all, wrote in German.

Unit 179
Attacking Enigma with cribs (Bombe)

The first step in using a crib is to place it against the ciphertext. We saw in Unit 177 that the Enigma never enciphers a letter to itself, and this property can help place a crib. For example, consider this ciphertext and crib:

```
UWLCDYAPLKYAINQIHXJVKMAFAETANCUBPDBASNRREDDFTGRTBXEWSZMQPNLPDGZV
TFEXMPQLTFYGPALMZXBJWRZWRGYTBNFRTWBUJBDKEACCVRYYOREBUWUWGDBDRYRU
WNECGDVBORVFEPIGHF

THECRIBINTOONEOF
```

This is an incorrect placement, because an E matches:

```
UWLCDYAPLKYAINQIHXJVKMAFAETANCUBPDBASNRREDDFTGRTBXEWSZMQPNLPD...
            THECRIBINTOONEOF
```

There may be several placements that satisfy the condition; here is one:

```
UWLCDYAPLKYAINQIHXJVKMAFAETANCUBPDBASNRREDDFTGRTBXEWSZMQPNLPD...
              THECRIBINTOONEOF
```

For a moment, let's think about a three-rotor Enigma with ring settings fixed and without a plugboard. Suppose that we have a ciphertext and know some of the plaintext (a crib placed correctly). There are two or three choices for the reflector (the Germans did not use reflector A in the war), $8 \cdot 7 \cdot 6 = 336$ ways to place three of eight rotors in the machine, and 26^3 message keys (starting positions of the three rotors), for a total of 5,905,536 possible keys. This is not a big number for those of us living in the modern age. We can try each of them and check to see if the portion of the ciphertext covered by the crib is decrypted correctly, then run the machine backwards to the start of the ciphertext to find the message key. This is computationally less expensive than calculating the index of coincidence or textual fitness of the decryption. We can hasten the process by checking the first letter and only continuing with the decryption if we have a match; for each key, stop decrypting when a discrepancy arises or we have a successful decryption.

Now consider a three-rotor machine with fixed ring settings, but with the plugboard. There are $26! / (6! \cdot 10! \cdot 2^{10}) = 150{,}738{,}274{,}937{,}250$ ways to plug ten cables into the plugboard (choosing 20 of 26 is $25 \cdot 24 \cdot \cdots \cdot 7$, which gives $26! / 6!$; the other factors account for the fact that the order in which we assign the ten cables is irrelevant, and the order of the two letters for each cable is irrelevant). Now the number of keys has grown to 890,190,309,219,827,525,120. Now we have a big number, and we haven't even included ring settings. Enter the *Bombe*, a machine and strategy by Alan Turing. The strategy is to try only the non-steckered keys and to map out pathways through the machine in its various states as it deciphers a text, and then to use them to form loops. Checking loops for consistency

allows cryptanalysts to rule out the vast majority of keys. Bombe's strength comes from the ability to simulate many Enigma machines simultaneously.

The best way to explain the strategy is to look at the details of an example. Consider the ciphertext and crib that we saw above:

```
UWLCDYAPLKYAINQIHXJVKMAFAETANCUBPDBASNRREDDFTGRTBXEWSZMQPNLPDGZV
TFEXMPQLTFYGPALMZXBJWRZWRGYTBNFRTWBUJBDKEACCVRYYOREBUWUWGDBDRYRU
WNECGDVBORVFEPIGHF

PUTTHECRIBINTOONEOFTHE
```

The machine used to encrypt it has ring settings 1, 1, 1, and the plugboard has all ten cables in use. Suppose that we found the correct placement of the crib earlier. The next thing we do is to show the correspondence from ciphertext letters to plaintext letters, and the state of the machine for each pair. Remember that the Enigma steps before it enciphers a letter, so when we write E_1, we mean the permutation that the machine uses to encrypt the first letter after it has stepped.

$$\begin{array}{c} \texttt{NQIHXJVKMAFAETAN} \\ E_1 \updownarrow\updownarrow\updownarrow\updownarrow\updownarrow\updownarrow\updownarrow\updownarrow\updownarrow\updownarrow\updownarrow\updownarrow\updownarrow\updownarrow\updownarrow\updownarrow E_{16} \\ \texttt{THECRIBINTOONEOF} \end{array}$$

Now, the full machine E is the steckerless machine M between two plugboards S:

$$E = S^{-1} M S$$

We remember that S is its own inverse (M is, too), so we can drop the "−1," but we keep it for a moment. What we have for the first sixteen letters is

$$\begin{array}{c} \texttt{N} = E_1(\texttt{T}) = (S^{-1} M_1 S)(\texttt{T}),\ \texttt{T} = E_1(\texttt{N}) = (S^{-1} M_1 S)(\texttt{N}) \\ \texttt{Q} = E_2(\texttt{H}) = (S^{-1} M_2 S)(\texttt{H}),\ \texttt{H} = E_2(\texttt{Q}) = (S^{-1} M_2 S)(\texttt{Q}) \\ \vdots \\ \texttt{N} = E_{16}(\texttt{F}) = (S^{-1} M_{16} S)(\texttt{F}),\ \texttt{F} = E_{19}(\texttt{N}) = (S^{-1} M_{16} S)(\texttt{N}) \end{array}$$

Move an S from the right-hand side to the left of each equation (multiply on the left by S on both sides):

$$\begin{array}{c} S(\texttt{N}) = M_1(S(\texttt{T})),\ S(\texttt{T}) = M_1(S(\texttt{N})) \\ S(\texttt{Q}) = M_1(S(\texttt{H})),\ S(\texttt{H}) = M_1(S(\texttt{Q})) \\ \vdots \end{array}$$

In other words, the stecker-less machine connects not the letters of ciphertext and crib, but their images under the plugboard's substitution:

$$S(\texttt{N}) \overset{M_1}{\longleftrightarrow} S(\texttt{T})$$

$$S(\texttt{Q}) \overset{M_2}{\longleftrightarrow} S(\texttt{H})$$

$$\vdots$$

These relationships give us sixteen edges in a graph whose vertices are labeled by S(A), S(B), ..., shown in Figure 179.1. Cryptanalysts call this graph the “menu” for this ciphertext/crib pair.

Nicely, the graph has loops, and they intersect. It helps because if we build an operator that goes around that loop, say from S(T) to S(N) to S(E) and back to S(T), then that operator must not alter S(T). The letter at the vertex that is represented by S(T) must be a fixed point of the permutation that the operator enacts:

$$S(\mathsf{T}) \;=\; M_{14}\, M_{13}\, M_1\, S(\mathsf{T})$$

There may be more than one fixed point of this permutation. Because there are other loops that begin and end on S(T), those other loops have their own set of fixed points. The correct S(T) must belong to all sets of fixed points, and so we can usually eliminate some. Also notice that S(A) and S(O) form a loop of their own. Therefore we require that $M_{12}\, M_{15}$ and $M_{15}\, M_{12}$ have at least the same two fixed points. The reason that there must be two and that they be the same is that we can start on S(A) and go either clockwise or counterclockwise around the loop, and we can do the same if we start on S(O).

For each configuration of the stecker-less machine and message key, we generate the first sixteen permutations that it has and calculate the composition in the equation above. If the resulting permutation does not have a fixed point, then we can rule out that configuration and message key combination. If it has one fixed point, then we tentatively assign it to be the partner of T on the plugboard. If there are more than one fixed point, then there are more options for S(T). We can do the same for S(E) and S(N) and for S(N) around the other loops. And don’t forget the S(A)-S(A) loop. If at any time we find a contradiction, we rule out the key that we have tried. See Table 179.1 for all of the loops in our example. A loop’s operator is its own inverse (showing why is an exercise), and a permutation and its inverse have the same fixed points, so we only need to write down the operators in one sense (clockwise or counterclockwise).

Suppose for our example we try B, I, II, III and message key AAA at the start of the crib. These are the first sixteen permutations of the machine as it steps from to AAB then AAC then AAD etc.

M_1 = BAQMFEXIHSWPDYTLCVJOZRKGNU
M_2 = DJRALKWPOBFEYQIHNCXZVUGSMT
M_3 = ZLEJCUITGDMBKONSVXPHFQYRWA
M_4 = GCBLWTAKQZHDOSMXIUNFRYEPVJ
M_5 = OSNIRGFMDPVUHCAJWEBXLKQTZY
M_6 = WYMVNQZJLHOICEKUFTXRPDASBG
M_7 = CJAFKDZTPBEUORMIWNVHLSQYXG
M_8 = XIJUYSLVBCZGNMQWOTFRDHPAEK
M_9 = LFZRWBYTJISAOVMUXDKHPNEQGC
M_{10} = TKEZCQYNUWBPVHSLFXOAIMJRGD
M_{11} = KIWFNDUXBSAOTELQPVJMGRCHZY
M_{12} = SFKUTBPOXYCRNMHGWLAEDZQIJV
M_{13} = BAQWLKHGRSFEXTPOCIJNVUDMZY
M_{14} = TEOFBDZXQKJYRSCVIMNAWPUHLG
M_{15} = MHYPSWRBZXQVAONDKGEUTLFJCI
M_{16} = CPASNRHGKUIXZEVBYFDWJOTLQM

If we look at the S(A)-S(O) loop and calculate the permutation of its operator we get

$$M_{12}\, M_{15} \;=\; \texttt{NOJGAQLFVIWZSHMUCPTDERBYKX}$$

This permutation has no fixed points, so we cannot find values for S(A) and S(O). We have to reject this machine setting and try another.

Now we try B, I, II, III and message key AAB at the start of the crib. The first sixteen permutations of the stecker-less machine are

$$\begin{aligned}
M_1 &= \texttt{DJRALKWPOBFEYQIHNCXZVUGSMT}\\
M_2 &= \texttt{ZLEJCUITGDMBKONSVXPHFQYRWA}\\
M_3 &= \texttt{GCBLWTAKQZHDOSMXIUNFRYEPVJ}\\
M_4 &= \texttt{OSNIRGFMDPVUHCAJWEBXLKQTZY}\\
M_5 &= \texttt{WYMVNQZJLHOICEKUFTXRPDASBG}\\
M_6 &= \texttt{CJAFKDZTPBEUORMIWNVHLSQYXG}\\
M_7 &= \texttt{XIJUYSLVBCZGNMQWOTFRDHPAEK}\\
M_8 &= \texttt{LFZRWBYTJISAOVMUXDKHPNEQGC}\\
M_9 &= \texttt{TKEZCQYNUWBPVHSLFXOAIMJRGD}\\
M_{10} &= \texttt{KIWFNDUXBSAOTELQPVJMGRCHZY}\\
M_{11} &= \texttt{SFKUTBPOXYCRNMHGWLAEDZQIJV}\\
M_{12} &= \texttt{BAQWLKHGRSFEXTPOCIJNVUDMZY}\\
M_{13} &= \texttt{TEOFBDZXQKJYRSCVIMNAWPUHLG}\\
M_{14} &= \texttt{MHYPSWRBZXQVAONDKGEUTLFJCI}\\
M_{15} &= \texttt{CPASNRHGKUIXZEVBYFDWJOTLQM}\\
M_{16} &= \texttt{DLFATCJWYGVBNMZRXPUESKHQIO}
\end{aligned}$$

This time, when we look at the S(A)-S(O) loop we find

$$M_{12}\, M_{15} \;=\; \texttt{QOBJTIGHFVRMYLUAZKWDSPNECX}$$

which has fixed points at G and H. If this is the correct setting of the machine, then S(A) = G and S(O) = H, or S(A) = H and S(O) = G. However, if we look at the S(E)-S(N)-S(T) loop, we find

$$M_{13}\, M_1\, M_{14} \;=\; \texttt{LVRXHZOKANSWFQITDUYPGBJEMC}$$

which has no fixed points. So S(E) cannot be assigned a value. We must reject this machine setting.

This game continues. But what happens if we find all the fixed points that we need? Let's consider B, I, II, III and message key AAX. The permutations are

$$\begin{aligned}
M_1 &= \texttt{BAFKTCQNVZDRYHSXGLOEWIUPMJ}\\
M_2 &= \texttt{OCBXGUERJIZTPVAMYHWLFNSDQK}\\
M_3 &= \texttt{UEJOBTPZWCNSRKDGVMLFAQIYXH}\\
M_4 &= \texttt{BAQMFEXIHSWPDYTLCVJOZRKGNU}\\
M_5 &= \texttt{DJRALKWPOBFEYQIHNCXZVUGSMT}
\end{aligned}$$

M_6 = ZLEJCUITGDMBKONSVXPHFQYRWA
M_7 = GCBLWTAKQZHDOSMXIUNFRYEPVJ
M_8 = OSNIRGFMDPVUHCAJWEBXLKQTZY
M_9 = WYMVNQZJLHOICEKUFTXRPDASBG
M_{10} = CJAFKDZTPBEUORMIWNVHLSQYXG
M_{11} = XIJUYSLVBCZGNMQWOTFRDHPAEK
M_{12} = LFZRWBYTJISAOVMUXDKHPNEQGC
M_{13} = TKEZCQYNUWBPVHSLFXOAIMJRGD
M_{14} = KIWFNDUXBSAOTELQPVJMGRCHZY
M_{15} = SFKUTBPOXYCRNMHGWLAEDZQIJV
M_{16} = BAQWLKHGRSFEXTPOCIJNVUDMZY

All of the permutations in Table 179.1 have enough fixed points (go ahead and check). But we have a problem: If we calculate the fixed points for S(T) (for example), we get inconsistent answers.

$M_{14}\, M_{13}\, M_1$ = AMPIKNDXTFYHUELVZQJWSGBORC
$M_1\, M_{16}\, M_{11}\, M_{12}\, M_{10}$ = CBNLDHOWUZSKEIFGJQPVYMATXR
$M_1\, M_{16}\, M_{11}\, M_{15}\, M_{10}$ = MRDLOUQJTZVIWNPAXECFHYSGBK

The first has fixed point at A, the second at B, and the third at N. Since we must assign an unique value to S(T), we must reject this machine setting as inconsistent.

After trying the 5,905,536 combinations of reflector, rotors, and message key, far fewer remain. The more loops we have in the graph, and especially if they intersect, the fewer candidates remain. For each candidate, we need to try to work out the remaining plugboard settings and see what we get when we decrypt the entire ciphertext. Here, we will only go through the process for the correct settings of the machine, which are B, I, II, III, ABP. Now let's look at the permutations there and see what we can gleam about the plugboard wiring.

M_1 = SKPMUOYLXQBHDTFCJVANERZIGW
M_2 = HJLWMZNARBQCEGXTKIYPVUDOSF
M_3 = IZRUKPXNAWEOTHLFSCQMDYJGVB
M_4 = UYOMVQLRXPNGDKCJFHZWAETIBS
M_5 = NROYTGFWZXVQUACSLBPEMKHJDI
M_6 = TYQKNXRZOUDMLEIVCGWAJPSFBH
M_7 = HNEOCSJAPGMZKBDIXTFRVUYQWL
M_8 = EQFWACINGLRJZHTSBKPOXYDUVM
M_9 = OKIZVPTMCUBRHYAFWLXGJEQSND
M_{10} = RTHXZONCYUQVSGFWKAMBJLPDIE
M_{11} = XPWGHODEQYRSNMFBIKLVZTCAJU
M_{12} = POQWGIEVFNRUZJBACKYXLHDTSM
M_{13} = QPNGXYDSOVMRKCIBALHWZJTEFU
M_{14} = PLTVYKONMQFBIHGAJZUCSDXWER
M_{15} = KODCXJSRQFAUZYBWIHGVLTPENM
M_{16} = OVEXCNJPMGSZIFAHYUKWRBTDQL

The S(A)-S(O) mini-loop has fixed points at B, L, M, O, U, and Z. But if we go around the larger loops containing S(A), one gives fixed points at A and B, and the other has one at B. These three sets has one element in common: B. So S(A) = B, and the plugboard has a cable connecting A and B. Next, S(E) only participates in one loop, and it has fixed point C, so the plugboard connects E and C. For F, N, O, and T, we find that the only common fixed points show that the plugboard does not alter these letters. So far, we know this much about the machine at the start of the crib:

B, I, II, III, 1, 1, 1, AB, CE, ABP

If we decipher the corresponding segment of the plaintext, we get

TKBRJUURHTOONEOF

In the second position, Q should decrypt to H, but we have K. One way to fix it is to add a cable H-K, but we could do it with two cables, and there are many ways to choose them. So we skip it and move on to the third letter. There, we need I to decrypt to E, but we got B instead. We already have a cables A-B and C-E, so we can fix this with a cable that connects I to whatever M_3 maps to C. That letter is R, so we add a cable I-R. This affects the fourth letter. With the three cables that we know for certain, we have this decryption and see that the fourth letter has become I:

TKEIJUUIHTOONEOF

The fourth letter of the plaintext segment is H. We already have cable C-E and M_4 maps E-V, so we can complete the chain with a plugboard cable H-V. This is the only way to fix the fourth letter, because a chain from the ciphertext letter to the plaintext letter can only have two plugboard cables, and one has already been determined. Having the cable H-V allows us to revisit the second letter. With the four cables that we have already found, the K in the decryption remains unchanged. The plaintext letter is H, there is a cable H-V, M_2 maps V-U. and the ciphertext letter is Q. The only way to complete the chain is to add cable Q-U. For the fifth letter, the chain is R-I-Z-...-X. So we need cable X-Z. For the sixth letter, the chain is I-R-G-...-J, and we must add cable G-J. With the seven cables that we now know, the decryption is

THECRIBIVTOONEOF

We still need to fix the ninth letter. Its chain is N-N-Y-...-M (remember that N is not steckered), so we add the cable M-Y. We have managed to recover eight of the ten plugboard cables.

Running the machine backward to the beginning of the ciphertext tells us that the message key is ABC. What we know so far about the full key is

B, I, II, III, 1, 1, 1, AB, CE, IR, HV, QU, XZ, GJ, MY, ABC

Decrypting the ciphertext with it gives

```
MLYMXDONAYPUTTHECRIBINTOONEOFTHEWPADAWQUASYSINLOSWAPRLRESIQOYTXT
HENWHESVNTBACKFOTHELLEWSERFORTHECNNLDEHTODIGHTITUPSITIXZTLOMKELN
PTHINTSANTINGTHEMX
```

There are only six letters still unaccounted for that can be used for the remaining two plugboard cables: D, K, L, P, S, W, and we can apply the technique from Unit 178 to find the cables. The index of coincidence of the decryption is 1.415. If we try all possibilities for one more cable, we find the largest IoC for D-L at 1.5647. If we add that to the key and search again, the highest IoC is for S-W at 1.6187. Add it to the list, and the full key becomes

B, I, II, III, 1, 1, 1, AB, CE, IR, HV, QU, XZ, GJ, MY, DL, SW, ABC

The full plaintext is from *Candle and Crib* by Katherine Frances Purdon:

```
MRSMOLONEYPUTTHECRIBINTOONEOFTHESMALLSQUAREWINDOWSANDDREWITOUTXT
HENSHEWENTBACKTOTHEDRESSERFORTHECANDLESTOLIGHTITUPWITHXITLOOKEDN
OTHINGWANTINGTHEMX
```

Mrs. Moloney put the Crib into one of the small square windows and drew it out. Then she went back to the dresser for the candles to light it up with. It looked nothing wanting them.

vertex	loop operators
S(A)	$M_{12}\ M_{15}$ $M_{10}\ M_1\ M_{16}\ M_{11}\ M_{12}$ $M_{10}\ M_1\ M_{16}\ M_{11}\ M_{15}$
S(E)	$M_{13}\ M_1\ M_{14}$
S(F)	$M_{11}\ M_{12}\ M_{10}\ M_1\ M_{16}$ $M_{11}\ M_{15}\ M_{10}\ M_1\ M_{16}$
S(N)	$M_1\ M_{14}\ M_{13}$ $M_{16}\ M_{11}\ M_{12}\ M_{10}\ M_1$ $M_{16}\ M_{11}\ M_{15}\ M_{10}\ M_1$
S(O)	$M_{12}\ M_{15}$ $M_{12}\ M_{10}\ M_1\ M_{16}\ M_{11}$ $M_{15}\ M_{10}\ M_1\ M_{16}\ M_{11}$
S(T)	$M_{14}\ M_{13}\ M_1$ $M_1\ M_{16}\ M_{11}\ M_{12}\ M_{10}$ $M_1\ M_{16}\ M_{11}\ M_{15}\ M_{10}$

Table 179.1: Loop operators for the graph in the example. See Figure 179.1.

Figure 179.1: Graph ("menu") for the ciphertext and crib in the example.

Reading and references

Graham Ellsbury, “The Turing Bombe: What it was and how it worked,” 1998, http://www.ellsbury.com/bombe1.htm

Wikipedia, “Bombe,” https://en.wikipedia.org/wiki/Bombe

David Kahn, *Seizing the Enigma: The Race to Break the German U-Boat Codes, 1939-1943*, New York: Houghton Mifflin, 1991.

Programming tasks

1. While it is possible to automate the entire process, you may prefer to make the menu and look for loops by hand. If that is the case, then you will create a new program or script for each ciphertext/ crib pair. Finding the solution can take a long time, so we recommend that you use a fast programming language, such as C.

Exercises

1. Explain why the operator/permutation that we get by going around a loop is its own inverse.

2. Here are a ciphertext and crib. The text was encrypted with a three-rotor enigma with ring settings 1, 1, 1. The crib represents the first twenty letters of the plaintext. Decrypt the entire text.

   ```
   TBLIFFTJPMFZPJFTOWMTGKUFQKBBEIQGEHSYFELZOVPCTWPXKCUSJILGAMETZIAG
   JGVATTPPFBHLAQUAQCRLRONUZJUFYWWTFTQTJTQSXMAKFRBOGEXFQHJCOBFIDHAW
   QKODRHPGBUFEJBTETJZYGVAFPKZFKGJ

   IAMABOUTTOGIVEYOUTHE
   ```

   If you enjoyed this exercise, there is a similar challenge at MysteryTwister: https://mysterytwister.org/media/challenges/pdf/mtc3-hoerenberg-02-turing-en.pdf

Unit 180
SIGABA

SIGABA, also known as *Electric Cipher Machine Mark 2* (*ECM II*), *M-134-C, ASAM 1*, or *CSP-889*, is a rotor machine used by the United States during and after World War II. In this machine, the signal for a plaintext letter passes through five rotors, similar to those of the Enigma. But there is no reflector, and the stepping of the rotors is controlled by a complicated process. There are ten rotors from which to choose the *cipher wheels*; the other five serve as *control wheels*. In addition, there are five smaller, ten-contact rotors, called *index rotors*, which help in determining the irregular stepping of the cipher wheels. See Figure 180.2 for an overall functional view of the machine.

The ten large rotors are numbered 1 to 10 (or 11 to 20, or 21 to 30, etc.). In the codebooks, rotor 10 is referred to by “0”. Each of these has twenty-six contacts on each side and is labeled on the rim with the letters of the English alphabet, like those of the Enigma machine. In code books, they are referred to by the second digit only. The alphabetic permutations of these rotors are listed in Table 180.1. These rotors can be inserted into a slot in either forward or reverse (“R”) orientation, i.e., a rotor can be turned over so that the signal travels in the reverse direction through it. This does not invert its alphabetic permutation, but does an inversion and a reflection (B↔Z, C↔Y, etc.). We leave it as an exercise to work out the alphabetic permutations of the reversed rotors.

Figure 180.1: SIGABA. Photo by United States Air Force.

The small rotors are labeled on their rims by two-digit numbers. The first digit identifies the rotor, and the second is its current setting. These serve as index rotors to help scramble the signals that control the stepping of the cipher wheels. They are removable and replaceable in any order, but during WW II they were kept in numerical order (1, 2, 3, 4, 5). These rotors are not reversible and do not rotate while encryption or decryption is occurring. Their wirings are given in Table 180.2.

The overall function of the machine during encryption is as follows. The operator presses a letter or space key on the keyboard. The machine translates `Z` to `X` and space to `Z`. A signal on the wire designating the plaintext letter runs through the five cipher wheels from first to fifth. The signal emerges from the fifth wheel on the wire designating the ciphertext letter. Then four signals are sent into the first control wheel, on lines `F`, `G`, `H`, and `I`. They pass through the control wheels in reverse direction, some logic circuits whose actions are listed in Table 180.4, through the index rotors, then more logic (Table 180.5), and out to control the stepping of the cipher wheels. Any cipher wheel that receives a signal steps; if the rotor is inserted in the forward direction, then it steps to the alphabetically preceding letter; if it is inserted in the reverse direction, it steps to the following letter. After the cipher wheels have advanced, the center control wheel advances one step. If it passes from `O` to `N` in the forward orientation, or from `O` to `P` in the reverse orientation, then the control wheel in the fourth slot advances one step. If it, too, passes from `O` to an adjacent letter, then the control wheel in the second slot advances one step. The wheels in the first and last slots do not rotate.

The cipher is not self-reciprocal like the Enigma, so in order to decrypt, signals must pass through the cipher wheels in the reverse direction. Stepping, and the control of it, of course, is the same as in the encryption process.

The internal configuration of the SIGABA serves as its daily key. It includes

- the choices and ordering of five rotors as cipher wheels (five numbers, 0-9)
- the orientation (forward or reverse) of each of the five cipher wheels
- the choices and ordering of five rotors as cipher wheels (five numbers, 0-9)
- the orientation (forward or reverse) of each of the five control wheels
- the ordering of the five index wheels (five digits, 1-5)
- the settings of the five index wheels (five digits)

The lists of five cipher wheels and five control wheels must include all ten of the large rotors; none can appear more than once in the key. Here is an example of a daily key as it might appear in a codebook:

day	control	cipher	index
today	4 1R 9 0 3R	2 6R 5R 7 8	13 26 34 42 59

For each new message, the operator chooses five letters at random for the control-wheel settings and derives the settings of the cipher wheels from them (see below). These ten letters serve as the message key, although the operator only needs to send the control-wheel settings to the receiver.

Figure 180.2: Functional overview of SIGABA.

rotor	input ABCDEFGHIJKLMNOPQRSTUVWXYZ
1	INPXBWETGUYSAOCHVLDMQKZJFR
2	WNDRIOZPTAXHFJYQBMSVEKUCGL
3	TZGHOBKRVUXLQDMPNFWCJYEIAS
4	YWTAHRQJVLCEXUNGBIPZMSDFOK
5	QSLRBTEKOGAICFWYVMHJNXZUDP
6	CHJDQIGNBSAKVTUOXFWLEPRMZY
7	CDFAJXTIMNBEQHSUGRYLWZKVPO
8	XHFESZDNRBCGKQIJLTVMUOYAPW
9	EZJQXMOGYTCSFRIUPVNADLHWBK
[1]0	YCHLQSUGBDIXNZKERPVJTAWFOM

Table 180.1: Alphabetic permutations of the SIGABA cipher and control rotors, in forward orientation and at position A.

rotor	input 1	2	3	4	5	6	7	8	9	10
1	8	6	10	2	5	9	3	7	4	1
2	4	9	2	1	6	10	3	8	7	5
3	5	1	9	7	2	6	4	3	10	8
4	4	10	9	1	6	3	7	2	8	5
5	7	5	10	8	2	4	6	3	9	1

Table 180.2: Wirings of the SIGABA index rotors. Permutations listed are for rotors in position 0.

inputs	output
0 0	0
0 1	1
1 0	1
1 1	1

Table 180.3: The logical OR *operation. The output of* OR *is on (*1*) if either input is on, and off (*0*) if both inputs are off.*

input line to index wheels	logical expression
1	OFF
2	B
3	C
4	D OR E
5	F OR G OR H
6	I OR J OR K
7	L OR M OR N OR O
8	P OR Q OR R OR S OR T
9	U OR V OR W OR X OR Y OR Z
10	A

Table 180.4: Logic applied to input to the index wheels in CSP-889. The first column contains the number of the control line entering the index wheels. The second column is the logical expression applied to the output of the control wheels.

control line for stepping cipher wheels	logical expression
1	(1) OR (10)
2	(8) OR (9)
3	(6) OR (7)
4	(4) OR (5)
5	(2) OR (3)

Table 180.5: Logic applied to output from the index wheels. The first column contains the number of the control line determining the stepping of a cipher wheel. The second column is the logical expression applied to the outputs of the index wheels, where "(n)" indicates the signal on line n.

Some attention should be spent explaining how the cipher-wheel settings are derived from a five-letter message key. First, the operator sets all of the large rotors to the letter O. Start with the control wheel in slot 1 and advance it one letter (if in forward orientation, then one earlier letter alphabetically; if in reverse orientation, then one letter later alphabetically). Then send the signal on the four lines (F, G, H, I) through the control wheels (in reverse direction), logic circuits, and index rotors and allow them to advance the cipher wheels as if the machine were in normal operation enciphering a text. Continue with this procedure until the control wheel in slot 1 is set to the first letter of the message key. Note that if that letter is O, then it must take 26 steps and go all the way around. Now advance the control wheel in slot 2 until it shows the second letter of the message key, and advance the cipher

wheels after each step. Do the same for the remaining three wheels. Any time the target is O, the wheel must go all the way around. When the procedure is finished, the letters showing on the cipher wheels are the second half of the message key. Suppose we start with the example daily key:

day	control	cipher	index
today	4 1R 9 0 3R	2 6R 5R 7 8	13 26 34 42 59

Choose five random letters as the first half of the message key: WICSO. Remember that when one of them is O, that wheel must go all the way around. With the given internal settings, the procedure gives us PUJRC as the cipher-wheel settings.

Now we can attempt an example of encryption using the daily key above and message key WISCO. We already know that the initial cipher-wheel settings are PUJRC. Start with a plaintext:

THIS MESSAGE IS ENCRYPTED WITH SIGABA

The first step is to replace all Zs with X (there are none), then all spaces with Z:

THISZMESSAGEZISZENCRYPTEDZWITHZSIGABA

To encipher the first letter, let us find the alphabetic permutation for the cipher-wheel bank. We know what rotations are, from Table 20.1. The reflection that we need has this permutation:

$$X = (\mathtt{AZYXWVUTSRQPONMLKJIHGFEDCB})$$

This is b_{25} in Table 54.1, and is the same as $R_1 \circ z$ or $z \circ R_{25}$, where, as you recall, z is the reversal of the alphabet. A signal first enters the cipher-wheel bank into rotor 2 which is in forward orientation and set to P (=15). The effect of this rotor is $R_{15}^{-1} \circ W_2 \circ R_{15}$ (we have kept the notation of "W" for "wheel"). The signal then passes through rotor 6, which is in reverse orientation and set to U; its action is $R_{20} \circ X \circ W_6^{-1} \circ X \circ R_{20}^{-1}$ (notice where the inverses of rotations appear, and that X is its own inverse). Considering all five rotors in the cipher bank, the action of the entire bank is (where we have decided to drop the symbol for composition) (remember that things work from right to left in operator notation)

$$\begin{aligned} P_{\text{cipher}} &= (R_2^{-1}\, W_8\, R_2)\,(R_{17}^{-1}\, W_7\, R_{17})\,(R_9\, X\, W_5^{-1}\, X\, R_9^{-1})\,(R_{20}\, X\, W_6^{-1}\, X\, R_{20}^{-1})\,(R_{15}^{-1}\, W_2\, R_{15}) \\ &= R_{24}\, W_8\, R_{11}\, W_7\, R_1\, z\, W_5^{-1}\, z\, R_{11}\, z\, W_6^{-1}\, z\, R_{16}\, W_2\, R_{15} \\ &= (\mathtt{CETVLDQWOUZFYIHPMRJABNGKSX}) \end{aligned}$$

The first plaintext letter is therefore enciphered to A. We now have to find the stepping of the cipher wheels. To do so, we find the permutation for the control-wheel bank. We apply the same technique as above to find it when the wheels are set to WICSO.

$$\begin{aligned} P_{\text{control}} &= (R_{14}\, X\, W_3^{-1}\, X\, R_{14}^{-1})\,(R_{18}^{-1}\, W_0\, R_{18})\,(R_2^{-1}\, W_9\, R_2)\,(R_8\, X\, W_1^{-1}\, X\, R_8^{-1})\,(R_{22}^{-1}\, W_4\, R_{22}) \\ &= (\mathtt{CPKVFDZLIBOEQAJHSTXYGNWMRU}) \end{aligned}$$

Remember that the signals pass through the control bank in reverse direction, so we need the inverse of the permutation:

$$P_{\text{control}}^{-1} = (\texttt{NJAFLEUPIOCHXVKBMYQRZDWSTG})$$

The inputs on lines `F`, `G`, `H`, and `I` are mapped to `E`, `U`, `P`, and `I`. From Table 180.4, we see that the active inputs to the index-rotor bank are on lines 4, 6, 8, and 9. We now have to play the same game with the index-rotor bank. With our key, the overall permutation of the index rotors is

$$\begin{aligned} P_{\text{index}} &= (R_9^{-1}\, I_5\, R_9)\,(R_2^{-1}\, I_4\, R_2)\,(R_4^{-1}\, I_3\, R_4)\,(R_6^{-1}\, I_2\, R_6)\,(R_3^{-1}\, I_1\, R_3) \\ &= R_1\, I_5\, R_7\, I_4\, R_8\, I_3\, R_8\, I_2\, R_3\, I_1\, R_3 \\ &= (1\ \ 4\ \ 7\ \ 8\ \ 9\ \ 5\ \ 10\ \ 3\ \ 6\ \ 2) \end{aligned}$$

where now the rotations are of ten items and I_n represents the nth index rotor's permutation, from Table 180.2. We see that the inputs (4, 6, 8, 9) are mapped to outputs on lines 8, 5, 3, and 6. From the logic in Table 180.5, we find that the cipher wheels that step are in slots 2, 3, 4, and 5. Because the wheels in slots 2 and 3 are in reverse orientation, they advance to the next letter alphabetically; so `U` goes to `V` and `J` to `K`. The wheels in slots 4 and 5 are in forward orientation, so they advance to the previous letter alphabetically; `R` to `Q` and `C` to `B`. The cipher wheels are now set to `PVKQB` for the encipherment of the second letter. The center control wheel advances, so the control bank is now set to `WIBSO`. We can find the new permutation of the cipher-wheel bank:

$$\begin{aligned} P_{\text{cipher}} &= (R_1^{-1}\, W_8\, R_1)\,(R_{16}^{-1}\, W_7\, R_{16})\,(R_{10}\, X\, W_5^{-1}\, X\, R_{10}^{-1})\,(R_{21}\, X\, W_6^{-1}\, X\, R_{21}^{-1})\,(R_{15}^{-1}\, W_2\, R_{15}) \\ &= (\texttt{KCAIXJTBFLRMUPDHZYNWSGVEQO}) \end{aligned}$$

The second letter of the plaintext, `H`, is enciphered to `B`. If we continue in this manner, we find that the full ciphertext is

`ABOTTYSYZJAQWNETZPSCCUQMUFSPJRUJTDUEI`

A later version of the SIGABA machine, known as *CSP-2900*, is similar to the CSP-889, with these differences:

- The set of active lines into the control wheels is expanded to six: `D`, `E`, `F`, `G`, `H`, `I`.
- The logic applied to the output of the control wheels to form the input to the index wheels is modified to be what is listed in Table 180.6.
- The cipher wheels in slots 2 and 4 rotate in the opposite direction to those in slots 1, 3, and 5, which rotate as in CSP-889.

input line to index wheels	logical expression
1	U OR V
2	B
3	C
4	D OR E
5	F OR G OR H
6	I OR J OR K
7	L OR M OR N OR O
8	S OR T
9	W OR X OR Y OR Z
10	A

Table 180.6: Logic applied to input to the index wheels in CSP-2900. The first column contains the number of the control line entering the index wheels. The second column is the logical expression applied to the output of the control wheels.

Reading and references

Mark Stamp and Richard M. Low, *Applied Cryptanalysis: Breaking Ciphers in the Real World*, Hoboken: Wiley, 2007, section 2.4.

George Lasry, "Functional Description of SIGABA." This document can be found in the supplementary materials for some challenges on MysteryTwister, in this zip file: https://mysterytwister.org/media/challenges/add/mtc3-kopal-25-SIGABA-CSP889-01-add.zip

George Lasry, "A practical meet-in-the-middle attack on SIGABA," Proceedings of the 2nd International Conference on Historical Cryptology, HistoCrypt 2019, Linköping University Electronic Press, 2019, 41-49, https://ep.liu.se/ecp/158/005/ecp19158005.pdf

> Note that the above paper has an error in the table of index input logic. The following paper has the correct table.

George Lasry, "Cracking SIGABA in less than 24 hours on a consumer PC," *Cryptologia* 47:1 (2023) 1-37, DOI: 10.1080/01611194.2021.1989522

Timothy J. Mucklow, The SIGABA/ECM II Cipher Machine: "A Beautiful Idea," Center for Cryptologic History, [U.S.] National Security Agency, 2015, https://www.nsa.gov/portals/75/documents/about/cryptologic-heritage/historical-figures-publications/publications/technology/The_SIGABA_ECM_Cipher_Machine_A_Beautiful_Idea3.pdf

Operating Instructions for ECM Mark 2 and CCM Mark 1, transcribed at https://maritime.org/doc/crypto/ecm/index.php

Crypto-operating Instructions for ASAM 1, transcribed at https://maritime.org/tech/ecminst.php

Programming tasks

1. Create a function that finds the alphabetic permutation of a reversed rotor. Tabulate the reversals of the ten rotors in Table 180.1.

2. Implement enough of the machine in Python so that you can find the initial cipher-wheel settings when given a five-letter message key.

3. Add the remaining pieces to simulate the entire SIGABA CSP-889. Inputs should be the daily and message keys and, of course, the plaintext. Allow for the message key to have five or ten letters.

4. Now simulate the CSP-2900.

Exercises

1. Find the cipher-wheel settings for these five-letter message keys for the CSP-889 version of SIGABA, given today's daily key (see above in the text of this unit for the key).

   a. `SMFCJ`

   b. `OTCDP`

   c. `XGDRF`

2. Encipher this text with version CSP-889 and today's daily key and message key `MSRZN IVLDO`. Do not include punctuation, and remember to replace spaces with `Z` in the plaintext, if your program does not already do so.

   Science shall extend the bounds of knowledge and power, adding unimaginable strength to the hands of men, opening innumerable resources in the earth and revealing new secrets and harmonies in the skies.

   (from "The Glory of Peace" by Charles Sumner, in *Standard Selections*, edited by Robert I. Fulton, Edwin P. Trueblood, and Thomas C. Trueblood)

3. Decipher this text with version CSP-889 and today's daily key and message key `NRJCK`. Remember that `Z` in the plaintext is space.

```
QMNTJQSULAPPIQHXDDKDBVFALCMOAYVXGJKRYGHOGVJHKMLUCLEAHBBKNTFWBQNI
QVHILLUHFSEBSALAGJIJEZLSHWXSOKAMWSAVEMXQRGHXMRKATEUDUTHAYILMYJBG
JNYCFKOWVKRGZTKJHTOEYEDPNAVLPYCKBSZROQZNXNZZCMNPSRKDLAYORMERYYKE
YDFCYUHQJJSXIPJDYLXOJXYXTTSCADSRATQNVXUSUGBQIVIVCMNFQJAETNGXMVHT
TJVCUKAUXEMGWYESDFKGXSKKLCXLYHLQXQLGTBUAFKTJZTLQDKXHEHOZVRUMUSJD
MMMIXOABZXXQDPZNWONRILGWLDOMWZOKEPRJUFEROUCIGNQBGHGKBCZKEIKINDCU
CQZZYXICWOBKENPNXAONRJKHKXECLQCUGLOZMCIQOSJLXCLBGAIWPEBJFTZALQUZ
XUJPNYTKQPBKDUUGLITRBIJYMZWZERSNQXMIBEDYJFFVMRZJDWSJTYGJJMZBQAIZ
ZWMAJZCZTQCZHTVASCFEXXOMQLCJDQOCEPIGTLGAIXSSVOKWTWPDDDZHTMVYSXIP
KXMGGEQDBOLZHBEYNFDEHGLPLQLNHREYREOFJTZQTYZOCHMNPBQFF
```

4. Decipher with version CSP-2900, the following daily key, and message key `ZIHMF`.

day	control	cipher	index
tomorrow	9R 8R 2 5 3R	0R 7 4R 1 6	29 30 18 58 44

```
RCWWMQXIVNASRUDGPRFKXLOYMUFUKDHHWEZBDMDVKARBUNPBBMEIMPEZPVUTILMO
NIBVQSAJPQPJHUKUYSDXXUVBFRULWVDCADCRDXFJOYHTZLKRQSNSJGBCVBRUEWKV
RXWEQHBDLFCEJQJSIEAUQCWRUZNRFOVMIUVFETDSSRLHCDAHUIFDMYVHUPPEUGHH
ERDCITZSSVRQOSHAMPENKJKDAHZRLXHTCYGCHONEFTQMDNQGEYGHKQJEHKXWSCWQ
ELMBVPPPWLEFCWRSEHOEZGXFBYRQWIDEDONXLMECOTYFHEMNWLAWSDDGSKEJKSUR
IYDPJKLXKUKBCFIDSRVJJYVFDMEQDYDMZGZJVAEJVRSOTLODMQYKPSYHKOPPVNQI
KUSRTKOPSLOMWKCREVIIJMGOQPZOBTJFKSBFZMUKSOUGFMODAWRREWANHPPSMRWY
YAHRAUVSGOBABIHTYVSWFVIMCEYEDWVQRURYQZBIQVZKPIOINSMKABDDHFNZBVTA
NVSDZYOILCBMDRIELW
```

Unit 181
Фиалка

Fialka, also known as *M-125*, or *Фиалка* in Russian, is a cipher machine used by the Soviet Union during the Cold War. It had ten rotors, irregular stepping, a reflector with special properties, and a punchcard reader for implementing an alphabetic permutation on the way into and out of the rotor assembly. Unlike Enigma and SIGABA, the Fialka's rotors permute the Russian alphabet, or at least most of it.

The Russian alphabet (a version of Cyrillic) is

АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ

Take note that Ы looks like two, but is a single letter. Russian cryptographers typically treat Е and Ё as the same letter; some Russian civilians do as well. The full alphabet has thirty-three letters, but the rotors and punchcard of Fialka are designed to handle only thirty. This is the modified alphabet used by the machine:

АБВГДЕЖЗИКЛМНОПРСТУФХЦЧШЩЫЬЮЯЙ

The letter Й ("short i" or "и краткое") has been moved to the end, probably so that the ten rotors of the machine could be labeled А-К without confusion. Letter Ё is understandably missing, but so are Ъ and Э. After one of the many Russian revolutions, there was an orthographic revision of the language, and Ъ ("hard sign" or "твёрдый знак" or "turdy snack" as one student may have called it) was dropped at the ends of words. It is now a rather rare letter, and I cannot even think of a word that contains it right now. Letter Э is also rare, and almost always appears in words derived from foreign words; a suitable replacement is Е.

Versions of the Fialka machine were used in different regions of the U.S.S.R. and countries in the Warsaw Pact, and we don't have complete information about every variation. They differed in their rotor wirings and in how keys on their keyboards could be used for Latin letters and symbols. To distinguish a set of rotors from another, a designation such as "3K" is stamped on them, along with the letter denoting which rotor we have from the set (А through К). In Tables 181.2, 181.3, 181.4, and 181.5 are the wirings for series 68, used in Russia; 3K, used in Poland; 5K, used in Hungary, and 6K, used in Czechoslovakia. For variants of keyboard mappings to Latin letters and symbols in Poland, East Germany, and Czechoslovakia, see Table 181.1. At the time that this book is being written, there is no information about other versions of rotors or keyboard layouts.

Figure 181.1: Fialka cipher machine. In the photo on the right, the cover is open and the rotors are visible. Photos from Tom Perera (EnigmaMuseum.com).

region	character set	Cyrillic key АБВГДЕЖЗИКЛМНОПРСТУФХЦЧШЩЫЬЮЯЙ
Poland	Latin: symbol:	FADULT5PBRKVZJGHCNE7QWXIOSM8Y2 Ą%-6)4 9?3(=5+ĘŁ:,2 01/78Ž.°'
East Germany	Latin: symbol:	F8DULT5PBRKVZ4GHCNEA327IOSM96J V:W6)4 9/3(X5JÄÜY,2 01F78Q.-Ö*
Czechoslovakia	Latin: symbol:	FADULT5PBRKVZJGHCNE7QWXIOSM8Y2 ŘÄČ6Ě4 9"3Ö=5ÜŽ+:,2 01/78Š.%-

Table 181.1: Substitutions for Latin letters and symbols used by Fialka. Ж and Ф serve as symbol shift and letter shift, respectively. Non-Cyrillic letters were laid differently onto the keyboards in different regions of the U.S.S.R. In East Germany, the usual Ä=AE, Ö=OE, Ü=UE, *and* ß=SS *or* SZ *can also be used.*

The Fialka cipher machine is very much like what an Enigma might have become if it had evolved under the control of the workers who have seized control of the means of production in order to lead the world into a brighter future. It has a keyboard, rotors, and reflector, but instead of the plugboard it has a punchcard reader for greater efficiency. The number of rotors has been increased to ten, and they have thirty contacts on each side. They are installed in any order into the machine beginning at the left end near the reflector (slot 1) and continuing to the end near the entry plate (slot 10). The reflector has been modified to avoid the weakness of Enigma that letters could not be enciphered to themselves. And the punchcard permutes the entire set of inputs, unlike Enigma's limit of the use of ten transpositions. We will now spend some time looking more closely at these components before we put the pieces together and see how a simulation might work.

The keyboard has thirty buttons for letters, of course, for the thirty letters listed above. But it also has a space bar, and it maps space to Й. Unfortunately, this means that yet another letter is unavailable. When Й is encountered in a plaintext, the output printer writes a space. The keyboard for machines used in various regions of the Soviet world had different substitutes for Latin letters. Those that are known are listed in Table 181.5. Between the keyboard and the punchcard reader the wiring of the machine carries a permutation which we call *S*:

$$S = (\text{СЩЙОЫХЕУАПЯФГЮШБЦЧТМЖДЬЗКИРНЛВ}) \tag{181.1}$$

The punchcard reader comes next. It holds a paper card with a 30×30 grid that unfortunately is not printed. Nevertheless, there can be exactly one hole punched in each row and exactly one in each column of this grid. Such a thing can be thought of like a 30×30 matrix with a single 1 in each row and column and 0 everywhere else (see Unit 123 for our introduction to matrices). Such a matrix, when multiplied by a column vector, performs a permutation on it. The punchcard reader takes the matrix defined by the holes in the card and applies the permutation to the thirty data lines coming to it from the keyboard (after passing through *S*). In Figure 181.2 is a reasonable facsimile of a Fialka punchcard. To read its permutation, look at the perforation (row of holes) near the top. If you discard the leftmost and rightmost holes of the perforation, then the remaining thirty define the columns of the grid. As you read from left to right, the letter that goes into writing down the permutation is which row is punched. In Figure 181.2, the first column has the fifth highest hole, so must be Д. The second column has the twenty-fourth highest hole, so must be Ш, etc. The permutation on this card is

$$P = (\text{ДШНЯЖЧВЙИБАЮХПЩТКГУСЫМЬЗФЕРОЛЦ})$$

As the signal leaves the punchcard reader and heads toward the rotors, it encounters another permutation built into the machine's wiring which we call *T*:

$$T = (\text{ЮОФШБРАКХЛСНУЙДЕЗПЧЩЬТВЯЮМЦЖИГ}) \tag{181.2}$$

Figure 181.2: Facsimile of a punchcard for use in Fialka. The number indicates the day of the year in which it should be used. The arrows indicate the holes in the perforation through which alignment pins will protrude (pins are not placed symmetrically so that the card would not be placed in the machine upside-down). The permutation that this card represents has been randomly generated and should not be used in real-world communication.

Figure 181.3: Disassembled configurable rotor from a Fialka machine. The red mark on the rim at the top is the indicator mark for the ring setting. The wiring core is A *and side 1 is showing. The white mark at the top of the wiring core is the indicator for its placement in the rotor frame. Photo from Tom Perera (EnigmaMuseum.com).*

Rotors have thirty contacts on each side, and thirty Russian letters on the rims. They come in two varieties: adjustable/configurable and non-adjustable/non-configurable. Non-configurable rotors cannot be changed. They can be installed into different slots in the machine and rotated to different positions (i.e. so that they display a different letter to the operator), but they cannot be installed in reverse orientation and their alphabet ring cannot be shifted, etc. The permutations in Tables 181.1, 181.2, 181.3, and 181.4 are for the non-adjustable rotors in series 68, 3K, 5K, and 6K. Each series has ten rotors, designated with one of the Russian letters А, Б, В, Г, Д, Е, Ж, З, И, К. Each has thirty contacts on each side, so effects a permutation of the subset of the Russian alphabet with thirty letters given earlier. The permutations in the tables are for a signal entering on the righ-hand side of a rotor and exiting on the left-hand side. The permutation when the signal returns and passes left to right is the inverse. For dealing with the rotations of rotors with thirty positions, we need a new set of rotation permutations. The first is

$$R_1 = (\text{БВГДЕЖЗИКЛМНОПРСТУФХЦЧШЩЫЬЮЯЙА})$$

The rest can be generated from it. As we have seen before, the permutation of a rotor W that has been turned by n steps is its unrotated permutation W sandwiched between two rotations:

$$R_{-n}\, W\, R_n \tag{181.3}$$

The configurable rotors have three components: a rotor frame, an alphabet ring, and a wiring-maze core. The frame has the thirty letters on its side, so that the alphabet ring and wiring core can each be aligned to one of those letters. In addition, the core can be flipped over so that one of two sides can be facing outward/leftward (sides “1” and “2”) (the frame cannot be installed into the machine in reverse orientation). Recall that in Unit 180 we saw that installing a rotor W into SIGABA in reverse orientation gave us $X\, W^{-1}\, X$, where X is a reflection about the axis that runs through letters A and N. With Fialka, when we place a wiring core into a rotor frame so that side 2 is facing outward, we do something similar, and we need the reflection about the axis that passes through (Russian letters) А and Р. So our new X is

$$X = (\text{АЙЯЮЬЫЩШЧЦХФУТСРПОНМЛКИЗЖЕДГВБ}) \tag{181.4}$$

Things are a little more complicated now, since the core can be rotated when placed in the frame as well as flipped over. Suppose we take a rotor in its base configuration $W_{\text{base}} = W(0, 0, 1)$, which is the same as the non-adjustable version, and which has the ring setting at А=0, the wiring-core indicator mark aligned to А=0, and the core with side 1 facing outward. The effective new rotor with ring setting r and core alignment m is

$$W(r, m, 1) = R_r^{-1}\, R_m\, W(0, 0, 1)\, R_m^{-1}\, R_r \tag{181.5}$$

When the core is flipped, the new rotor is

$$W(r, m, 2) = R_r^{-1}\, R_m\, X\, W(0, 0, 1)^{-1}\, X\, R_m^{-1}\, R_r \tag{181.6}$$

rotor	input АБВГДЕЖЗИКЛМНОПРСТУФХЦЧШЩЫЬЮЯЙ
А	ГИЖЬУЦШЙВДБТЛАФЧЫОЯЮКПЕХЩРЗНСМ
Б	ПЧЯДЗТЙКЮОЫЛХГФНЩМШВБСИЦАУЖРЕЬ
В	ДЩОГЯАЬНЦРЖЛФПЙБУЫСЧВКШЗМТЕИЮХ
Г	ФЧТХНПМАУЫСЙЖЩОКЯЗИГЦДЬРБЕЛЮШВ
Д	ДФЗКГОЧЬЙШИНЕЫМАЛЩБЖСВРТЮХЯПУЦ
Е	ЬТБДРКУЯЮМИГЫХЩАЛЕЖФСПНЙЦШОЧЗВ
Ж	ЫЙЯВРЕОЛКШБДЦУМЩХФЬЗНГИЮАПЖТЧС
З	ЗЫРМШЖЦКХЬПИГОТЩДНЯЕБФЛСАЧУВЮЙ
И	ТСДМБЮЫУРЧФВЯШПКЦХЖИЗГЬОАЩЛНЙЕ
К	КЗШЬБДЖЛГЕМЙЧТЫПЯНВЩЮРАЦОИХФУС

Table 181.2: Rotor wirings of Fialka for rotors in series 68 used in Russia. These wirings are for non-adjustable rotors and for a signal entering from the left. For the adjustable rotors, these wirings match the configuration in which the wiring maze and rotor frame have the same letter designation, А *on the ring is aligned with* А *on the rotor frame, the indicator mark on the wiring maze is aligned with* А *and side* 1 *faces outward.*

rotor	input АБВГДЕЖЗИКЛМНОПРСТУФХЦЧШЩЫЬЮЯЙ
А	ЧЦВЖГЗРЕКФПСШИЙМЩЛАЮЬДЯЫБТХОНУ
Б	ВШФБЕХЫЖТГСЧПУКЙНЮЯЛИЩАОЦЗЬДМР
В	ФДЖПХЬГАЦСЧНЙЕЫКРОУТЯШВМИЛБЮЗЩ
Г	РХЮЛЬВПМШЙИСГФЩЗАЯУТОКДЧЫЖЕЦБН
Д	ТПАЦУРЯЗСГВОЕЙЧДЫНЩКМХЬФЖЛШИБЮ
Е	ИОНФШЗБЕДУЛЮЙВТПЖЩРАМЧЬЯСКХГЦЫ
Ж	ЖИДЫЕГУВЗЮЦМХШЧКНАРЯБЩЬПТЛОСЙФ
З	ЯЬПНЗБЩМЕЧИТШАОХСКВЛЦЖРГУЫДЙЮФ
И	ДУБЬФЫЖЛРТВНГЧЮХЕШЯЙПСИМЗЦЩКАО
К	ФШЗЩУАСДПЬИМЦКТВРЙГОЖЧЛБЯЫЮХЕН

Table 181.3: Rotor wirings of Fialka for rotors in series 3K used in Poland. These wirings are for non-adjustable rotors and for a signal entering from the left. For the adjustable rotors, these wirings match the configuration in which the wiring maze and rotor frame have the same letter designation, А *on the ring is aligned with* А *on the rotor frame, the indicator mark on the wiring maze is aligned with* А *and side* 1 *faces outward.*

rotor	input АБВГДЕЖЗИКЛМНОПРСТУФХЦЧШЩЫЬЮЯЙ
А	НЛКПАФШБГИЬСЫВМЗДРУЦЧЮОЩТЕЙЯЖХ
Б	ЕУЦАХНЮЫЙКИЖМЧСШЩЗФРГБПДТЯВЬЛО
В	ЩПЮЖУТОКШЕРИГФВЬЦДЗЯАМХЧСЫНБЙЛ
Г	ЧВШНГЫТПЛЕЯЦЩЖИХФБОСУЙЗКМАРЮДЬ
Д	РЦИБЕДАМГСОХТПЮЬЗЛЯВЩКУЫЧНШФЖЙ
Е	КЕЗФЧЬЫГЯТСХШЛОЮАДНМЦВРУЖИЩЙБП
Ж	ХЙЫВРЮФИЩАЖСКЕЬЧМБЦШОЛЗГУДНПЯТ
З	ШСЛФЙЯУКЖХЕТБАДНЬМПЩГЧОЮИВЫРЗЦ
И	СИБГФЬРКХЖПЯЫЩТОЛЙЦМЮЧНЗАЕДВШУ
К	БМДЗНЯКУЩШРЖЦЬЫЮЙЕГИЛТСЧАВОФПХ

Table 181.4: Rotor wirings of Fialka for rotors in series 5K used in Hungary. These wirings are for non-adjustable rotors and for a signal entering from the left. For the adjustable rotors, these wirings match the configuration in which the wiring maze and rotor frame have the same letter designation, А *on the ring is aligned with* А *on the rotor frame, the indicator mark on the wiring maze is aligned with* А *and side* 1 *faces outward.*

rotor	input АБВГДЕЖЗИКЛМНОПРСТУФХЦЧШЩЫЬЮЯЙ
А	НЦЗТФМЮГПЬВДРОЧЫАЩСЛЙКШЖЕХЯБИУ
Б	ФЗДПГЮХАШНЯМОЧЩЖИЙЬВЛТСУЦКБЫЕР
В	ЯЛГЦШРТБЧВСЗФДЮМПЫЙЖХУНКЬЩИАОЕ
Г	ГМУЯШЧЖЙПАФОТБРЬКЩСЮЕХЛЗЦДИВЫН
Д	ТБПЖФЮЗНЧМУЬГШКОЛЕЙВСЫЦАЯЩРХДИ
Е	РГОШЧУЙВАЗЬНИДЯКПЫЦЖЩСФЛБЕХЮТМ
Ж	ЫЧЖДНЗШЙЯФЦИМКЩРВХУТГАЮЬЕБПСЛО
З	РЦОЙШПСФГЖЬМЕНЩХАДЫЗЛЧЯЮВТКУБИ
И	МАСЯЕГЖЛПВХЩИЫЙНЦФКШЬОЮЧБДУТРЗ
К	ИХЗРЩДФЦГЬОУАПЙБКСЖШМТЯВЧЕНЮЫЛ

Table 181.5: Rotor wirings of Fialka for rotors in series 6K used in Czechoslovakia. These wirings are for non-adjustable rotors and for a signal entering from the left. For the adjustable rotors, these wirings match the configuration in which the wiring maze and rotor frame have the same letter designation, А *on the ring is aligned with* А *on the rotor frame, the indicator mark on the wiring maze is aligned with* А *and side* 1 *faces outward.*

Stepping of the rotors occurs after each letter is enciphered and is irregular. Rotors in slots 1, 3, 5, 7, and 9 (counting from left to right) are ganged together, and their motions are interrelated. They only rotate in the forward direction, i.e., so that the letter describing its positions changes to the following one in the thirty-letter subset of the alphabet (with wrap-around). The rotor in slot 9 steps every time. The rotors in slots 1, 3, 5, and 7 cannot rotate if an advancement-blocking pin is present on the rotor in slot 9 at a position 20 further than its current position. If the pin is not present, then the rotor in slot 7 steps. The rotors in slots 1, 3, and 5 cannot rotate if a pin is present on the rotor in slot 7 at a position 20 further than its current position. Without that pin, the rotor in slot 5 steps. The same behavior continues, so we see that a rotor in an odd-numbered slot cannot advance if it is blocked by a pin on a rotor in any other odd-numbered slot to its right. Rotors in slots 2, 4, 6, 8, and 10 work together and rotate in the reverse direction, i.e., so that the letter indicating its current position changes to the one preceding it in the alphabet. The rotor in slot 2 is the one that steps every time. Each of the others is blocked from stepping if there is a pin in a position 17 further than the current position of any rotor in an even-numbered slot to its left. The positions of the stepping-inhibiting pins for series 68, 3K, 5K, and 6K are in Tables 181.6, 181.7, 181,8, and 181.9.

rotor	pin locations АБВГДЕЖЗИКЛМНОПРСТУФХЦЧШЩЫЬЮЯЙ
А	×× × × ×× ××× ×× ××× ×××
Б	× ×××× ×× ××××× ×××× ××××× ××
В	× × × × × × ××× ××
Г	× ×××× ×× ××××× × ××× × ××
Д	×× ×× × × ×
Е	× × ×××× × ××××× ×× ×××
Ж	×××× × × ×× ×× ×× ×
З	× × × × × ×× ×
И	× ×××× × × ×× ××
К	×× ××× × ×× × × ×

Table 181.6: Locations of stepping-inhibiting pins on the rotors in series 68.

rotor	pin locations АБВГДЕЖЗИКЛМНОПРСТУФХЦЧШЩЫЬЮЯЙ
А	× × ×× × × ×× ×× × × ×
Б	× × × × × ×××××× ×× × ×××
В	× × × × × × ×
Г	× ××××× ×× × ×××× ×××××× ××× ×
Д	× × ×× × × ×
Е	× × × ×××× × ×××× ×
Ж	× ×××××× ×××××× × ×××× ×××× ×
З	× ××× × × × ×
И	×××× × × ×× × × ×××××× ×
К	× × × ××× ×× ×× ×××

Table 181.7: Locations of stepping-inhibiting pins on the rotors in series 3K.

rotor	pin locations АБВГДЕЖЗИКЛМНОПРСТУФХЦЧШЩЫЬЮЯЙ
А	×××× × ×× ×× × ×
Б	× × × × × × ×
В	× × × × × ××
Г	× × × × × × ×
Д	× × × × ××× × × × ×
Е	×××× × × × × × ××
Ж	× ×× ××× × × ×× × ××
З	×× × × ×××× ×× ×× × ×××× ××
И	×× ×× ×××× × × ×××× ×××
К	×× × ×××××× ××× ××× ×× ××

Table 181.8: Locations of stepping-inhibiting pins on the rotors in series 5K.

rotor	pin locations АБВГДЕЖЗИКЛМНОПРСТУФХЦЧШЩЫЬЮЯЙ
А	××× × × × ×× ×× ×× ×××× × ××
Б	× × × × × × × ×× × ×
В	× × × × × × ×
Г	× × ××× ××××× ×××× × ××× ×
Д	× × × × × ×
Е	× ××× ××× ××××× ××× × ×
Ж	××× ××× ××× × ×××××× ×
З	×× ×× × × ×××××
И	× ×× × × × × ×
К	××× ×× ××× ××××× ××× ××× ×× ××

Table 181.9: Locations of stepping-inhibiting pins on the rotors in series 6K.

Unlike Enigma, Fialka has only one reflector. It is permanently installed in the machine; it cannot be removed; it cannot be rotated. And to improve on the Enigma, it has some innovations. While most of the connections are wired in pairs, like in any other self-respecting reflector, a signal on connector 12 (counting from zero) indicates that a plaintext letter should be enciphered to itself. This eliminates the weakness that Enigma had in its inability to do this. In Fialka, since one connector has this property, any given plaintext letter has a one in thirty probabililty of being its own ciphertext letter; this is just what would be expected from a process that randomly assigns ciphertext letters. Since it is impossible to send a signal back again on the same connector, some circuitry in the reflector instead sends a signal back to the keyboard to tell it to bypass the entire encryption process and send the plaintext letter to the output. Nevertheless, when we are creating a simulation of the Fialka, we can ignore this physical impossibility and simply allow the reflector's permutation to map Н to Н. But now we have another problem: with one connection used for itself, there is an odd number remaining. We cannot map another connection to itself, because that would spoil the appearance of randomness. To solve this problem, the makers of the Fialka introduced another bit of circuitry that performs a 3-cycle on connections 15, 17, and 23. This breaks the self-reciprocity of the reflector, so some more circuitry must be added to reverse this 3-cycle when the machine is in decryption mode. Here is the permutation of the reflector during encryption. We call it *U* as we did for Enigma. Above it we write the rotor alphabet so that we can see the 3-cycle more easily.

$$\begin{array}{l} \quad\;\; \text{АБВГДЕЖЗИКЛМНОПРСТУФХЦЧШЩЫЬЮЯЙ} \\ U \;=\; (\,\text{ЧЕФЮОБМСЦЛКЖНДЯТЗШЬВЩИАРХЙУГПЫ}\,) \end{array} \tag{181.7}$$

Notice the 1-cycle (Н) and the 3-cycle (РТШ). All other letters are in 2-cycles.

We are finally ready to put all of the pieces together. In Figure 181.4 is an overview of the Fialka cipher machine. When a letter is processed, these things happen:

1. A key for the letter is pressed on the keyboard. If the key is the space bar and the machine is in encryption mode, then the character is changed to Й by the keyboard. The signal leaves the keyboard.
2. The signal passes through permutation S (Equation 181.1).
3. The signal passes through permutation P of the punchcard reader.
4. The signal passes through permutation T (Equation 181.2).
5. At the entry plate to the rotor assembly, there is a rotation by 3 because the letters indicating the positions of the rotors are not centered in the machine.
6. The signal passes through the ten rotors in reverse order (right to left, slot 10 to slot 1). Despite the reversal of rotor order, the permutation of each is in its forward direction.
7. Between the rotor assembly and the reflector is a rotation by −3. See number (5) above.
8. The signal passes through permutation U (Equation 181.7) if the machine is in encryption mode. If the machine is in decryption mode, the permutation is U^{-1}.
9. A rotation by 3 when entering the rotor assembly.
10. The signal passes through the ten rotors in forward order (left to right, slot 1 to slot 10). The permutation of each is the inverse of the permutation in step (6).
11. Rotation by −3 at the entry plate.
12. Permutation T^{-1}.
13. Permutation P^{-1}.
14. Permutation S^{-1}.
15. If the machine is in decryption mode and the resulting signal is Й, then the printer outputs a space. Otherwise, it prints the letter that it receives.

The overall permutation effected by the machine during encryption is (ignoring the bits about space bar and Й)

$$F = S^{-1}\, P^{-1}\, T^{-1}\, R_{27} \prod W_i^{-1}\, R_3\, U\, R_{27} \prod W_i\, R_3\, T\, P\, S \tag{181.8}$$

Remember that operators act on what comes to them from the right and so this product of operators works from right to left. We have used $\prod W_i$ and $\prod W_i^{-1}$ as shorthand for

$$\begin{aligned} \prod \mathrm{W}_i &= \mathrm{W}_1\, W_2\, W_3\, W_4\, W_5\, W_6\, W_7\, W_8\, W_9\, W_{10} \\ \prod W_i^{-1} &= W_{10}^{-1}\, W_9^{-1}\, W_8^{-1}\, W_7^{-1}\, W_6^{-1}\, W_5^{-1}\, W_4^{-1}\, W_3^{-1}\, W_2^{-1}\, W_1^{-1} \end{aligned}$$

where W_i is the permutation of whatever rotor is in slot i in whatever position it currently is in. A configurable rotor is constructed as in Equations 181.5 and 181.6. All rotors are rotated to their current positions according to Equation 181.3.

Figure 181.4: Functional overview of the Fialka cipher machine. Input from the keyboard (30 wires) passes through permutation S, the punchcard reader P, permutation T, a rotation for the entry plate, ten wheels, a rotation between the wheels and reflector, the reflector U, and back again in reverse to the output.

The internal settings of the Fialka machine form the daily key. For machines with rotors with fixed wiring, the daily key is a list of ten rotors in the order in which they are installed on the spindle, from left to right (from the reflector toward the punchcard). For machines equipped with configurable rotors, the specific contents of the daily key are five lists of ten items and the punchcard. The first item in each list describes what is placed in the first slot, etc. The lists describe

- the choice of rotor frame (Russian letter)
- the ring setting (Russian letter)
- the identity of the wiring core (Russian letter)
- whether the wiring core is installed upside-up or upside-down (digit 1 or 2)
- the position of the indexing mark on the wiring core (Russian letter)

The initial positions of the rotors serve as the message key.

For example, if we are working with a machine with non-configurable rotors from series 3K, and our key is this one, which was randomly generated:

daily key: `ЕЗКАВ ИГЖДБ`

message key: `ВЫДЮЛ ЗХСКЧ`

The permutation given by the punchcard is

P = (ЗКЬГЦПЮДЫШВТОЯЛФЕБЖЩЙИМСЧАХНРУ)

Here is a typical message that we can encrypt:

THIS MESSAGE IS ENCRYPTED WITH FIALKA

The first thing to do is rewrite the message in Russian, with Й in place of spaces.

ЕРШЫЙЬУЫЫБПУЙШЫЙУТСКЯЗЕУВЙЦШЕРЙАШБДЛБ

The machine is loaded with non-configurable rotors such that rotor Е is in the first slot, rotor З in the second, К in the third, etc. The rotors are initial set to the positions given by the message key: the first at В, the second at Ы, etc. The resulting permutation from Equation 181.8 is

УЛНЮЫЙПФСЩБЧВХЖШИТАЗОРМЦКДЯГЬЕ

This permutation maps the plaintext letter Е to Й. Next, since there is a steeping-inhibiting pin at Н on rotor З in slot 2 (17 steps ahead of its current position, Ы) (Table 181.7), the rotors in slots 4, 6, 8, and 10 do not step at this time. There are no pins blocking the rotors in slots 7 and 5 from stepping, but those in slots 3 and 1 cannot step. So the rotor positions go from ВЫДЮЛЗХСКЧ to ВЩДЮМЗЦСЛЧ and we are ready for the second letter of the text. We continue until we have the full ciphertext:

ЙЩНШВЯХГЬЛЮЫЧЮРФУЧЙМКЗЛЕНШЕРЙГЦЫЮЗМДБ

Now let's expand the daily key for a machine with configurable rotors. The new settings were also randomly generated.

daily key: ЕЗКАВ ИГЖДБ
ШЖВЙМ ЛПХЦЕ
БЗГКИ АДЖВЕ
21121 11221
ВЧЩЫР ЯЛПБИ

message key: ВЫДЮЛ ЗХСКЧ

The first slot contains a rotor comprised of the Е frame with the Б wiring core installed with side 2 facing outward, ring setting Ш=23, and core alignment В=2. The other slots contain rotors configured according to the respective columns in the key. The permutation on the punchcard is the same as before. With this key, the same plaintext becomes this ciphertext:

МТЧЬГДХМАБФПЬЫОСФЕГЯУЮТЩОРШЦЧЛРЧЛПГИТ

Reading and references

Thomas B. Perera and David Hamer, “General Introduction: Russian Cold War Era M-125 and M-125-3MN Fialka Cipher Machines,” 2005, http://w1tp.com/enigma/mfialka.htm

Thomas B. Perera and David Hamer, “Rotor Technical Descriptions and Wiring and Stepping Data for the M-125-MN and M-125-3MN/-3MP3 Russian Fialka Cipher Machines,” 2005, http://w1tp.com/enigma/mfr.htm

Paul Reuvers and Marc Simons, Fialka M-125: Detailed description of the Russian Fialka cipher machines, version 2.0, 2009-06-22, https://www.cryptomuseum.com/pub/files/Fialka_200.pdf

If you see “36” in the table for the rotor wirings of series 5K, it should be “26”.

Crypto Museum pages on the Fialka:

main page:	https://www.cryptomuseum.com/crypto/fialka/index.htm
functional overview:	https://www.cryptomuseum.com/crypto/fialka/block.htm
rotors:	https://www.cryptomuseum.com/crypto/fialka/wheels.htm
rotor wirings:	https://www.cryptomuseum.com/crypto/fialka/wiring.htm
reflector:	https://www.cryptomuseum.com/crypto/fialka/magic.htm
punchcards:	https://www.cryptomuseum.com/crypto/fialka/card.htm

Python tips

Python version 3 does everything in eight-bit Unicode by default, so there is nothing special that you need to do (in Python 2 you needed to set the encoding of your files in a special comment). Printing to console may be a problem if you don’t have that set up right.

The Russian letters are at these Unicode points (in hexadecimal):

А-Я	U+0410 through U+042F
а-я	U+0430 through U+044F
Ё	U+0401
ё	U+0451

Of course, it may be easier to just open this document in software that allows you to copy and paste this alphabet:

АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ

Remember that Ы is one letter and that often, Е and Ё are treated as the same letter.

The string methods `.upper()` and `.lower()` work on Russian, too.

Programming tasks

1. Simulate the Fialka in Python. This can be a good exercise in using a non-English alphabet and creating files in Unicode.

2. If you are ambitious, try installing the `Pillow` package in Python and use it to create a punchcard reader that takes a graphic file and outputs the permutation. Specifications for the physical size and positions of the holes can be found in the reference by Reuvers and Simons. For information about `Pillow`, you might start here: https://pillow.readthedocs.io/en/stable

Exercises

1. Use a Fialka with series-3K non-configurable rotors and the Polish keyboard to encrypt this English text with the given key. Use the punchcard in Figure 181.2. Discard punctuation but retain spaces, which become Й before encryption. Express the ciphertext in two ways: as Cyrillic output and as Latin output.

   daily key: `ИЗГЖВ БКДАЕ`
   message key: `ИЫШЬЮ ГДФХЯ`

   There was once an emperor, very great and mighty, and he ruled over an empire so large that no one knew where it began and where it ended. But if nobody could tell the exact extent of his sovereignty everybody was aware that the emperor's right eye laughed, while his left eye wept. One or two men of valour had the courage to go and ask him the reason of this strange fact, but he only laughed and said nothing; and the reason of the deadly enmity between his two eyes was a secret only known to the monarch himself.

   (from “The Fairy of the Dawn” in *The Violet Fairy Book* by Andrew Lang)

2. Use a Fialka with series-68 non-configurable rotors to encrypt this Russian text with the given key. Use the punchcard in Figure 181.2. Discard punctuation and spaces. The lower-case Russian letters almost always look like their upper-case counterparts; the only ones that do not are Аа, Бб, Ее, Ёё, and even then б is the only weird one.

   daily key: `ВИКАБ ГЖДЕЗ`
   message key: `ДЩСЧИ СХЫРС`

   Возможно ль? вместо роз, Амуром
   насажденных,
   Тюльпанов, гордо наклоненных,
   Душистых ландышей, ясминов и лилей,
   Которых ты всегда любила
   И прежде всякий день носила
   На мраморной груди твоей—
   Возможно ль, милая Климена?

   (from «Красавице, которая нюхала табак» by Aleksandr Sergeevich Pushkin)

3. Decrypt this ciphertext which was prepared with a Fialka using the series-6K configurable rotors and the Czech keyboard. The key is given, and the punchcard is shown below. Spaces, but not punctuation, were kept in the English plaintext.

daily key: ИДАЖЕ ЗВБКГ
ДЧВЩЫ НЫСАЕ
ГИДВБ ЕЗКАЖ
12112 11212
ЛВЩВФ ХАРВТ

message key: ЬЙФШЯ ЬЦЫНЦ

```
ШЙРЙНАЖТМРВТПЩЛШЯДЦСОСХТЛЖБСХПЮИХЧФЛАЧРКНЯЩГНЛОИНКЙБВСПМШЧЕЫ
ДИШЙЗНЕВДЬВЕЯЬСРТХКШЖФМЫЬЯЬЗЧРКНБШОЖЧЩВПЗДНТЦЫСБЗЧУЬРОХЛПХДФ
АСУГЗЖЗХРЫЖДЩУЙСЙИЫСДЙВРТНБАСКУАЛВМКЛНЦЗТЙКТРМЖЩФАЕФМУХЖИЫШР
ЧБЦОЫЛИТМЛЩЮНВМАМТРСШТКЗПЦЦНЙФЖЗЗТСНШЗБГКЬЛЬЕФПКОЕИЮЖХГЧДЖМБ
ШУХЬСТМЫБДВХУЫРРШСЫПЮЧЮЩФКГЫДЬЯПЕБСЮЖСИРЛБТФХПТЦЖСЬЕЦУЖКФЫАМ
ЦЯФЦЬХВБЧТОЬХПРПЮЫНОИЖФЬЖЧНХДКУОМРЙСЯВГХХНЖВЧЖХБШАНКМЖХЖФЩЦЛ
ВОСЦЗЮЗЮРЧЮАЙЛБАДЛШБЧЖЗЬИЖАЮЬЙЗЯПВХГГЩЖЮГВКПРФАЕХСЦРРЛЕЮШТНФ
ССЧЮПУМЙЛВЯДЮЗВАФЩАУЩЖАФРСУЩОФЯФБ
```

4. Find the missing parts of the message key. The text is in English using the Polish keyboard mapping. Spaces, but not punctuation, were retained in the plaintext. The rotor set is from series 3K. The punchcard is shown below.

daily key: КГБДИ ЖЗАВЕ

message key: ПШХГБ ?????

```
ТТТАГЩЖБШПТИЯЯСЕХДЮИКЛАПДЮОЗОМЩДСЛМФОФЛАЧТЗУЯЖЬДГЫДЙМАНСАЩРФ
ЯРВЮШДТХДФЫЫЧВРЮРАЛЗКЗШШГЬХДЩГРНЙПУЖЦНЙЯУСАЬФСИЧЬУИХЖТШКСТЯМ
АДРЖТОЩМПОЬФЙЙАДТРИОЫХРХБСШЕЯВЬЕУВЮЕТШЖГПЧОЖАХГШЬЯЬНМЬКПЖЬНЮ
ГФНОШАЕИЩЖЛЙЧЖФУОВФВКЧЧМГХРЖНААБЫДЧЯЮЬЕЙЬТЖХЛКДКЙЕКЬЗБЛЫУРНФ
МЕКДУПРЧЦБЬСЕХЩОШАННЗИЗБУГШЮЛЙПУЙСВКЬЖЮДФЬЧУААЩНЩИИЛМСУФЧДТГ
ЯЗТХЮУЦЦУНМГЬВЫНБЗЖЬФХМУРБАФЧОЕАЕЯВНЧРЮОЗЕЧРРЙУГФШУЧЛМЯТСМЮЕ
ИАЧББЮЗЮРТЕБФЗУМЧЫЬТДЬМПВЫБШБ
```

5. Learn the Russian language.

6. Используй твою симуляцию Фиалки чтобы зашифровать сообщение Кремлю.

Unit 182
Boolean variables and logical operations

In the next cipher machines that we will see, characters are encoded with electric signals that can be on (1) or off (0), and signals pass from one part of a machine to another with similar values. The basic thing that can have a value of 0 or 1 is called a *bit*. When we are talking about manipulating bits with logic, we may label them with letters and call them *boolean variables* (named after George Boole), and we often use T or TRUE to represent 1 and F or FALSE to represent 0.

The first thing we might want to do to a boolean variable is apply an unary operation to it. An *unary operation* (*unop*) is an operation that acts on a single variable; in arithmetic the negation operation − is an unary operation: $-x$ has the value that is just as far from zero as x but on the other side of it. Suppose we have a boolean variable A and want to see what sorts of unary operations can be applied to it. We can tabulate all possible values of A in a *truth table* and add a column for the output of every possible unary operation. There are four possibilities:

A	UNOP_0	UNOP_1	UNOP_2	UNOP_3
0	0	0	1	1
1	0	1	0	1

Two of these are trivial: UNOP_0 always gives FALSE and UNOP_3 always gives TRUE. Operator UNOP_1 gives the same value that it receives; electrical engineers call this operator a *buffer*, but for us it is useless. Operator UNOP_2 is the interesting one: it flips the bit of A. This operation we call NOT, and we write "NOT A" or "$\neg A$" (or some people write "$\overline{A}$" or "$\sim A$"). At the risk of being overly pedantic, its truth table is in Table 182.1. We can forget about the other three unops.

A	$\neg A$
0	1
1	0

Table 182.1: The NOT *operation.*

Next we look at binary operations. A *binary operation* (*binop*) is one that acts on two variables. You are already familiar with this concept, since addition (+), subtraction (−), multiplication (·), and division (/) are binary operations on real numbers. Since we are dealing with two boolean variables, each of which can have one of two values, there are $2^4 = 16$ possibilities:

A	B	BINOP_0	BINOP_1	BINOP_2	BINOP_3	BINOP_4	BINOP_5	BINOP_6	BINOP_7
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

A	B	BINOP_8	BINOP_9	BINOP_{10}	BINOP_{11}	BINOP_{12}	BINOP_{13}	BINOP_{14}	BINOP_{15}
0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

Again we have some trivial and useless operations: BINOP_0 and BINOP_{15} are constants, and BINOP_3, BINOP_5, BINOP_{12}, and BINOP_{10} are A, B, $\neg A$, and $\neg B$. That leaves ten interesting binary operations.

The AND *operation*

BINOP_1 is TRUE when and only when its operands A and B are both TRUE, so we can rename it the AND operation. It is equivalent to multiplication (modulo 2) (see the unit on modular arithmetic), and it has an identity element (1), and all non-zero elements (just 1) have inverses (still 1). Symbols for this operation are AND, $\wedge$, and $\cdot$.

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

Table 182.2: The AND *operation.*

The OR *operation*

BINOP$_7$ is TRUE when A is TRUE or B is TRUE, and FALSE otherwise, so we call it the OR operation. Symbols for this operation are OR and $\vee$.

A	B	$A \vee B$
0	0	0
0	1	1
1	0	1
1	1	1

Table 182.3: The OR *operation.*

Exclusive or

BINOP$_6$ is TRUE if and only if exactly one of its operands is TRUE. Because it excludes the case when both operands are TRUE, we call rename it the *exclusive or* operation, or XOR. It is equivalent to addition modulo 2 (see the unit on modular arithmetic), and it has an identity element (0), and all elements have inverses ($-0 = 0$ and $-1 = 1$). It is also the same as inequality, since it is TRUE only when the operands are different. It is also the negation of a bi-implication (see below), meaning that the truth of one operand implies the falsehood of the other. Symbols for this operation include XOR, $\oplus$, $\veebar$, $\not\leftrightarrow$, and $\not\equiv$, but we will usually use XOR and $\oplus$.

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Table 182.4: The XOR *operation.*

Nor

BINOP_8 is TRUE when A OR B is FALSE, and FALSE when A OR B is TRUE, so we call it the NOR operation. Symbols for it include NOR, $\downarrow$, and $\bar{\vee}$.

A	B	$A \downarrow B$
0	0	1
0	1	0
1	0	0
1	1	0

Table 182.5: The NOR *operation.*

Not and

BINOP_{14} is the opposite of AND, so we call it the NAND operation. Symbols for it include NAND, $\uparrow$, and $\bar{\wedge}$.

A	B	$A \uparrow B$
0	0	1
0	1	1
1	0	1
1	1	0

Table 182.6: The NAND *operation.*

Implications

A statement like "if A then B" means that whenever A is TRUE then B is also TRUE. It also means that when A is FALSE, then B can have any value. We say that A *implies* B, and write $A \rightarrow B$. From the table of binary operations, we see that BINOP_{13} fits this description: it is TRUE if A and B are both TRUE and FALSE if A is TRUE but B is FALSE; if A is FALSE then the implication is vacuously satisfied, and BINOP_{13} is TRUE. There is no reason that we cannot have implication in the other direction. This is called the *converse* of $A \rightarrow B$, and it is the operation BINOP_{11}. If the operands imply each other, then they must be equal to get a TRUE result. We call this *bi-implication* or *equivalence* or *exclusive nor*. Symbols for it

include $\leftrightarrow$ and $\equiv$, and XNOR. This operation is the opposite (the NOT) of XOR, and is the one we used to call BINOP_9. Finally, BINOP_2 and BINOP_4 are the NOTs of $\rightarrow$ and $\leftarrow$.

A	B	$A \rightarrow B$	$A \leftarrow B$	$A \leftrightarrow B$	$A \nrightarrow B$	$A \nleftarrow B$
0	0	1	1	1	0	0
0	1	1	0	0	0	1
1	0	0	1	0	1	0
1	1	1	1	1	0	0

Table 182.7: Implication operations.

Python tips

Python has a data type called `bool`. A variable of this type can have value `True` or `False`, which are defined as constants in the Python language. However, in a logical expression, any non-zero value is treated as `True`, and any zero value or empty item, such as an empty string or list or dictionary, is treated as `False`. Logical operators `and`, `or`, and `not` do what you expect. The (bitwise) operator `^` can be used as XOR between two `bool` variables.

Reading and references

Any book on symbolic logic.

Python reference: https://docs.python.org/3/reference/expressions.html#boolean-operations

George Boole, *The Mathematical Analysis of Logic: Being an Essay towards a Calculus of Deductive Reasoning*, 1847, https://www.gutenberg.org/ebooks/36884

George Boole, *An Investigation of the Laws of Thought*, 1854.

Wikipedia has many pages about boolean arithmetic and operations. Start with https://en.wikipedia.org/wiki/Boolean_algebra

Exercises

1. Use truth tables to show these facts for any boolean variables A, B, C. Tabulate all possible values for the inputs and check that the column for the left-hand side matches the column for the right-hand side for each equation. This is called *proof by exhaustion*, not only because it is tiring, but also because it exhausts all possibilities.

   a. $A \rightarrow B \;=\; (A \wedge B) \vee (\neg A)$

   b. $(A \wedge B) \wedge C \;=\; A \wedge (B \wedge C)$ (associativity of AND)

   c. $(A \vee B) \vee C \;=\; A \vee (B \vee C)$ (associativity of OR)

   d. $A \wedge (B \vee C) \;=\; (A \wedge B) \vee (A \wedge C)$ (distributivity of AND over OR)

   e. $A \vee (B \wedge C) \;=\; (A \vee B) \wedge (A \vee C)$ (distributivity of OR over AND)

   f. $A \vee (A \wedge B) \;=\; A$ (absorption)

   g. $A \wedge (A \vee B) \;=\; A$ (absorption)

   h. $(A \wedge B) \vee (B \wedge C) \vee (C \wedge A) \;=\; (A \vee B) \wedge (B \vee C) \wedge (C \vee A)$

2. Show that De Morgan's laws hold for any boolean variables A and B:

   a. $\neg (A \wedge B) = (\neg A) \vee (\neg B)$

   b. $\neg (A \vee B) = (\neg A) \wedge (\neg B)$

3. Convince yourself that any binary operation can be written as an expression containing only the operations AND, OR, and NOT (and parentheses). Now convince me.

4. Write a convincing argument that any n-ary logical operation can be written as an expression containing only the operations AND, OR, and NOT (and parentheses). An n-ary operation is one that acts on n operands. We can think of such things as *binary functions* with n arguments (a function that takes n boolean values and returns one boolean value). If your argument is sound, then you have proven that {AND, OR, NOT} is a *complete* set of operators and that any binary function can be written in terms of these three operations.

5. Show that {AND, NOT} is a complete set of operators. Hint: use Exercise 4 and De Morgan's law(s).

6. Show that {OR, NOT} is a complete set of operators.

7. Show that {NAND} is a complete set of operators. That's right—everything can be written in terms of NAND.

8. There is another binary operation that forms a complete set by itself. What is it? Prove it.

Unit 183
Exclusive or

The exclusive or (XOR) operation is special for cryptographers, so it deserves extra attention. Recall from Table 182.4 that its truth table is

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

We also mentioned in Unit 182 that XOR is equivalent to addition modulo 2. It should therefore not surprise you that it has these properties (for all boolean variables A, B, C):

- $(A \oplus B) \oplus C = A \oplus (B \oplus C)$ (associativity) (183.1)
- $A \oplus B = B \oplus A$ (commutivity) (183.2)
- $A \oplus 0 = A$ (identity element) (183.3)
- $0 \oplus 0 = 1 \oplus 1 = 0$ (inverses)

While you would expect the above properties simple because ({TRUE, FALSE}, XOR) is an abelian group (the same as $(\mathbb{Z}_2, +_2)$), it also has some other very nice properties:

- $A \oplus 1 = \neg A$ (183.4)
- $A \oplus A = 0$ (nilpotence) (183.5)
- $A \oplus \neg A = 1$
- $A \oplus B = C \quad \Rightarrow \quad A = B \oplus C$
- $A \oplus (\neg B) = (\neg A) \oplus B = \neg (A \oplus B)$

That fourth one is important, because when we use XOR as an encryption mechanism, we can apply the same key to decipher as we use to encipher. What we mean is clearer if we replace A and B with P (plaintext) and K (key), while C denotes the ciphertext:

$$P \oplus K = C \quad \Rightarrow \quad P = C \oplus K \tag{183.6}$$

$$(P \oplus K) \oplus K = P$$

Reading and references

Any good textbook on symbolic logic.

Wikipedia: https://en.wikipedia.org/wiki/Exclusive_or

Exercises

1. Use a truth tables or manipulate equations to show that for any boolean variables A and B,

   a. $A \oplus B = (A \vee B) \wedge \neg (A \wedge B)$

   b. $A \oplus B = (A \vee B) \wedge ((\neg A) \vee (\neg B))$

   c. $(A \oplus B) \oplus B = A$

2. Use truth tables or manipulate equations to prove each of the properties of XOR listed in this unit.

Unit 184
Baudot encoding

During and after World War II, texting (“SMS” [“short-message service”] for European readers) was done with large devices, called “teleprinters,” that did not fit easily into one’s pocket. Bits were very expensive then, and most users could not afford a complete byte of eight bits. Therefore, teleprinters used five parallel transmission lines, and a means of encoding characters with five bits (voltage on and off for each line) was needed. By this time, however, punctuation and digits had been invented, and message senders needed more than the thirty-two possibilities afforded by five bits. Hence, the symbol and letter shifts were invented. A special character, called *symbol shift*, changes the character set to one of digits and punctuation. Another special character, the *letter shift*, returns the teleprinter to the character set containing letters. Upper-case letters were all that existed, so there was no such thing as that sort of shift key. The entire scheme, including the assignments of bit patterns to characters, is called *Baudot* encoding. In Table 184.1 we see the bit patterns and the characters assigned to them for both letter and symbol mode. In addition, we see the notation used by cryptanalysts at Bletchley Park during World War II as they worked to break the cipher machines that used it. These machines used innovative ways to flip and/or rearrange bits with rotors. Unlike the Enigma and such machines, in which the rotors defined permutations of the alphabet, these new machines used rotors to control the flipping or permuting of the five bits that encoded the characters.

Python tips

The carriage return is written “`\r`” in strings and bytes, and line feed is “`\n`” (think “return” and “new line”). The end of a line usually has both of these: `\r\n`. A null character is written “`\0`”. The “bell” character is “`\a`”, and if your terminal supports it, it will be converted to a sound (this author’s terminal flashes a bell icon but makes no noise). It is possible that the “enquire” character is “`\x05`”, but we will not need this character in this book.

bits	letter mode	symbol mode	Bletchley notation
00000	[null]	[null]	/
10000	E	3	E
01000	[carriage return]	[carriage return]	4
11000	A	-	A
00100	[space]	[space]	9
10100	S	'	S
01100	I	8	I
11100	U	7	U
00010	[line feed]	[line feed]	3
10010	D	[enquire]	D
01010	R	4	R
11010	J	[bell]	J
00110	N	,	N
10110	F	!	F
01110	C	:	C
11110	K	(	K
00001	T	5	T
10001	Z	+	Z
01001	L	)	L
11001	W	2	W
00101	H	£	H
10101	Y	6	Y
01101	P	0	P
11101	Q	1	Q
00011	O	9	O
10011	B	?	B
01011	G	&	G
11011	[symbol shift]		+
00111	M	.	M
10111	X	/	X
01111	V	;	V
11111		[letter shift]	8

Table 184.1: Baudot symbol table.

Programming tasks

1. Implement an encoder/decoder that can translate between text containing letters, punctuation, and digits and five-bit Baudot.

2. Implement an encoder/decoder that can translate between five-bit Baudot and the Bletchley notation.

3. Create a function that takes text in Bletchley notation and prints it with spaces, punctuation, digits, and end-of-line characters.

4. Take your textual corpus (Task 1 of Unit 3), including spaces, punctuation, and occasional digits, and encode it with Bletchley Baudot. Use the result to compile a table of frequencies for the 32 symbols. Put this table into a data structure that you can import into your Python scripts.

5. Make a copy of your function that calculates the index of coincidence (Unit 12) and modify it to operate on strings of Baudot characters. Take some texts of various lengths (at least 100 characters) in order to find a range for the IoC of English text.

6. Take your textual corpus (Task 1 of Unit 3), including spaces, punctuation, and occasional digits, and encode it with Bletchley Baudot. From the result, compile a table of tetragram frequencies. The table will have 32^4 (just over a million) entries.

7. Use the table from Task 6 and make a modified copy of your tetragram-fitness function (Unit 11) that evaluates the fitness of a piece of text in Bletchley Baudot encoding. Experiment with texts of various lengths in order to find a range of fitness for typical English.

Unit 185
Siemens and Halske T52

The Siemens and Halske model T52 (code-named “Sturgeon” by British analysts) was an encryption machine that acted on the output (for encryption) or input (decryption) to a teleprinter. A teleprinter used five data lines and encoded/decoded its messages in Baudot (see Unit 184). The T52 performed two sorts of operations on the data in those five lines: it could flip a bit (change it from 0 to 1 or from 1 to 0), or it could swap two bits. Which bits were flipped and which were swapped was determined by ten wheels and some circuitry. Unlike the wheels in machines like the Enigma, the wheels of the T52 did not act on letters directly; rather, they provided control signals to the circuitry that performed operations on the five bits of the Baudot-encoded characters. The output of each wheel is mapped to the control of one bit operation by something like a plugboard and optionally some additional circuitry. There were five versions of the machine, which we will examine below; they differ in how the wheels rotate and in what additional circuitry they have.

The ten wheel are designated A, B, C, D, E, F, G, H, J, and K (“I” was skipped, probably because it looks like a Roman numeral). Each has its own number of positions: 73, 71, 69, 67, 65, 64, 61, 59, 53, and 47, respectively. These numbers are all coprime, so that if the wheels advance regularly (one step each after encrypting each character, as in some versions of the machine), then they will not return to their original configuration until 893,622,318,929,520,960 characters have been encrypted. At the time that the machine was in use, that was considered a big number. The wheels have pins that determine whether it emits a control signal that is active (“on” or “1”) or inactive (“off” or “0”) for each of its positions, which are numbered, starting from 1. The pin patterns of the ten wheels are given in Table 185.1.

```
A    ·××·×××···×··××··×··××××··×××·×·×·×××··××··×××·××·····×××···××××·×·····×
B    ·××××··××××××···×···×·×··××·×××××·····×××···××·×××·····××·×·×·×××·····×
C    ·×××××·×·××··×·××····××·×····××××·×××·×××···×··×··×××····××····××···×
D    ·××·×·×···×××·····××·×····×··×××··×××··×·×××···××··×··×·××××··×···×
E    ·×···×···×·×××××××·×·××××××··×·×··××·×····×·×××××··×××·×·×·××···×
F    ·×·×·×××××····×·××××··×·××·××××·××··×××····×····×××·××××××···×·×
G    ·××××···××··××···×·×××××·×··×··×××·×·××·×·××·×····××·×··×·×××
H    ·×··×××××···××××·×··××××···××·×···××·×·××···××××××··××·×·××
J    ·×·××·××··××·×·××···×××···×××··×·×××·×····××××···×·××
K    ·×····×××××···×··×·×××··×·×·×××··×××·××××···×××
```

Table 185.1: Pin patterns of the wheels of the T52. A cross indicates an active position (pin is present), and a dot indicates an inactive position (no pin). The first dot is at position 1.

Bit-flipping is done by five exclusive-or logic circuits. Exclusive OR (XOR) takes two input bits and outputs a 1 if and only if exactly one input is 1. Otherwise, the output is 0. See Table 185.2. The symbol for XOR is usually taken to be ⊕, since it has a close relationship with addition modulo 2 (see Unit 19) (actually, it *is* addition modulo 2). In the T52, the two inputs to an XOR circuit are a bit of the plaintext character and a control wire. Those control wires are labeled "I" for the circuit that acts on the first bit, "II" for the second, "III," "IV," and "V." So the first bit is flipped if the control wire I is active, the second if II is active, etc.

inputs	output of OR	output of XOR
0 0	0	0
0 1	1	1
1 0	1	1
1 1	1	0

Table 185.2: The OR *and exclusive* OR *(*XOR*) operations, from Tables 182.3 and 182.4. The output of* OR *is on if either input is on. The output of* XOR *is on if and only if exactly one input is on.*

Figure 185.1: Siemens and Halske T52 version D on display at the U.S. National Cryptologic Museum. Photo credit: U.S. National Security Agency.

Bit-swapping is performed by relays that connect to the data lines and which are controlled by the control lines from the wheels in such a way that if the control line is active (1) then the bits are not swapped, but if the control line is inactive (0) then they are. There are five such relays, and each can be plugged into the data lines at two points. Therefore, we need ten points of access to the data; they are numbered and displayed schematically in Figure 185.2. A relay plugged into points 1 and 5, for example, can swap the bits on lines 1 and 4. This swapping comes *after* a swap that is wired to points 4 and 6, for example. Any possible wiring of the permutation circuit can be represented as five transpositions (swappings) in a definite order. For example, the fixed standard wiring of version C of the machine is 1-2, 3-4, 5-6, 7-8, 9-10, which corresponds to transpositions (1 5), (4 5), (3 4), (2 3), and (1 2), in that order.

There are 5! = 120 possible permutations of five bits. However, with five transpositions, there are only $2^5 = 32$ that can be implemented. There happen to be 120 ways to wire the swapping relays so that 32 permutations can be enacted, and there are 240 ways to wire them so that 30 permutations are possible. These 360 wirings were considered acceptable by the Germans, and we have listed them in Table 185.3. Any wiring not listed would result in fewer possible permutations or in loops in the data lines that disrupt the operation of the machine.

Figure 185.2: Schematic diagram of the access points and their labels for the swapping relays in the permutation circuitry. For example, a relay plugged into points 1 and 5 controls the swapping of bits 1 and 4.

The rest of this section is divided into subsections for each of the five versions of the T52. They all use the same wheels, the same XOR circuitry, and the same permutation circuitry. However, some fix the wiring of the swapping relays so that the permutation circuitry cannot be changed. Some versions introduce new features, such as additional logic circuitry between the wheels and the operations on the data bits, or irregular stepping of the wheels.

1-2 3-4 5-6 7-8 9-10	1-2 3-4 5-6 7-9 8-10	1-2 3-4 5-7 6-8 9-10	1-2 3-4 5-7 6-9 8-10	1-2 3-4 5-8 6-10 7-9
1-2 3-4 5-8 6-9 7-10	1-2 3-4 5-9 6-10 7-8	1-2 3-4 5-9 6-8 7-10	1-2 3-5 4-6 7-8 9-10	1-2 3-5 4-6 7-9 8-10
1-2 3-5 4-7 6-8 9-10	1-2 3-5 4-7 6-9 8-10	1-2 3-5 4-8 6-10 7-9	1-2 3-5 4-8 6-9 7-10	1-2 3-5 4-9 6-10 7-8
1-2 3-5 4-9 6-8 7-10	1-2 3-6 4-10 5-8 7-9	1-2 3-6 4-10 5-9 7-8	1-2 3-6 4-7 5-8 9-10	1-2 3-6 4-7 5-9 8-10
1-2 3-6 4-8 5-10 7-9	1-2 3-6 4-8 5-7 9-10	1-2 3-6 4-9 5-10 7-8	1-2 3-6 4-9 5-7 8-10	1-2 3-7 4-10 5-8 6-9
1-2 3-7 4-10 5-9 6-8	1-2 3-7 4-6 5-8 9-10	1-2 3-7 4-6 5-9 8-10	1-2 3-7 4-8 5-10 6-9	1-2 3-7 4-8 5-6 9-10
1-2 3-7 4-9 5-10 6-8	1-2 3-7 4-9 5-6 8-10	1-2 3-8 4-10 5-6 7-9	1-2 3-8 4-10 5-7 6-9	1-2 3-8 4-6 5-10 7-9
1-2 3-8 4-6 5-9 7-10	1-2 3-8 4-7 5-10 6-9	1-2 3-8 4-7 5-9 6-10	1-2 3-8 4-9 5-6 7-10	1-2 3-8 4-9 5-7 6-10
1-2 3-9 4-10 5-6 7-8	1-2 3-9 4-10 5-7 6-8	1-2 3-9 4-6 5-10 7-8	1-2 3-9 4-6 5-8 7-10	1-2 3-9 4-7 5-10 6-8
1-2 3-9 4-7 5-8 6-10	1-2 3-9 4-8 5-6 7-10	1-2 3-9 4-8 5-7 6-10	1-3 2-4 5-6 7-8 9-10	1-3 2-4 5-6 7-9 8-10
1-3 2-4 5-7 6-8 9-10	1-3 2-4 5-7 6-9 8-10	1-3 2-4 5-8 6-10 7-9	1-3 2-4 5-8 6-9 7-10	1-3 2-4 5-9 6-10 7-8
1-3 2-4 5-9 6-8 7-10	1-3 2-5 4-6 7-8 9-10	1-3 2-5 4-6 7-9 8-10	1-3 2-5 4-7 6-8 9-10	1-3 2-5 4-8 6-10 7-9
1-3 2-5 4-8 6-9 7-10	1-3 2-5 4-9 6-8 7-10	1-3 2-6 4-10 5-8 7-9	1-3 2-6 4-10 5-9 7-8	1-3 2-6 4-7 5-8 9-10
1-3 2-6 4-7 5-9 8-10	1-3 2-6 4-8 5-10 7-9	1-3 2-6 4-8 5-7 9-10	1-3 2-6 4-9 5-10 7-8	1-3 2-6 4-9 5-7 8-10
1-3 2-7 4-10 5-9 6-8	1-3 2-7 4-6 5-8 9-10	1-3 2-7 4-6 5-9 8-10	1-3 2-7 4-8 5-10 6-9	1-3 2-7 4-8 5-6 9-10
1-3 2-7 4-9 5-10 6-8	1-3 2-8 4-10 5-6 7-9	1-3 2-8 4-10 5-7 6-9	1-3 2-8 4-6 5-10 7-9	1-3 2-8 4-6 5-9 7-10
1-3 2-8 4-7 5-10 6-9	1-3 2-8 4-7 5-9 6-10	1-3 2-8 4-9 5-6 7-10	1-3 2-8 4-9 5-7 6-10	1-3 2-9 4-10 5-7 6-8
1-3 2-9 4-6 5-10 7-8	1-3 2-9 4-6 5-8 7-10	1-3 2-9 4-7 5-10 6-8	1-3 2-9 4-8 5-6 7-10	1-3 2-9 4-8 5-7 6-10
1-4 2-10 3-6 5-8 7-9	1-4 2-10 3-6 5-9 7-8	1-4 2-10 3-7 5-8 6-9	1-4 2-10 3-7 5-9 6-8	1-4 2-10 3-8 5-6 7-9
1-4 2-10 3-8 5-7 6-9	1-4 2-10 3-9 5-6 7-8	1-4 2-10 3-9 5-7 6-8	1-4 2-5 3-6 7-8 9-10	1-4 2-5 3-6 7-9 8-10
1-4 2-5 3-7 6-8 9-10	1-4 2-5 3-7 6-9 8-10	1-4 2-5 3-8 6-10 7-9	1-4 2-5 3-8 6-9 7-10	1-4 2-5 3-9 6-10 7-8
1-4 2-5 3-9 6-8 7-10	1-4 2-6 3-10 5-8 7-9	1-4 2-6 3-10 5-9 7-8	1-4 2-6 3-5 7-8 9-10	1-4 2-6 3-5 7-9 8-10
1-4 2-6 3-8 5-7 9-10	1-4 2-6 3-8 5-9 7-10	1-4 2-6 3-9 5-7 8-10	1-4 2-6 3-9 5-8 7-10	1-4 2-7 3-10 5-8 6-9
1-4 2-7 3-10 5-9 6-8	1-4 2-7 3-5 6-8 9-10	1-4 2-7 3-5 6-9 8-10	1-4 2-7 3-8 5-6 9-10	1-4 2-7 3-8 5-9 6-10
1-4 2-7 3-9 5-6 8-10	1-4 2-7 3-9 5-8 6-10	1-4 2-8 3-10 5-6 7-9	1-4 2-8 3-10 5-7 6-9	1-4 2-8 3-5 6-10 7-9
1-4 2-8 3-5 6-9 7-10	1-4 2-8 3-6 5-7 9-10	1-4 2-8 3-6 5-9 7-10	1-4 2-8 3-7 5-6 9-10	1-4 2-8 3-7 5-9 6-10
1-4 2-9 3-10 5-6 7-8	1-4 2-9 3-10 5-7 6-8	1-4 2-9 3-5 6-10 7-8	1-4 2-9 3-5 6-8 7-10	1-4 2-9 3-6 5-7 8-10
1-4 2-9 3-6 5-8 7-10	1-4 2-9 3-7 5-6 8-10	1-4 2-9 3-7 5-8 6-10	1-5 2-10 3-6 4-8 7-9	1-5 2-10 3-7 4-8 6-9
1-5 2-10 3-7 4-9 6-8	1-5 2-10 3-8 4-6 7-9	1-5 2-10 3-9 4-6 7-8	1-5 2-10 3-9 4-7 6-8	1-5 2-4 3-6 7-8 9-10
1-5 2-4 3-6 7-9 8-10	1-5 2-4 3-7 6-8 9-10	1-5 2-4 3-7 6-9 8-10	1-5 2-4 3-8 6-10 7-9	1-5 2-4 3-8 6-9 7-10
1-5 2-4 3-9 6-10 7-8	1-5 2-4 3-9 6-8 7-10	1-5 2-6 3-10 4-8 7-9	1-5 2-6 3-10 4-9 7-8	1-5 2-6 3-4 7-8 9-10
1-5 2-6 3-4 7-9 8-10	1-5 2-6 3-8 4-7 9-10	1-5 2-6 3-8 4-9 7-10	1-5 2-6 3-9 4-7 8-10	1-5 2-6 3-9 4-8 7-10
1-5 2-7 3-10 4-8 6-9	1-5 2-7 3-10 4-9 6-8	1-5 2-7 3-4 6-8 9-10	1-5 2-7 3-8 4-6 9-10	1-5 2-7 3-9 4-6 8-10
1-5 2-7 3-9 4-8 6-10	1-5 2-8 3-10 4-6 7-9	1-5 2-8 3-10 4-7 6-9	1-5 2-8 3-4 6-10 7-9	1-5 2-8 3-4 6-9 7-10
1-5 2-8 3-6 4-7 9-10	1-5 2-8 3-6 4-9 7-10	1-5 2-8 3-7 4-6 9-10	1-5 2-8 3-7 4-9 6-10	1-5 2-9 3-10 4-6 7-8
1-5 2-9 3-10 4-7 6-8	1-5 2-9 3-4 6-8 7-10	1-5 2-9 3-6 4-8 7-10	1-5 2-9 3-7 4-6 8-10	1-5 2-9 3-7 4-8 6-10
1-6 2-10 3-4 5-8 7-9	1-6 2-10 3-4 5-9 7-8	1-6 2-10 3-5 4-8 7-9	1-6 2-10 3-5 4-9 7-8	1-6 2-10 3-8 4-7 5-9
1-6 2-10 3-8 4-9 5-7	1-6 2-10 3-9 4-7 5-8	1-6 2-10 3-9 4-8 5-7	1-6 2-4 3-10 5-8 7-9	1-6 2-4 3-10 5-9 7-8
1-6 2-4 3-7 5-8 9-10	1-6 2-4 3-7 5-9 8-10	1-6 2-4 3-8 5-10 7-9	1-6 2-4 3-8 5-7 9-10	1-6 2-4 3-9 5-10 7-8
1-6 2-4 3-9 5-7 8-10	1-6 2-5 3-10 4-8 7-9	1-6 2-5 3-10 4-9 7-8	1-6 2-5 3-7 4-8 9-10	1-6 2-5 3-7 4-9 8-10
1-6 2-5 3-8 4-10 7-9	1-6 2-5 3-8 4-7 9-10	1-6 2-5 3-9 4-10 7-8	1-6 2-5 3-9 4-7 8-10	1-6 2-7 3-4 5-8 9-10
1-6 2-7 3-4 5-9 8-10	1-6 2-7 3-5 4-8 9-10	1-6 2-7 3-5 4-9 8-10	1-6 2-7 3-8 4-10 5-9	1-6 2-7 3-8 4-9 5-10
1-6 2-7 3-9 4-10 5-8	1-6 2-7 3-9 4-8 5-10	1-6 2-8 3-10 4-7 5-9	1-6 2-8 3-10 4-9 5-7	1-6 2-8 3-4 5-10 7-9
1-6 2-8 3-4 5-7 9-10	1-6 2-8 3-5 4-10 7-9	1-6 2-8 3-5 4-7 9-10	1-6 2-8 3-7 4-10 5-9	1-6 2-8 3-7 4-9 5-10
1-6 2-9 3-10 4-7 5-8	1-6 2-9 3-10 4-8 5-7	1-6 2-9 3-4 5-10 7-8	1-6 2-9 3-4 5-7 8-10	1-6 2-9 3-5 4-10 7-8
1-6 2-9 3-5 4-7 8-10	1-6 2-9 3-7 4-10 5-8	1-6 2-9 3-7 4-8 5-10	1-7 2-10 3-4 5-9 6-8	1-7 2-10 3-5 4-8 6-9
1-7 2-10 3-5 4-9 6-8	1-7 2-10 3-8 4-6 5-9	1-7 2-10 3-9 4-6 5-8	1-7 2-10 3-9 4-8 5-6	1-7 2-4 3-10 5-8 6-9
1-7 2-4 3-10 5-9 6-8	1-7 2-4 3-6 5-8 9-10	1-7 2-4 3-6 5-9 8-10	1-7 2-4 3-8 5-10 6-9	1-7 2-4 3-8 5-6 9-10
1-7 2-4 3-9 5-10 6-8	1-7 2-4 3-9 5-6 8-10	1-7 2-5 3-10 4-8 6-9	1-7 2-5 3-10 4-9 6-8	1-7 2-5 3-6 4-8 9-10
1-7 2-5 3-8 4-6 9-10	1-7 2-5 3-9 4-10 6-8	1-7 2-5 3-9 4-6 8-10	1-7 2-6 3-4 5-8 9-10	1-7 2-6 3-4 5-9 8-10
1-7 2-6 3-5 4-8 9-10	1-7 2-6 3-5 4-9 8-10	1-7 2-6 3-8 4-10 5-9	1-7 2-6 3-8 4-9 5-10	1-7 2-6 3-9 4-10 5-8
1-7 2-6 3-9 4-8 5-10	1-7 2-8 3-10 4-6 5-9	1-7 2-8 3-10 4-9 5-6	1-7 2-8 3-4 5-10 6-9	1-7 2-8 3-4 5-6 9-10
1-7 2-8 3-5 4-10 6-9	1-7 2-8 3-5 4-6 9-10	1-7 2-8 3-6 4-10 5-9	1-7 2-8 3-6 4-9 5-10	1-7 2-9 3-10 4-6 5-8
1-7 2-9 3-10 4-8 5-6	1-7 2-9 3-4 5-10 6-8	1-7 2-9 3-5 4-10 6-8	1-7 2-9 3-5 4-6 8-10	1-7 2-9 3-6 4-8 5-10
1-8 2-10 3-4 5-6 7-9	1-8 2-10 3-4 5-7 6-9	1-8 2-10 3-5 4-6 7-9	1-8 2-10 3-5 4-7 6-9	1-8 2-10 3-6 4-7 5-9
1-8 2-10 3-6 4-9 5-7	1-8 2-10 3-7 4-6 5-9	1-8 2-10 3-7 4-9 5-6	1-8 2-4 3-10 5-6 7-9	1-8 2-4 3-10 5-7 6-9
1-8 2-4 3-6 5-10 7-9	1-8 2-4 3-6 5-9 7-10	1-8 2-4 3-7 5-10 6-9	1-8 2-4 3-7 5-9 6-10	1-8 2-4 3-9 5-6 7-10
1-8 2-4 3-9 5-7 6-10	1-8 2-5 3-10 4-6 7-9	1-8 2-5 3-10 4-7 6-9	1-8 2-5 3-6 4-10 7-9	1-8 2-5 3-6 4-9 7-10
1-8 2-5 3-7 4-10 6-9	1-8 2-5 3-7 4-9 6-10	1-8 2-5 3-9 4-6 7-10	1-8 2-5 3-9 4-7 6-10	1-8 2-6 3-10 4-7 5-9
1-8 2-6 3-10 4-9 5-7	1-8 2-6 3-4 5-10 7-9	1-8 2-6 3-4 5-9 7-10	1-8 2-6 3-5 4-10 7-9	1-8 2-6 3-5 4-9 7-10
1-8 2-6 3-9 4-10 5-7	1-8 2-6 3-9 4-7 5-10	1-8 2-7 3-10 4-6 5-9	1-8 2-7 3-10 4-9 5-6	1-8 2-7 3-4 5-10 6-9
1-8 2-7 3-4 5-9 6-10	1-8 2-7 3-5 4-10 6-9	1-8 2-7 3-5 4-9 6-10	1-8 2-7 3-9 4-10 5-6	1-8 2-7 3-9 4-6 5-10
1-8 2-9 3-4 5-6 7-10	1-8 2-9 3-4 5-7 6-10	1-8 2-9 3-5 4-6 7-10	1-8 2-9 3-5 4-7 6-10	1-8 2-9 3-6 4-10 5-7
1-8 2-9 3-6 4-7 5-10	1-8 2-9 3-7 4-10 5-6	1-8 2-9 3-7 4-6 5-10	1-9 2-10 3-4 5-7 6-8	1-9 2-10 3-5 4-6 7-8
1-9 2-10 3-5 4-7 6-8	1-9 2-10 3-6 4-8 5-7	1-9 2-10 3-7 4-6 5-8	1-9 2-10 3-7 4-8 5-6	1-9 2-4 3-10 5-6 7-8
1-9 2-4 3-10 5-7 6-8	1-9 2-4 3-6 5-10 7-8	1-9 2-4 3-6 5-8 7-10	1-9 2-4 3-7 5-10 6-8	1-9 2-4 3-7 5-8 6-10
1-9 2-4 3-8 5-6 7-10	1-9 2-4 3-8 5-7 6-10	1-9 2-5 3-10 4-6 7-8	1-9 2-5 3-10 4-7 6-8	1-9 2-5 3-6 4-8 7-10
1-9 2-5 3-7 4-10 6-8	1-9 2-5 3-7 4-8 6-10	1-9 2-5 3-8 4-6 7-10	1-9 2-6 3-10 4-7 5-8	1-9 2-6 3-10 4-8 5-7
1-9 2-6 3-4 5-10 7-8	1-9 2-6 3-4 5-8 7-10	1-9 2-6 3-5 4-10 7-8	1-9 2-6 3-5 4-8 7-10	1-9 2-6 3-8 4-10 5-7
1-9 2-6 3-8 4-7 5-10	1-9 2-7 3-10 4-6 5-8	1-9 2-7 3-10 4-8 5-6	1-9 2-7 3-4 5-10 6-8	1-9 2-7 3-5 4-10 6-8
1-9 2-7 3-5 4-8 6-10	1-9 2-7 3-8 4-6 5-10	1-9 2-8 3-4 5-6 7-10	1-9 2-8 3-4 5-7 6-10	1-9 2-8 3-5 4-6 7-10
1-9 2-8 3-5 4-7 6-10	1-9 2-8 3-6 4-10 5-7	1-9 2-8 3-6 4-7 5-10	1-9 2-8 3-7 4-10 5-6	1-9 2-8 3-7 4-6 5-10

Table 185.3: Acceptable permutation-circuit wirings for the T52 versions A, B, and D.

Versions A and B

The first two versions of the machine differed only superficially; their operation was identical. They contain no new elements beyond what we have already discussed. The machine is initialized by putting the ten wheels into the positions specified by part of the key. The output of each wheel is assigned to control a bit-flipping (XOR) or a bit-swapping (one of the relays) via a plugboard; this mapping is another part of the key. After each character is encrypted, all ten wheels advance (rotate) their positions by one; this is what we call "regular stepping." A schematic diagram of the T52a/b is in Figure 185.4.

To specify the key, we need to give the assignments and starting positions of the ten wheels. The names of the wheels are implied, so if we write the key as

9-10, 1-6, I, II, 3-5, 4-7, IV, 2-8, III, V; 52, 19, 21, 50, 47, 24, 14, 46, 39, 7

we mean that wheel A starts at position 52 and is assigned to a swapping relay that is connected to points 9 and 10 (see Figure 185.2), wheel B starts at 19 and is assigned to a swapping relay between points 1 and 6. wheel C starts at 21 and is assigned to the XOR on data line 1, etc.

Let's see how to translate between the language in which the key is expressed into a set of operations on the data bits. For the example above, we see that the XOR operations are controlled by the output of wheels C, D, J, G, K. To glean which bits are swapped in the permutation circuit, we can shift things in Figure 185.2 so that points connected to the same relay are in the same vertical segment, as seen in Figure 185.3. From this we see that wheel B controls a transposition (1 3), which occurs first (if it occurs, based on the output of B). It is followed by (2 5) which is controlled by H, then (3 4) controlled by F, (4 5) controlled by E, and finally (1 2) controlled by A.

To see that we understand the machine and that your implementation of it is working, consider the following test vector. Encode the first line of Hamlet's soliloquy in Baudot:

`TO9BE9OR9NOT9TO9BE+N89THAT9IS9THE9QUESTION+C843`

Encrypt with T52a/b using the example key to get

`9NQHMTA9XYJDYSFVUOAZ9/YEIW/8GX84RAURJSRA/9KOMH9`

The encryption of the first letter goes as follows. The wheels are in their initial positions, so A's output is 0, B 0, C 0, D 0, E 1, F 0, G 1, H 1, J 0, K 1. The first character is `T`, which has bits 00001. Its first bit is XORed with 0 (the output of C) to remain 0. The second bit is XORed with 0 (output of D) to remain 0. The third is XORed with 0 (output of J) to remain 0. The fourth is XORed with 1 (output of G) to become 1. The last is XORed with 1 (output of K) to become 0. After the XOR circuitry, the data lines carry the bits 00010. The output of B is 0, so the first transposition occurs, and bits 1 and 3 are swapped (no effect). H outputs 1, so its transposition does not occur. Because F outputs 0, the transpositions (3 4) does occur and now we have 00100 on the data lines. The output of wheel E is 1, so its transposition does not occur. The last transposition, (1 2) occurs, but it swaps two zero bits. The output of the machine is therefore 00100, which is written as `9` in Bletchley notation and which represents a space.

Figure 185.3: Example of shifting the access points in the permutation circuit so that we can glean the transpositions. Note that access points cannot be shifted through one another.

Figure 185.4: Schematic diagram of the T52a/b. The diagram is incomplete; only two transpositions are shown in the permutation circuitry.

Version C

Version C has the following changes from versions A and B:

- Between the wheels and the wheel-assignment plugboard is a set of switches for setting a message key and circuitry to shuffle the signals on the control lines from the wheels. When too many messages are sent with the same daily key, it is easy for analysts to reconstruct the plaintexts. Modifying the key for each message obviates this problem.
- After the control lines leave the wheel-assignment plugboard, they pass through what is called the “SR logic” module which uses XOR circuits to further scramble them.
- The permutation circuitry is fixed so that only the wiring 1-2, 3-4, 5-6, 7-8, 9-10 is used. The corresponding transpositions (in order) are (1 5), (4 5), (3 4), (2 3), (1 2). Because only one configuration of the permutation circuitry is used, the connections are renamed with abbreviated labels: “1” for 1-2, “3” for 3-4, “5” for 5-6, “7” for 7-8, and “9” for 9-10.

The message-key unit has five switches, each of which can be set to positions labeled `P`, `S`, `T`, `U`, `W`, `X`, `Y`, and `Z`. Each switch’s position activates up to three transposition relays that can swap the signals on the control lines leaving the wheels; see Table 185.4. The transposition relays are placed into three layers, so that layer 1 acts before layer 2, which acts before layer 3; see Table 185.5 for the transpositions and their placements into the three layers. It goes without saying that the action of the message-key unit and the wheel-assignment plugboard together are like using the plugboard with a new set of assignments; the addition of the message-key unit does not change the T52 in a fundamental way.

As an example of the function of the message-key unit, consider message key `WUPSY`. Switch 1 is in position `W`, which means that transpositions T01 and T06 are active. Switch 2 is in position `U`, so T02 and T07 are active. Switch 3 activates T03. Switch 4 activates T09 and T14. Switch 5 activates T05 and T15. In the first layer of transposition relays, T01, T02, T03, and T05 are active. From Table 185.5 we see that the control signals from wheels A and B are swapped, from C and D are swapped, from E and F are swapped, and from J and K are swapped. In other words, line B now carries the signal from wheel A, line A now carries the signal from wheel B, etc. In the second layer, T06, T07, and T09 are active. So the signals on lines B and C are swapped, D and E are swapped, and H and J are swapped. In layer 3, T14 and T15 are active, and they swap the signals on lines D and J, and E and K. The overall permutation enacted by the message-key unit in our example takes the signal from wheel A and puts it onto line C, the signal from wheel B onto line A, etc.:

A	B	C	D	E	F	G	H	J	K
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
C	A	J	B	F	K	G	D	E	H

switch number	transpositions (swappings)	switch positions P	S	T	U	W	X	Y	Z
1	T01				✓	✓	✓		✓
	T06		✓			✓		✓	✓
	T11			✓			✓	✓	✓
2	T02			✓	✓	✓		✓	
	T07	✓			✓		✓	✓	
	T12		✓			✓	✓	✓	
3	T03	✓	✓	✓		✓			
	T08		✓		✓	✓		✓	
	T13			✓	✓	✓			✓
4	T04	✓		✓				✓	✓
	T09		✓	✓		✓			✓
	T14	✓	✓	✓			✓		
5	T05	✓				✓	✓	✓	
	T10	✓		✓			✓		✓
	T15	✓			✓			✓	✓

Table 185.4: Switches in the message-key unit. Each of the five switches can be set to one of P, S, T, U, W, X, Y, Z. *The effect of each switch is to activate up to three swappings (transpositions) of the control signals from the wheels* A, ..., K. *See Table 185.5 for the action of the transpositions.*

transposition number	layer	control wires affected
T01	1	A B
T02	1	C D
T03	1	E F
T04	1	G H
T05	1	J K
T06	2	B C
T07	2	D E
T08	2	F G
T09	2	H J
T10	2	A K
T11	3	A F
T12	3	B G
T13	3	C H
T14	3	D J
T15	3	E K

Table 185.5: Transpositions in the message-key unit. Transpositions that are active in layer 1 occur before layer 2, which occur before layer 3. If a transposition is active, then it swaps the signals on the designated control lines.

output	inputs									
	I	II	III	IV	V	1	3	5	7	9
I		X		X				X	X	
II			X		X		X	X		
III		X			X	X	X			
IV	X		X			X				X
V	X			X					X	X
1	X		X			X	X			
3		X		X			X	X		
5			X		X			X	X	
7	X			X					X	X
9		X			X	X				X

Table 185.6: SR logic for version C. Each output line carries the XOR *of four input lines.*

The wheel-assignment plugboard maps the control lines A, ..., K to ten lines labeled I, II, III, IV, V, 1, 3, 5, 7, and 9. These labels reflect their rôles in controlling the XOR and permutation circuitry. But before they carry out those rôles, they pass through the SR-logic unit. Each of the outputs of this unit is the XOR of four of the inputs; see Table 185.6 for the assignments of the inputs to outputs.

The key for T52c consists of the settings of the wheel-assignment plugboard (with the abbreviated labels for permutation relays), the initial wheel positions, and the message key. For example:

9, 1, I, II, 3, 7, IV, 5, III, V; 52, 19, 21, 50, 47, 24, 14, 46, 39, 7; `WUPSY`

With this key, the first line of Hamlet's soliloquy is encrypted to

`FAYUMGR+UHOEC8/8BB+STFJKAMWX/GPZJMLGUZJZZV/JJ9G`

Version CA

Version CA is identical to version C with one exception: the SR-logic circuitry is changed (see Table 185.7) to reduce some vulnerabilities.

With the same key and plaintext as in our example for version C, the ciphertext for version CA becomes

`E9VNT9BJL9GFXMMM+/CO/JG9CYNGL9RBDHPJA4BDDSJP99O`

output	inputs									
	I	II	III	IV	V	1	3	5	7	9
I		X			X	X	X			
II	X	X	X	X						
III				X			X	X		X
IV	X		X					X	X	
V			X		X		X	X		
1	X			X					X	X
3						X	X	X	X	
5	X	X			X				X	
7		X		X		X				X
9			X		X	X				X

Table 185.7: SR logic for version CA. Each output line carries the XOR *of four input lines.*

Version D

Version D has the same features as A and B, with one addition: irregular stepping of the wheels. For each wheel, a pin is tapped at a specific offset further ahead of its current position (offsets are given in Table 185.8). The outputs of those taps are carried on ten new control lines, labeled a, b, ..., k. An eleventh control line carries the middle bit (bit 3) of the previous plaintext character and is labeled z. Whether the eleventh control line is used depends on the setting of a switch on the machine, called the "klartext feature" or "KTF" ("klartext" simply means "plaintext"), and which is set to ON or OFF. Whenever a control line is active, we say that it carries a 1 or TRUE; when inactive, a 0 or FALSE. Whether a wheel advances to the next position (steps) depends on the evaluation of a logical expression involving the truth values of the control lines. What that expression is also depends on whether the KTF is in use. See Table 185.9 for the expressions for each wheel for each KTF setting.

The key of the T52 version D resembles the key for version A/B, with the addition of the KTF setting. For example, if we take the key we used for version A/B, and set KTF off, the key is

9-10, 1-6, I, II, 3-5, 4-7, IV, 2-8, III, V; 52, 19, 21, 50, 47, 24, 14, 46, 39, 7; OFF

and the first line of Hamlet's soliloquy is encrypted to

`9983TW+WDHYV8V+8WVOJCOZSERDSU++QBXPVAR3LF+4PD8F`

When the KTF feature is used and the key is

9-10, 1-6, I, II, 3-5, 4-7, IV, 2-8, III, V; 52, 19, 21, 50, 47, 24, 14, 46, 39, 7; ON

the ciphertext is

`9989NAK43O4GQYQ8CMZVZUQZOHTAB+3FVYOR9+3LIU9CZVB`

wheel	stepping-pin offset
A	25
B	24
C	23
D	23
E	22
F	22
G	20
H	20
J	18
K	16

Table 185.8: Offsets for the pins controlling irregular stepping. The pin whose signal helps to control irregular stepping is the given number of places beyond the current position of the wheel (modulo the size of the wheel).

wheel	logical expression when KTF=OFF	logical expression when KTF=ON
A	(NOT e) OR (NOT f)	b OR c OR z
B		(NOT c) OR d OR z
C		(NOT d) OR e
D		(NOT e) OR (NOT f)
E	f OR (NOT g)	f OR (NOT g) OR (NOT z)
F	g OR h	g OR h OR (NOT z)
G	(NOT h) OR (NOT j)	
H	j OR (NOT k)	
J	(NOT a) OR k	
K	(NOT d) OR e	a OR (NOT b)

Table 185.9: Logic of the irregular stepping in versions D and E. A wheel steps if the required logical expression evaluates to 1=TRUE*. The logical expression depends on whether the klartext feature is* ON *or* OFF*. The variables in the logical expressions are the outputs of the pin from the corresponding wheel (“*a*” for wheel* A*, etc.) at an offset given in Table 185.8. The variable* z *is the middle bit (bit 3) of the previous plaintext character. Variables carry the values* 0=FALSE *or* 1=TRUE*. The* NOT *operation flips a bit (*0↔1*). See Table 185.2 for a definition of the* OR *operation.*

Version E

Version E resembles versions C and CA, with these changes:

- The message-key unit is not present.
- The SR logic is changed. See Table 185.10.
- Irregular wheel stepping (including the KTF) is added in the same way as in version D.

The key is specified as it was for versions C and CA, with the modification that the message key is replaced by the KTF setting. For example, using the key

9, 1, I, II, 3, 7, IV, 5, III, V; 52, 19, 21, 50, 47, 24, 14, 46, 39, 7; OFF

we encrypt the first line of Hamlet's soliloquy to

`SVPWX9PT3ULW3TFMKE3KC4N9CPNESJSJXVTVJ/SGOOJGIPI`

With the key

9, 1, I, II, 3, 7, IV, 5, III, V; 52, 19, 21, 50, 47, 24, 14, 46, 39, 7; ON

we get

`SVP4AV9TZWPMZRTEMQQQBYI3WAVQ+NTECMX/IHSX8/WCQTR`

output	inputs									
	I	II	III	IV	V	1	3	5	7	9
I		X		X				X	X	
II					X		X	X		X
III	X			X		X	X			
IV	X		X		X					X
V	X			X					X	X
1			X		X	X	X			
3	X	X	X	X						
5			X		X			X	X	
7		X				X	X	X		
9		X				X			X	X

Table 185.10: SR logic for version E. Each output line carries the XOR *of four input lines.*

Figure 185.5: Overview of the T52. Each module is labeled according to which versions of the machine it appears in. All versions have the ten wheels. From them come ten control lines, A, ..., K. In versions C and CA, the control wires pass through the message-key unit which shuffles their signals; in other versions the control lines run directly to the wheel-assignment module, which is present in all models and which also shuffles the signals. In versions C, CA, and E, the control wires enter the SR-logic module which modify the signals in such a way that each output depends on four of the inputs. Five control lines actuate XOR *operations in the XOR module, while the other five control the permutation module. In versions D and E, there is an irregular-stepping-control module. It takes signals from the wheels and plaintext to determine which wheels advance.*

Reading and references

Frode Weierud, “Bletchley Park’s Sturgeon, the Fish That Laid No Eggs,” *The Rutherford Journal: The New Zealand Journal for the History and Philosophy of Science and Technology* 1 (2005-2006), https://www.rutherfordjournal.org/article010106.html (Figure 2 of this document has errors)

George Lasry, “T52 Functional Description.” This document can be found in the supplementary materials for some challenges on MysteryTwister, in this zip file: https://mysterytwister.org/media/challenges/add/mtc3-lasry-T52-add.zip

George Lasry, “T52 Cryptanalysis.” This document can be found in the supplementary materials for some challenges on MysteryTwister, in this zip file: https://mysterytwister.org/media/challenges/add/mtc3-lasry-T52-add.zip

Donald W. Davies, “The Early Models of the Siemens and Halske T52 Cipher Machine,” *Cryptologia* 7:3 (1983) 235-253, https://doi.org/10.1080/0161-118391857964

Donald W. Davies, “The Siemens and Halske T52e Cipher Machine,” *Cryptologia* 6:4 (1982) 289-308, https://doi.org/10.1080/0161-118291857118

Donald W. Davies, “New Information on the History of the Siemens and Halske T52 Cipher Machines,” *Cryptologia* 18:2 (1994) 141-146, https://doi.org/10.1080/0161-119491882801

Wolfgang W. Mache, “The Geheimschreiber,” *Cryptologia* 10:4 (1986) 230-242, https://doi.org/10.1080/0161-118691861065

Wolfgang W. Mache, “The Siemens Cipher Teletype in the History of Telecommunications,” *Cryptologia* 13:2 (1989) 97-117, https://doi.org/10.1080/0161-118991863817

David Kahn, “The Geheimschreiber,” *Cryptologia* 3:4 (1979) 210-214, https://doi.org/10.1080/0161-117991854089

Crypto Museum, “T-52 Geheimschreiber: Teleprinter cipher machine (STURGEON),” https://www.cryptomuseum.com/crypto/siemens/t52/index.htm

Programming tasks

1. Simulate the T52. Include a switch to determine the version (A/B, C, CA, D, E). For versions A, B, and D, include a function that checks that a permutation-circuit wiring is acceptable (there may be ways of doing this without using a look-up table).

2. Develop a brute-force attack for a situation in which the wheel assignments are known and some of the wheel positions are known. It may be faster to use the index of coincidence to reject most decryptions, and the fitness as a second stage of verification. Using a faster, lower-level language (such as C) would also be a benefit.

3. Develop a hill-climbing attack that maximizes the index of coincidence if the wheel assignments are known but their positions are not. This is not likely to work for versions D and E.

Exercises

1. Decrypt this ciphertext with version A/B and the given key.

   1-4, 2-5, IV, I, II, 3-9, 6-8, 7-10, III, V; 43, 45, 15, 67, 50, 35, 22, 29, 39, 42


   ```
   RJLQHMILUAMODRT8LX8IYVFYW4WPB9WUW/VWCNKPXW/R3F+AA3F9EZGLNJABW8KP
   IT/IRP3NKHEACWGBJ4W9DN8/NEH/T+XHT+WKPURIDTH8YV++JHACHXUMZTHUIAGI
   OPCTPTFOQDWK3ZSSAF883AEGI+RIT/XWRTNEM4DN9KBW+IJCMXO9RKBUTANRYZ4D
   YV+W3XBBUPM8EOS/HG9DYE+RE+J/WHRV8VKA/W8MQ+89Y8PR9RHJ9QDQVUY3EBTZ
   XJLXZN4WZYSOMHSV3NOD+WU4HPR/UWWOTQS+YEVPJRUFUT84ZU4HTCUF8LE/RWEJ
   3WCY4WULXTB/MLE3EG
   ```


2. Decrypt this ciphertext with version C and the given key.

   II, 5, V, 1, I, III, 3, 7, 9, IV; 53, 65, 32, 5, 42, 56, 38, 42, 29, 38; WXPSY


   ```
   4S8/9ZXBINQY+DO3ZEOYBFXAOYDRG9KCTRD9PFJSIJMU+EIDWJUQZYKKW+S8BV3E
   +FNYWXOMUUBZVSYVSKSJLHLY+ZRDPDMWDFWLIBPOC4CNXME9IIMO+PBAACFS+GU3
   YSSG9NEYPFS8QB/DYFNM9KV+3MYYH3MR4R3B+PUBFDKEVBFOGVZFQ4/8QTRPEMRJ
   OROSCM4+TG4ANYMKAQ9KSTMDRXCXFWFSY/TMXG8V+/U3C+T9H3DHZVHW3JHD343R
   ILJLCR4RYETNSTW9VT9U3BPXDA/NGF
   ```


3. Decrypt this ciphertext with version CA and the given key.

   1, 5, V, III, I, IV, II, 9, 3, 7; 53, 17, 56, 31, 65, 30, 44, 57, 24, 32; PZSXZ


   ```
   9O8+KIVIASQEWPBPARL4VCGI9FW4/YCNZK/RCMSOX+ZVCGDFU4O+/FMDLSU3W/8D
   EWK88CG/G94FD8NRDVLCFXYO/EOPEII3CESZMNNIHU83TMUZ//AIMSBMPYONZPVJ
   9PWTFZDEJIOSTY+PEANV+PGVDKOYK/CMT9HE8+CHH//CKSLQVGJOH+BQDP+BNVAF
   MK8LWVA3DLP/YI9/O93HYYVUC4S9E4KECDAQXURZUOCFCE9SR9WKHACIF3DMNCH3
   X9Z+C/EN44L4GW8
   ```


4. Decrypt this ciphertext with version D and the given key.

   2-5, 7-10, II, 3-6, III, V, 4-8, IV, I, 1-9; 46, 7, 30, 52, 51, 10, 60, 33, 21, 8; OFF


   ```
   R/XJY3AB+MYJ/344VCJHW8GHTF8QFTVAOXCG9MDH8FMR9+YWBQOPYEFV3D94BUQT
   FB8B394FS33DMBC4+KAU93UP+UAQ4QTUMFGFURDIYT+4ISLVKWVKTELTNIME3L/H
   Z8SXJNHOMOQFIITEHWG8ZPD+D4N/TLJ+/FRPO8MI/J4I8TB+URQBPV+4V3STNPFO
   LE/L+WH/3NZ9HK8WPOWXUS9SODIOPR3+SCNZ9H3OM+WJ8CCGA4LBDZU4+38SKATI
   IKE4FK4BOEST9HVQ4VKBGAD3DW8MX/UMTIOS/PUURDBZGEMG+NA/XQC8/MHLGIUR
   OAXSRBZ+RJ3ADO/M/SPRY4CKGODFUASTVE+LAJOMHRVPBVMDX494+/PCSZINQJCM
   EJKDCQT/RCRICUG9XBSXEQO4KUMMKNMX4+PPYV/QPYYPI9LFZ+OMQWSJP9VZBP8I
   ASZ+/9IUZZ+H9+W8MGMQPH4YX4ROK4NR4+SAJLRQXZKQTZCGJCZFN+APELPAL8N8
   IS
   ```


5. Decrypt this ciphertext with version E and the given key.

   III, 9, IV, 5, 1, II, I, 7, 3, V; 56, 32, 59, 59, 11, 27, 29, 10, 52, 41; ON

```
RAZIGAH+ESFJD4G/S8HOWUSQ9ZSKDZOR9WATHMEAV/RBDQDJWYYBRCDUHSOTS
YNVT4/HJWWMIVLDFIPXKOPMZD9CHSRINYSDXYHVBRKUHCRQZQQL/VDOGUBSA+
BHKDWRP8EAL8KJXYA8/KSJD9KRP4HPH9EPE/N9ZLGTRT9GEG+NSNPY39KZVNP
9ABPTDS9GCEEKVFIQW
```

6. The following ciphertext was encrypted with a T52a/b. The wheel assignments and half of the initial wheel positions are known. Brute-force the remaining wheel positions and decrypt the text.

   wheel assignments: A IV, B I, C 6-9, D 8-10, E 2-4, F III, G 1-5, H V, J 3-7, K II
   wheel positions: A 11, B 27, C 4, D 15, E 48

```
+3RAYZQMHNJWFF8N8OFESRUYUTM/4VKONQK8RCTNA4TTMTQ3TYZKMDEFWK3/NNKI
MCICDEK+OWDVHOE9DERPYV3DHS4+LC/MR4SCJBWUISWKMZM8TCOX/E+FJHRCFM8U
XTNTTPJF49QOEZISIJWNHOGIC8CAWOUCD3/YFW84LENYTNI+H9BBOLJBQRQLDAVP
N+A/BCMO4WTJ4PVSWPP4MS8UJVYP3LLPDJNGEBUZ9AT3HRHO/OCQ8KGS+UILYYYY
+CKYCEKGHELMAFYSJTVH/ZWDW8EOHB+IE3YS8LAHSETWP3CJAKYKP9O9M9ZCOR
```

   This exercise is similar to a challenge from MysteryTwister (https://mysterytwister.org), which can be found at https://mysterytwister.org/media/challenges/pdf/mtc3-lasry-14-T52-01-en.pdf.

7. The following ciphertext was encrypted with a T52c. The wheel assignments and half of the initial wheel positions are known, as is the message key. Brute-force the remaining wheel positions and decrypt the text.

   wheel assignments: A 7, B 9, C IV, D II, E V, F 1, G III, H 5, J I, K 3
   wheel positions: A 33, B 52, C 34, D 4, E 9
   message key: ZXYYS

```
G/PBVQSM9YLM+E4HKUXFPN4+GGCVKNRMDBRX3JLKI4+JDEGF+DT4RO9RDS8NWZLY
TZG+LM44V9TVXYMZTYW99LEHT3NVOJNARQ9T9FIO/XKLS8KM98OIRPMM9+GH/WP8
+Q/GEWQU9XXWJDCQ83W/3ZNZM/YM+AFWLY8JXH+YBMMQ4QHX+PERX+FNMSF/F9JZ
C/X8UDXNB+J8CSSMEIYLB4+TMXZPBG+LAOHSPFENHPA3/CZ4I9LSWDFJOBRJEA8C
8FPGROMGWS4SGCD9YLVTJGNG+ABFGXFVI+9KIC/M8EMQBZWSLP99STHU3AYC/VCB
YWUGNO9M/MUFQGZIK9Q99+OTU8HEFDA+MWJD9FQU9+CIBNEUKXXP34POB3CMD4Y/
UDWBIPUNLCFPNYT/AQPOQ9IKI98JKF9DCR+9/OES89MCEYP+SCNEQIR8YHOLYGJM
NJB+XNHV4TV4DLKSS8PTNOM/DJTZESIKWYJOF3U4SGFIO8M4VUYOSPF+IXXZXBW3
RQS38
```

   This exercise is similar to a challenge from MysteryTwister (https://mysterytwister.org), which can be found at https://mysterytwister.org/media/challenges/pdf/mtc3-lasry-15-T52-02-en.pdf.

8. The following ciphertext was encrypted with a T52d. The wheel assignments and half of the initial wheel positions are known, as is the KTF setting. Brute-force the remaining wheel positions and decrypt the text.

   wheel assignments: A II, B I, C 4-6, D 9-10, E V, F 3-7, G 5-8, H III, J 1-2, K IV
   wheel positions: A 1, B 43, C 6, D 62, E 2
   KTF : ON

```
B9JCH4CCDNHCJ83JXHCNG+GM+NLYIBFRNOV9BEWLAB+QDOT3DHJQUHEXHJJ+/HYO
KPZ3SZP9C+X3M98J9JOWB/GZLMEUXVDWYNF8ZC9OVYYCLALAQVS/8V9XVUKDEWMQ
```

```
JVT3O/BXSDRZXCN/GSH3E+DPALBSVCSUV+XP9N9APFJL3LGDGP+B4++SQLUJGFMW
3ZKGIHWRWRN+TQTM9WNGSM93UFB+ZVHOBUBA3YMUJGBAPC9IGYGMJZ+WFXR4E9ZP
POYQ9CTPL9XGOIESCGGX+XRMBYWTVPYQB3R4MYXJFZTGN8TJ/W/UNNY/IUY98YAS
KISW/CPWC/RYAWTKE3VFWG/PSUZ/4V4AR9CMQLFXPJ8MULXB4H8NTVKYU+JXZDT8
ZQR+8CQQRODGJJH/KFTTW/DLQF+FCMLJTTPRUTPLYZQDA+EQ4R/3DCA+IY34LD8H
+EBYSYOSHRSP39
```

This exercise is similar to a challenge from MysteryTwister (https://mysterytwister.org), which can be found at https://mysterytwister.org/media/challenges/pdf/mtc3-lasry-16-T52-03-en.pdf.

9. Below is a ciphertext in Bletchley Baudot notation. It was encrypted with a T52 version A or B. Part of the key is known:

   wheel assignments: A 4-9, B III, C IV, D 3-6, E V, F I, G II, H 1-7, J 2-8, K 5-10

   Use your hill-climbing attack to find the wheel positions and decrypt the text.

```
XOYDGSTFC9EN8QVGMADUIHYXBQARRML+/YDQP8KCR93SF+G8WQ+EQKU/33V8XHIK
HUU9F3YQWXJLJO/BKQ9M+SOETV/FA8L83CSR//CIIW+E8XNNAPC8J+LBQLJ/8UTR
HFZOB3QT/PA+M4V/K3FIQBG8/MLEJ8R/4FZLEOIX9+3LF/F8VVMBP3GQG4NPRTZP
ZMDDKGDWHL3F+LLCBPSBJFC/B8TYB4CZ/NTBHRMJO/JDXAEJUK/UIT8CO4MN+ILK
BNU/NFDMIIJ/XKJYY3YKCN8YIFKVEE9UUFHXI9FNNQ++FTOKFUMYD/B8DHCICAYQ
NJLZESKWQJ+MRY8GRMWWDJESVTVC/PFMMMZFO8J3/ORSAIG9LDJZBOMDEIX3QMRI
83IF+TLYWUELHMAGOSPMMIT3YJ/XTQQYGN4G8HTBNWFGHNBR9JH+FUC3PXFHBAEV
QQ3ITBMOHY9NGWQXPTNKB/GCL3XRSCUR3HR4X8QPP4BTD3VI4SATOJNJIKNAUHF/
FSJZIO+ETSWEMF84M3DMIBESKD8LY88BARJNMBAX8HOYYAFQYBK9GQHXPCGXILRL
34SCJ9REJRKZQRGIYRYRQEK8RPPQUKNKGALWKM+PAOWGJ4NK9GWKPDDYKX3SMCBC
OL99MRTTGWZ/KSZVZGGGQX9TDT+BBW4PABVFFKFIMLMZVQY+F/GHA+YC+8J3YS+M
RL/XPOJA+8OJWRBZ/9SQ/XG8QGLRHAMI/GANR9BQDDGJ3RCXR+9DKE4UMPT8HXAM
SMNSX/FZTOSI33QFA+J+QMVAI4ZX+H3GO9YTLRF/PY3ZURW+DRLGA3UG/+IOHTMG
BYLL9LCOJI3M/QKJFHAQEDT8RVTCLKISTVIGF3XAIBBNMSRCDWBG9FXJDDKWVDTQ
/98VQTQT3N4M8IGYXKN+B3CXEH+EWK8TAKIJU9+3DVOPVZEOIYBARD
```

This exercise is similar to a challenge from MysteryTwister (https://mysterytwister.org), which can be found at https://mysterytwister.org/media/challenges/pdf/mtc3-lasry-17-T52-04-en.pdf.

10. The broken T52a: Imagine a T52 version A that is broken so that its transposition relays stick and the permutation that it uses is set by the initial configuration of the machine and does not change as a text is encrypted. Find a way to break this ciphertext that was produced by the broken machine:

```
/PGU9VJYZJROE3T9398VEE8XJPT+ICVESQ44XJIDDB8EKZCUEBLS+JRZ/T3HGWU+
8NNUDKUQLPOMXUS/AOOSBRWW3W3PKFDUQQQE+S3CZW9FFE4SEOHWBGLQO9XOOHMQ
LMRNLKQDFEORUDL8W/DGIWZVJ98GMUNBQO4MX8B3F+8RD9CCQHUKKI3CNZ+RKVFA
+AGFULBDCOD/GZ/3AXY9/WG+TVM/TLMI3SD9HDN9FWX4DT/W9KLFDCWOPAITUL/4
B+CXQBSMB+3AWASW+ES4JBYQBXKMTDWOF+/+PJYE+MS84FSJ4GW84YLRVOM+4PCZ
D8UAXP8XXFFORMRMJHLVSIMSXQSNCPQFCYGCX+4QF/W9+BDWQW+ZVQRULCRPOZGQ
RE/4EPZRLUWJ8IIK8QU3ZKSKN/H3WSGG98EGT4LDH4Q4BF+JE3I+4C9MZ9SZ9NP3
E+LT88JYWYODU9EH+9CZAWHODMLWV/JWFQPEPBAZBG+O8QYKN4//YKJWTLYI+NTA
S9IS4F+HE+OP8COJ8IDI+8JQ8NYFPX+AZCVM+Q/OPBZ+8FLJWKPVWPTIU3X9D/V8
OI9WO4S/YE43Y93M+MWDXT+ZFBLQJJ44J/+FP9NSQ3W8UJY+FZ4+FL9CO+E88MSQ
+JLVDOJGJTKJ+M8SAIWPOCXAYCLVDBK8O/VG9LC8OG/PDAH+WN/3P8HEY8+KHI/A
```

8Y3NEWBUAVG+UKXHU9FIPPBAIOURYFWUANA+BX3IWOOUKZCPYAHMKTG3OJOIR3L+
BQGLE+XDIRQTC3JEYHZ9RAHW4CM3PJ4DQYSV8XUE3X8MEFMW9NQSNBFMTDS4UOLK
VEKKNXQ8A/IQXHKCFTDSQ/ANREIVIBQA+CK3EQMX4QFEZYZ8ZT8EYSW9/WSQOJRO
Q8PQAX3MSA9TDZBAUWLNRJIX4RBHPK/EJLUVBBV8U3EW9XFUNVS83SCLNP8FAO4Q
QSNDEQW4QNYCERWF8AAYUT+HQ9RVPY3OO49VBW8YTRYAIBKJMPJUAEHPOXQPC/VV
K3TQA/UXPHJHT9GN83+HXTDUHQPVJ84RB+VMZBQ+O9QXAEODXL/QSG/PLEIZTWHM
+NIMJTJ8A3VIRJ8/XFJ98ZRSYJI+4R//ZCAYO83ADNXQZF3EINPIKZG/WM9MARBE
Q+IEYGRW+LL4U+FKZHB9OWAJC9F9+KESKZ89MKHWEWGZ+KHAWDVNENPEIXQERIIY
QQKCZEBJELLEBXWCDORBEMK3WBQZTAPXLGZTBVZRWFYJPEGAD/PDXLHDGKBEMIAD
YHTU/+JDTEGB4MQY+JJHVBCQYMDLCUFLQLNV/LHVNHYW3A++ANMIXZAMRQBANLL+
FMGNQ8VQGOV88SR4G+U9UE+UES/44XY8JXYK3G9FRVNSUU4SEWNSKDVMJLQS4NSE
CKWYK/OV

Hint: ILUVW, FRPPXWH WKH SHUPXWDWLRQ SDVW WKH ARULQJ. WKHQ IRU HDFK GDWD OLQH WUB DOO SRVVLEOH ZKHHOV DQG VWDUWLQJ SRVLWLRQV DQG FKRRVH WKH RQH WKDW PDALPLCHV WKH LQGHA RI FRLQFLGHQFH. BRX PDB KDYH WR GR WZR OLQHV DW RQFH LI WKHUH DUH PRUH WKDQ RQH JRRG FDQGLGDWHV IRU KLJKHVW LRF.

Unit 186
Lorenz SZ40 and SZ42

The *Lorenz SZ40* and *SZ42* formed a family of teleprinter cipher machines used by Germany in World War II. They were so named as an abbreviation of *Schlüssel-Zusatz* (“cipher add-on”). British cryptanalysts nicknamed them *Tunny*, which is their way of saying “tuna fish.” Like the T52, SZ40/42 used wheels with pins to modify the bits of baudot-encoded messages. Unlike T52, the pins of the wheels in SZ40/42 could be added or removed, and only XOR operations, but not permutations, were used. Look back at Table 182.4 for a definition of the XOR operation. A fun property of XOR is that

$$A \oplus A = 0$$

(Equation 183.5) for any bit (0 or 1) A. Therefore, if we use XOR to encrypt, then the same operation is used to decrypt. If we write P for the plaintext, C for the ciphertext, K for the key, and think of “$\oplus$” as acting on all the bits in parallel,

$$C = P \oplus K$$

$$C \oplus K = (P \oplus K) \oplus K = P \oplus (K \oplus K) = P \oplus 0 = P$$

(Look back at Equation 183.6.) This property makes life easier for German cipher clerks. The rest of this unit will be spent on describing the various members of this family of machines.

Figure 186.1: Lorenz SZ40 on display at the U.S. National Cryptologic Museum. Photo from U.S. government.

There was an initial model, the “old SZ40,” that was more of a prototype than anything else. It had a small production run and was used for only a short time. It has ten wheels, and all of them stepped after each letter was enciphered. They have fixed pins. They are grouped into five sets of two, so that the signal from two wheel affects one bit of the baudot data, i.e., each bit is XORed with the output of two wheels:

$$b_i \rightarrow b_i \oplus X_i \oplus Y_i$$

(i = 1, ..., 5). By giving the wheels sizes that are coprime, the period of the machine is lengthened.

The proper SZ40 has twelve wheels with variable pins. Pins can be flipped into place (“active,” denoted by × or 1) or out of place (“inactive,” denoted by · or 0). Like the old version, the proper SZ40 XORs the output of two wheels to each bit of data. However, only five of those ten wheels step after each letter is enciphered. The additional two wheel control how the remaining five wheels step.

Five of the wheels are called *Spaltencäsar* or *chi* (χ) wheels. The output of the first one is XORed into the first bit of the plaintext character, the output of the second into the second bit, etc. Another five of the wheels are called *Springcäsar* or *psi* (ψ) wheels. Again, the output of the first of these is XORed into the first bit of the data, etc., so that a bit is altered thus:

$$b_i \rightarrow b_i \oplus \chi_i \oplus \psi_i$$

The remaining two wheels are called *Vorgelege* or *mu* (μ) or *motor-control* wheels. More about them soon. Each of these twelve wheels has a different number of pins around its rim. See Table 186.1 for those numbers. A pin can be in active or inactive position. Only if a pin is in an active position can it conduct a signal out from the wheel. The wheels have numbers on their rims, like in other machines we have seen, to indicate which pin is being probed at the moment.

Stepping of the wheels occurs after each character is enciphered. All of the chi wheels advance by one position. If the output of μ_2 is a positive signal, then all of the psi wheels advance by one. If the output of μ_1 is a positive signal, then μ_2 steps. Finally, μ_1 steps every time. Got it? Take a look at Figure 186.2 for a sketch of the encryption process.

We need to discuss rules imposed by the creators of the Lorenz machines for the setting of active and inactive pins on the wheels. For each wheel, there is a prescribed number or range for the number of active pins; see Table 186.1. For the chi wheels, there can be no more than five active pins in a row, or five inactive in a row. Furthermore, the number of segments (sets of pins with the same setting all in a row) is prescribed also. For the psi wheels, the number of segments is determined by the number of active pins on μ_2; see Table 186.2. For μ_1 the constraints are that there can be no more than five inactive pins in a row, and no more than fifteen active pins in a row. For μ_2, there can be at most five inactive or six active pins in a row.

wheel	pins	active pins	segments
χ_1	41	20 or 21	20
χ_2	31	15 or 16	16
χ_3	29	14 or 15	14
χ_4	26	13	12 or 14
χ_5	23	11 or 12	12
μ_1	61	30-36, 38-50	
μ_2	37	9-23	
ψ_1	43	22	26-34
ψ_2	47	24	28-38
ψ_3	51	26	32-42
ψ_4	53	27	32-48
ψ_5	59	30	36-48

Table 186.1: Wheel sizes and key restrictions for the Lorenz SZ40 and SZ42. A "segment" is a group of contiguous pins in the same state. The number of allowed segments on chi wheels depends on the number of active pins on μ_2.

active pins on μ_2	allowed segments on psi wheels				
	ψ_1	ψ_2	ψ_3	ψ_4	ψ_5
9	34	38	42	42	48
10	34	36 or 38	40	42	46
11	34	36	40	40	46
12	32	36	38	40	44
13	32	34	38	40	44
14	32	34	38	38	42
15	30	34	36	38	42
16	30	32	36	38	42
17	30	32	34 or 36	36	40
18	28	32	34	36	40
19	28	32	34	36	38
20	28	30	32 or 34	34	38
21	28	30	32	34	38
22	26	30	32	34	38
23	26	28	32	32	36

Table 186.2: Key restrictions on the psi wheels in LZ40/42.

Figure 186.2: Simplified diagram of SZ40.

The key for the machine consists of the pin layout and starting position (counting from 1) of each of the twelve wheels. For example, this is a valid key:

```
χ1:   35   ·····×××··×·××××·××··×··×·×·×××····××××·×
χ2:   22   ××××···××·×·×·××··×··×××····×··
χ3:   17   ··××····××··×·××××·××·×·××···
χ4:    9   ××××·××·×·×·····×···××·××·
χ5:    5   ···×××·×·×··×·×××××·×··

ψ1:   32   ·×·×····××·×··×·×·×××·××··××·××·×·×·××·×··×
ψ2:   41   ·××··×·×·×·×·×··×·····××·×·××××·×··×·×·×××·×·××
ψ3:    3   ×·×··×·×××·×·×·×·×····×·×·××××·××·×·××·×··×·××··×··
ψ4:   10   ·×·×·×××··×××···×·×·×·×···×·×·××·×·××·××·×·××··×··×·×
ψ5:   23   ×·××·×··×·×·×·×·×·×××··×·×·××·××·×··×·×××·×·×···××··×···××·

μ1:   59   ··××···×··×××··××··×·××××·····××··×··××··××·×·×××····×·××××××
μ2:   12   ××××··×····×·×···×··×××·×··×·×···××··
```

Notice that for χ_3 and χ_5 we have the correct numbers of segments, due to wrapping around the wheel.

Suppose we want to encipher a message with the example key above. Start with a plaintext:

```
THIS MESSAGE WAS ENCRYPTED WITH LORENZ SZ40
```

The zeroth step is to encode the message in Baudot, and list the bits for each character:

```
THIS9MESSAGE9WAS9ENCRYPTED9WITH9LORENZ9SZ+RP8
00010011110101110100010011010000000101011100 1
00100000011001100001101000011000101000000111 1
01111101100010011011011000101011000010110001 1
00000100001000000011100001000000011010000110 1
11000100001001000000011100010110110001001101 1
```

The machine is in the state χ = 01111, ψ = 01100, μ = 11. To encipher the first letter, we xor the bits of T with the bits of χ and ψ:

```
T    0 0 0 0 1
χ    0 1 1 1 1
ψ    0 1 1 0 0
---------------
3    0 0 0 1 0
```

Next, the chi wheels advance to 36, 23, 18, 10, 6. Since μ_2 outputs 1, the psi wheels also advance, to 33, 42, 4, 44, 24. Since μ_1 outputs 1, μ_2 advances to 13. Lastly, μ_1 advances to 60. The encipherment of the next letter is

```
H    0 0 1 0 1
χ    1 1 1 0 1
ψ    1 1 0 1 1
---------------
O    0 0 0 1 1
```

Now, all chi wheels advance, as usual, but because μ_2 is at position 13 which outputs 0, the psi wheels do *not* advance. Since μ_1 outputs 1 at its current position, μ_2 advances. Then finally μ_1 may advance. The next letter is enciphered:

I	0 1 1 0 0
χ	1 1 0 1 0
ψ	1 1 0 1 1
P	0 1 1 0 1

The full ciphertext is

3OPOD9NKZK/+/3SVKYRH8OMRONZGHIYSNXGMRFPEFFXDN

In Unit 151 we mentioned that we must never reuse the key in a one-time pad. The keystream of the Lorenz SZ40 ($\chi \oplus \psi$) is like the key of an OTP. It is not truly random, but appears to be. But the problem arises when two messages are enciphered with the same key. Suppose we have two such messages, P_1 and P_2. Encipher them by XORing bit-by-bit with the *keystream K*:

$$C_1 = P_1 \oplus K$$

$$C_2 = P_2 \oplus K$$

But when an adversary adds the two ciphertexts together,

$$C_1 \oplus C_2 = (P_1 \oplus K) \oplus (P_2 \oplus K) = P_1 \oplus P_2 \oplus K \oplus K = P_1 \oplus P_2 \oplus 0 = P_1 \oplus P_2$$

The result does not include the key at all! With only a few such messages, the plaintexts can be disentangled, and the keystream can be recovered. Such messages are called *messages in depth*, and they must be avoided for security.

To combat the problem of lazy cipher clerks reusing keys and producing messages in depth, new models, the SZ42A and SZ42B, were introduced with features to modify the control of the stepping of psi wheels, and some of them were dependent on the plaintext. By using plaintext information in the encryption process, different messages are encrypted to different ciphertexts, even with the same key. There are four options (see Table 186.3), two of which incorporate a bit from the plaintext. These are called *limitations* and they cause the psi wheels to advance even when the output of μ_2 is 0. All of them involve the output of χ_2 from the encipherment of the previous character. This bit is called "chi2 one back" and is notated with an overline. In addition, one or two bits can be incorporated: the output of ψ_1 one character previous ("psi1 one back") and/or the fifth bit of the plaintext character two letters back ("P5 two back") (denoted with a double overline). The one, two, or three bits are XORed together and then inverted (XORed with 1). If the result is 1, then the psi wheels advance. If the result is 0, then the output of μ_2 determines whether they advance. So that you can check your emulation of the machine, here are the encipherments of our example message with each of the four limitations added to the key. With chi2 1 back,

3OPODW9TCT9/E3KHGKH4W/W/LBYVSQMWDWWYW+JLVZVRK

With psi1 1 back,

```
3OPOVN9TCJ9/E3KHS/JY9RBG3IJ/CXNHOJKDES9W8C/GT
```

With P5 two back,

```
3OPOVNWTCT9/E3KHS/JY9RBGLIJ/T3W+TLCYW+JLZCVQI
```

And with psi1 one back P5 two back,

```
3OPODW9UBU89W+NDO89RWXWT3MH8CXNHOJK3MRYMTT/HZ
```

There was a third version of the SZ42. The SZ42C used output of the chi wheels to control the stepping of the psi wheels. The two motor-control wheels were removed. This model never made it to production.

name	notation / limit bit
chi2 one back	$1 \oplus \overline{\chi}_2$
psi1 one back	$1 \oplus \overline{\chi}_2 \oplus \overline{\psi}_1$
P5 two back	$1 \oplus \overline{\chi}_2 \oplus \overline{P}_5$
psi1 one back P5 two back	$1 \oplus \overline{\chi}_2 \oplus \overline{\psi}_1 \oplus \overline{P}_5$

Table 186.3: Limitations on stepping of the psi wheels in Lorenz SZ42.

Reading and references

Orr Huettenhain and Dr. Fricke, OKW/Chi Cryptanalytic Research on Enigma, Hagelin, and Cipher Teleprinter Machines, TICOM report I-45, 1945, https://drive.google.com/file/d/0B7sNVKDp-yiJOWYxZWFmNDgtODUyMS00Y2FiLThkNWItYmQ5N2JmMzEyMzIz

Irving John Good, Donald Michie, and Geoffrey Timms, *General Report on Tunny with Emphasis on Statistical Methods*, 1945. This document was edited, annotated, and given additional material by James A. Reeds, Whitfield Diffie, and J. V. Field; published as *Breaking Teleprinter Ciphers at Bletchley Park: General Report on Tunny with Emphasis on Statistical Methods (1945)*, IEEE and Wiley, 2015, ISBN 9780470465899 and 9781119061601, https://doi.org/10.1002/9781119061601

George Lasry, Solving a Tunny Challenge with Computerized “Testery” Methods, Proceedings of the 3rd International Conference on Historical Cryptology, HistoCrypt 2020, 96-105, https://www.ep.liu.se/ecp/171/013/ecp2020_171_013.pdf

Wolfgang Mache, “Geheimschreiber,” *Cryptologia* 10:4 (1986) 230-242, https://doi.org/10.1080/0161-118691861065

Programming tasks

1. Make a function that checks the validity of a key. This is a project in itself.

2. Implement the Lorenz SZ40.

3. Add the stepping-control features of SZ42A and SZ42B to your implementation.

Exercises

1. Decipher this text that was encrypted with SZ40. Use the key from the SZ40 example above.

```
3OPODXJICJ3BC8YKRYHVICKVS4+XPOBF+44FUYP4TLUJLSUWKQGWZMINNCPD4B9Y
YT8Q4KEVI9Z+9PB+NUOUZF43DPJ+ZNYGW8UL/ZK8EHDJ4Y3ZLF3Z4WSPMENB9SKA
QKSFDZGS4GR9QTGZHX+PNMDALX9PX4NPGDJEVUIP9UVY4A8Q3IAAW3CR3T9XYGYV
DKKOBVP+CLWSRJBV9CORTW/E3+HDERUMEFDQQKYPF3IRWHC3K+VUGCBEH/DPS4HD
QUFYOLTGDAPVJPPEZJ8MA+Z3HRHGVK8GYYV9DQTBETMDGJPGLY4D+XVPUJ
```

2. Decipher this text that was encrypted with SZ42. Use the key from the SZ40 example above, but add the limitation P5 two back.

```
ADY4V+KBRM9/WO+WITARUFPKYWM3GUAPEA8TYVCX9+SLCB9AKFW+XKJ4ATFJ+/9R
U9M+Y9ZH3VR44NECFJYNOBATSYYXCRGMDIWT+PQDZX34VNT+LZT8ZZIEDU4RDHYU
ZJDMZJDPJU9VDBBUKHKD/ROY84ONHYRE9C8+IK+HDHPRWHR/IMJRKOFHMO8A3D//
RIOKCYXDJRJUEG/SCKUBC8Y9I/84B8V9UFKYXN4F9HWUNTBW/9XKWUFZXLWE4994
N/4RTOOWZW9YDGMALQUICFMQYC/ZCXRDLBLBJWTHVA4QQE/LDQ4AL9DOYUHD839C
/I+O9RIGFOKVO/SYEUE83L9TWGNPXCXUZIC4LTPF
```

3. These five messages were encryped by an SZ40 with the same key. See if you can recover (most of) the plaintexts and the keystream.

```
SPBMKROMZCNARFNBOQ9RKOUJPQO+Y4BWB4L4JJQ/DBSDGNNCABTHN3WUJFGNQNK+
TQMLFPVHV+OKNF4NKZIVMFO9YI3RFJWBLNA8RASTWOWR+LSRAKRIRMK/FXHSUGYF
RM33RUAQFKKDPKRRNLHIGHXTVHBKDO+
```

```
WBXUXPD4YSBAOEJYXLQCK4LCNLEJWTOEYN/3USYDHBGO/DATOYLMJ/VMFA8OM9SJ
LPCEMJO4TVEFC8R3BNLWOYNU9QIWSNK9QOKWFIKQP9DFWIEXLK8HTVWCBTROH9Y9
V8LW/H/DGDXDJ9UW
```

```
DCYSJRYXYY+JI4N/JJOPFA9XLJFCTK8UPOUKKJL/9+KFCCAUJFUSKO3+4F/YTJFJ
UL/HT8GMQCRKR3TNEFMWKFD+383WFQ9QUNJRADWONYZ++SUZ8KWJLJYYYQPIAAY8
T43H/D93
```

```
/3OIJPQERNOHE3NGNQOW3S9NRJKY/XTFQRIFDRY3MVTDCJACJFR4EEZGAHWTCDYJ
LAPDFSJBC+DUEKJYE9DZWF/4HOKVSPDNTU8R/QKOWZLC9CCFYIJLQWVZCL9HX+9
```

```
SDKSPG8AFU9/LSJ/BYOXKQFDYJ4IZ4NITNT8L3IS9HVF94DWDSB9Y/LBXOCTRMM8
YXLXSBUHH++KBCRVZYZPT3Q4NQI3/IU/BTUPQWR/ZVF3TILSGKR8IBLNWERIT8QK
RCSPJAA8+Q
```

Unit 188
Attacking Purple with cribs

If we know the settings of the Purple machine at any point in a text, we can run the machine forwards or backwards and find its settings at any other point. So for the purposes of illustrating the method of attacking it with a crib, we may as well look at cases in which we know the plaintext from the beginning of the text; in other words, a known-plaintext attack.

Let's just work an example and explain along the way. Take this ciphertext and plaintext pair from *Riders of the Purple Sage* by Zane Grey:

```
YBUMVFPNMDCXUYRRQNGEQYAQLEWKCLHDZRDQUZSJNYBBWGINJSZEYZODOAVXEOQG
SWFGFDZUQTYHOSBXTOUQXLQBUPCHOSDLEQLYZEJHZBOQQSOKOJGBCXGBNCXOHWJF
KATSEPTUUHOYIDJZDMWZBBNJWKZYSBUXXNIQAIMNZXIWZOQOGJLIUBVEZHVTZDNL
OBETYGONLXUZGXWQFPGQATHFETKNZVCSTAQWNFVZJCRQFIXYMUKPEMTMBIBEXLSR
OTNVZHIVMSYGDLINFKWZHWRECUQBSWREKFCZJSKMWURPJSDXODELJIFMAFNHWLYO
GSCBDZUDSRNSOIOMFQFSDOIXDSVEKGLOTCEWKCNPPYVVMLQJDLNFSJYFBOLFEFDK
RNNWRVXYAPXRAUPVFAMMJSZOHEYMMWWNJZGIRRZHSRWSGEELICRWXNPAHPJKRDUT
PNLCMFFWNYJAOGSJVTCGGOUHNOQMIOOXTNQUTFODYSFJLVZMTRROPPKRTYYJCTZO
UMDIOATKSFSEYY

ITWASTHEMOMENTWHENTHELASTRUDDYRAYSOFTHESUNSETBRIGHTENEDMOMENTARI
LYBEFOREYIELDINGTOTWILIGHTANDFORVENTERSTHEOUTLOOKBEFOREHIMWASINS
OMESENSESIMILARTOAFEELINGOFHISFUTUREANDWITHSEARCHINGEYESHESTUDIE
DTHEBEAUTIFULPURPLEBARRENWASTEOFSAGEHEREWASTHEUNKNOWNANDTHEPERIL
OUSTHEWHOLESCENEIMPRESSEDVENTERSASAWILDAUSTEREANDMIGHTYMANIFESTA
TIONOFNATUREANDASITSOMEHOWREMINDEDHIMOFHISPROSPECTINLIFESOITSUDD
ENLYRESEMBLEDTHEWOMANNEARHIMONLYINHERTHEREWEREGREATERBEAUTYANDPE
RILAMYSTERYMOREUNSOLVABLEANDSOMETHINGNAMELESSTHATNUMBEDHISHEARTA
NDDIMMEDHISEYE
```

The first thing we want to do is identify the sixes and the twenties. For that purpose, we tabulate the ciphertext letters of each plaintext letter.

plaintext letter	ciphertext letters
A	MADOCK
B	GFJYQPNU
C	OD
D	KCOMDA
E	NXQSBEZVGUYTHLFWIPJR
F	QFSBWZUHNY
G	JXBWIQLET
H	PREZSUBYIGNFVJX

I	YNGTSXQWHZJFEPLI
J	
K	OM
L	YSHLIBGPRNXW
M	MCDAOK
N	UNYXBHLJPIETFSWZRVQ
O	DOKCMA
P	XFEWVQU
Q	
R	EHIQZLXJTVRNSGPY
S	VQRJBHFSTUWENGLYXI
T	BFYGLUWZETPHQXVSRI
U	WNQXZTRFHJ
V	EUG
W	URQXNTJPIZSFW
X	
Y	LZWQBFNJY
Z	

Any letter that has more than six images (what mathematicians call the thing you get after applying a mapping, such as the encryption function) is one of the twenties. Furthermore, any letter with an image that is one of the twenties is also one of the twenties. And any letter that is an image of a twenty is also a twenty. The sixes must form their own closed group. We now know the sixes and the twenties, but not in any particular order, and definitely not in the order as mapped by the plugboard.

sixes:	ACDKMO
twenties:	BEFGHIJLNPQRSTUVWXYZ

Next we are going to determine the initial setting of the sixes switch *S*. Start by tabulating the mappings of the sixes and arrange them according to the position in the text, modulo 25 (the number of switch settings). In this table, we take the first position in the text as zero. Right away we notice that for position mod 25 equal to 2, there are two fixed points: M→M and O→O. By studying Table 187.1, we know that this means that at position 2 in the text, the sixes switch can be in position 1, 5, 8, 10, 14, 15, 18, 19, or 22. Position mod 25 equal to 14 also has two fixed points. So positions 2 and 14 (12 apart) can have sixes-switch positions 10 and 22, or 14 and 1, or 18 and 5. Position 16 has a chain of five: A→D→M→O→C. This is either a five-cycle with a fixed point (A→D→M→O→C→A and K→K) or a six-cycle (A→D→M→O→C→K→A). Sixes switch position 24 has a five-cycle plus fixed point, so position 2 in the text is consistent with switch position 10. Switch position 7 has a six-cycle, so position 2 in the text is consistent with switch position 18. Switch position 3 does not have a five- or six-cycle, so we can eliminate switch position 14 for text position 2. Suppose text position 16 is switch position 24. Then its fixed point is K, which must be 5, since the permutation is (3 4 6 1 5 2). Text position 13 has A→K, so A must be 4, because the permutation of switch position 21 is (3 4 6 5 2 1). In text position 14, switch position 22, 4 is a fixed point, so that is consistent. But now suppose that text position 16 is switch position 7. Text position 17 has a cycle of length at least three, but switch position 8 does not. This is not consistent. We conclude that text position 16 has switch position 24. This means that at the start of the text, the sixes switch is in position 8.

position mod 25	mappings of sixes	position mod 25	mappings of sixes
0	C→O A→A D→M	13	A→K D→M→O
1	D→O→A→C	14	A→A D→D O→M
2	A→D→K O→O M→M	15	A→C K→M O→D
3	A→M→O→K D→C	16	A→D→M→O→C
4	D→O M→A	17	D→O→K M→A
5	M→D→C	18	A→K C→D
6	A→D O→O M→K	19	A→O→D→K
7	M→A→O→C D→D	20	A→M D→D
8	M→M A→C D→K	21	M→C D→A
9	O→D→O M→A	22	A→A O→C D→M
10	A→M→C O→O	23	A→O M→K
11	A→O→K M→M	24	M→O→D
12	K→O→M→D A→A		

Now let's see if we can deduce the plugboard mapping for the sixes. We already know that text position 16 has switch position 24, which has a five-cycle and a fixed point, which is 5. From the table above, that fixed point must be K. So we have 5=K. We also saw that in text position 13 with switch position 21, A→K, and the permutation there is (3 4 6 5 2 1), so 4=A. At text position 2 with switch position 10, A→D→K, and the permutation is (5 4 3 1 2 6), so 1=D. Also at that point, O and M are fixed points, which are 3 and 6 (but we don't know which is which yet), so the remaining letter C must be 2. At text position 3 with switch position 11, A→M, and the permutation is (2 1 6 3 4 5), so 3=M, leaving 6=O. We can now conclude that the sixes portion of the plugboard is DCMAKO.

Now that we know the starting position of the sixes switch, we not only know its position at every point in the text, but we also know when the fast twenties switch advances (every point) and when the medium switch advances (when the sixes switch advances from 24 to 25). There are $3! \cdot 25^3 =$ 93,750 possibilities for the initial settings of the twenties switches. This is a manageable number with modern computers. Our strategy is to try them all and find ones that are consistent with the plaintext-ciphertext pair that we have. A good place to start is with fixed points. The chance that a random permutation of twenty items has a fixed point is about 65%, which is rather high. But keep calm and carry on. The fixed points of letters in the twenties in our texts occur at these positions:

position in text	fixed letter	position in text	fixed letter
17	N	385	N
51	E	388	R
68	F	420	R
80	T	426	W
85	L	429	E
131	S	450	L
132	E	480	T
187	T	496	T
279	E	515	I
286	R	522	S
339	S	523	E
347	E	524	Y

The chances that any one position might have a fixed point may be 65%, but the chances that all of these positions have fixed points is about 0.003%, and that positions 51, 132, 279, 347, 429, and 523 all have the *same* fixed point is so much smaller. It looks like we will be able to narrow down the possibilities to one or a few.

We have 93,750 settings for the twenties switches to check, so let's begin with the first one: left L = middle M = right R = 1 and fast = left, medium = middle, slow = right. For these settings and what we know about the sixes switch, here are the overall permutations of the three twenties switches at the positions of interest in the text:

position in text	overall permutation of twenties switches																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
17	12	5	19	1	8	9	18	10	6	4	20	2	13	16	17	11	7	15	14	3
51	5	13	19	16	7	1	20	6	4	18	15	11	14	8	3	10	9	17	12	2
68	2	19	14	16	1	15	17	18	5	9	6	10	20	3	4	12	8	7	13	11
80	4	9	13	19	15	2	5	14	1	20	6	3	7	17	10	11	18	12	8	16
85	18	15	17	14	8	13	7	5	16	12	19	10	20	6	11	2	1	3	9	4
131	2	20	8	3	10	16	13	17	1	5	9	7	4	6	14	18	19	11	12	15
132	10	1	9	20	13	19	7	5	16	11	2	14	18	4	6	8	3	17	15	12
187	6	20	8	14	19	10	7	2	3	12	11	13	15	9	1	17	4	16	5	18
279	7	10	13	8	1	15	6	2	20	11	17	5	19	14	3	18	12	16	4	9

286	5	14	17	18	16	19	11	7	15	9	13	12	4	20	1	10	3	8	6	2
339	14	3	1	2	7	4	9	13	6	11	15	8	10	16	5	17	18	12	20	19
347	10	20	18	19	4	7	8	9	2	6	11	5	15	12	14	1	17	13	16	3
385	9	3	18	1	17	6	14	13	5	8	10	20	19	4	7	11	16	12	2	15
388	20	19	7	11	5	14	4	15	16	3	6	9	10	1	17	12	18	2	8	13
420	9	15	1	3	14	6	8	12	7	11	20	17	5	10	13	18	19	4	16	2
426	8	14	6	12	17	11	19	10	3	16	4	1	20	15	13	9	2	7	5	18
429	11	18	19	4	8	13	17	16	15	2	12	20	9	7	1	6	10	14	3	5
450	10	20	6	7	11	2	19	4	3	16	15	17	5	9	12	13	8	1	18	14
480	13	11	9	16	4	2	12	20	14	8	17	5	1	3	7	6	10	19	15	18
496	4	6	7	15	19	9	1	18	20	8	17	12	5	2	11	3	16	10	14	13
515	13	9	17	10	16	1	7	5	15	2	14	3	11	20	18	12	4	8	6	19
522	17	9	14	6	5	11	15	19	3	2	4	18	1	20	13	12	10	8	16	7
523	5	3	4	9	15	10	18	2	11	8	17	13	12	1	20	14	6	19	7	16
524	1	14	6	10	18	20	9	7	4	16	2	19	11	5	3	17	8	15	13	12

Position 17 has a fixed point (13), but the fixed point at 385 (6) does not match. Nine of these permutations don't even have fixed points. So these settings are inconsistent with our plaintext-ciphertext pair. After searching all 93,750 possibilities, we find four settings that give fixed points at all of the relevant twenty-four positions in the text:

left *L*	middle *M*	right *R*	fast	medium	slow	fixed point(s) at 17	fixed point(s) at 385
20	15	2	middle	left	right	4	6
8	18	25	middle	right	left	7	14
21	4	16	middle	right	left	2	2, 10
22	11	15	right	middle	left	12	2

Only one of these has the same fixed point at positions 17 and 385, which is what we need. So that must be the correct key.

Can we now find the plugboard settings for the twenties? Here are the fixed points for sixes = 8, left = 21, middle = 4, right = 16, fast = middle, medium = right, slow = left:

position in text	overall permutation of twenties switches																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
17	17	2	14	8	1	18	20	11	4	12	15	10	6	5	13	7	19	9	16	3
51	13	6	5	18	3	16	15	11	17	9	12	7	8	19	2	4	1	10	14	20
68	16	6	17	10	14	3	11	8	19	18	5	2	7	4	15	12	9	1	20	13
80	20	15	9	16	2	14	6	12	5	8	11	4	13	7	18	1	17	10	3	19
85	3	18	5	8	20	6	4	12	14	11	2	10	17	1	15	13	7	19	16	9
131	13	1	6	8	10	20	16	4	18	14	11	12	7	9	2	17	3	5	15	19
132	12	7	5	14	11	4	10	17	1	13	3	8	18	15	16	6	9	2	19	20
187	19	14	8	11	13	12	7	20	2	5	3	9	16	15	4	18	17	10	6	1

279	10	15	9	17	2	5	8	11	6	7	16	18	19	14	1	4	3	12	13	20
286	8	19	5	4	12	6	2	20	15	1	13	7	9	11	3	16	10	18	17	14
339	3	15	17	20	14	10	16	9	6	7	11	4	8	5	2	1	19	13	18	12
347	8	10	19	15	16	18	13	7	5	12	6	1	4	9	3	14	2	11	17	20
385	5	2	1	16	13	3	4	19	17	10	8	15	12	7	9	6	18	20	14	11
388	17	6	3	7	12	15	16	9	20	11	14	19	10	8	5	4	2	18	13	1
420	12	8	17	5	1	11	16	15	4	3	20	13	14	7	10	19	9	18	2	6
426	6	17	5	15	11	4	8	7	9	1	2	19	13	18	12	16	3	14	20	10
429	18	11	14	12	15	1	7	13	5	2	17	4	6	8	16	3	19	9	10	20
450	17	8	10	16	1	11	18	12	6	19	3	20	5	9	15	2	13	4	7	14
480	4	7	1	5	20	19	3	13	16	8	6	12	11	2	9	10	17	15	18	14
496	11	10	4	16	8	9	7	13	14	20	6	19	1	18	3	12	17	5	15	2
515	2	19	17	4	9	6	20	7	18	13	15	8	10	11	5	16	3	12	1	14
522	2	20	9	6	12	8	1	17	19	13	11	5	18	10	15	14	4	7	3	16
523	3	10	6	12	17	7	4	9	15	1	16	14	13	2	8	18	19	5	11	20
524	1	7	8	17	5	3	18	4	20	10	15	19	9	16	2	6	11	12	13	14

By comparing to the letters that are enciphered to themselves, we find

twenties letter	twenties wire	plugboard letter
B	1	
C	2	N
D	3	
F	4	
G	5	
H	6	
J	7	
K	8	F
L	9	
M	10	
N	11	S
P	12	
Q	13	
R	14	
S	15	L
T	16	
V	17	T
W	18	R
X	19	
Z	20	E

We cannot place the I, W, and Y just yet, because there are too many choices for each of them. To fill in the rest of this table, we need to locate each of the missing letters in the plaintext, check to see that we already know the mapping of the corresponding ciphertext letter, find the overall twenties permutation

at that point, and see on which line the plaintext letter must be. For example, at position 66 in the text, B is enciphered to F, which we know to be on wire 8. The overall twenties permutation at that point is

(18 19 11 13 4 9 8 15 5 3 1 2 7 10 12 16 20 6 17 14)

Since 8 is in the 7th place in the permutation, B must be on wire 7. We play the same game for the remaining twenties letters.

plaintext letter	ciphertext letter	position in text	overall permutation of twenties switches 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
B	F on 8	66	18	19	11	13	4	9	8	15	5	3	1	2	7	10	12	16	20	6	17	14
G	L on 15	307	18	9	13	12	1	19	11	5	7	17	20	2	8	4	10	15	14	16	6	3
H	R on 18	15	6	9	12	11	17	5	7	16	8	14	10	3	18	13	1	19	2	20	4	15
I	N on 2	47	5	11	13	2	19	12	18	4	10	20	8	15	9	3	17	6	14	7	1	16
J	none																					
P	F on 8	208	12	2	1	18	13	17	10	3	6	16	11	19	9	8	20	14	4	7	5	15
Q	none																					
U	N on 2	40	6	4	17	18	14	16	11	7	3	2	12	19	8	20	13	10	1	15	5	9
V	E on 20	96	3	11	8	5	19	10	16	1	9	18	2	6	13	7	12	17	14	4	20	15
W	R on 18	14	7	11	20	3	10	19	4	2	18	16	6	12	1	17	14	15	13	5	9	8
X	none																					
Y	L on 15	29	19	12	10	11	15	18	20	6	8	4	17	16	2	13	14	5	7	1	9	3
Z	none																					

We can now fill in more of the list. For example, B must be on wire 7, since that one is mapped to 8 at position 66.

twenties letter	twenties wire	plugboard letter
B	1	
C	2	N
D	3	
F	4	I
G	5	Y
H	6	
J	7	B
K	8	F
L	9	W
M	10	U
N	11	S
P	12	
Q	13	H
R	14	P
S	15	L
T	16	G

V	17	T
W	18	R
X	19	V
Z	20	E

The twenties plugboard setting is therefore _N_IY_BFWUS_HPLGTRVE, where the blanks are for letters that do not appear in the plaintext and are therefore irrelevant. We can find them, however, because they appear in the ciphertext.

plaintext letter	ciphertext letter	position in text	overall permutation of twenties switches																			
			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
E on 20	J	367	18	17	12	2	5	19	16	8	14	4	6	10	7	11	1	13	15	20	9	3
E on 20	Q	16	12	17	7	5	6	13	11	9	19	18	2	20	16	14	10	8	4	3	15	1
E on 20	X	11	9	16	19	15	2	13	8	5	14	10	11	6	7	4	18	1	17	20	3	12
E on 20	Z	147	4	15	16	11	8	19	12	14	10	17	1	3	9	2	13	18	5	7	20	6

And now we can fill in the remaining spaces:

twenties letter	twenties wire	plugboard letter
B	1	Q
C	2	N
D	3	J
F	4	I
G	5	Y
H	6	Z
J	7	B
K	8	F
L	9	W
M	10	U
N	11	S
P	12	X
Q	13	H
R	14	P
S	15	L
T	16	G
V	17	T
W	18	R
X	19	V
Z	20	E

We now have the entire key: $S = 8$, $L = 21$, $M = 4$, $R = 16$, fast = M, medium = R, slow = L, sixes = DCMAKO, twenties = QNJIYZBFWUSXHPLGTRVE.

Reading and references

Barjol Lami, Gledis Kallco, Nicholas Guo, and Sean Shi; “Cryptanalysis of Purple, Japanese WWII Cipher Machine,” 2019, https://courses.csail.mit.edu/6.857/2019/project/24-Lami-Kallco-Guo-Shi.pdf

Programming tasks

1. Automate as much of the process as you like.

Exercises

1. Find as much of the key as you can from this pair of texts, taken from two parts of *The Purple Cloud* by M. P. Shiel.

```
ALONETHATSAMEDAYIBEGANMYWAYSOUTHWARDANDFORFIVEDAYSMADEGOODPROGRE
SSONTHEEIGHTHDAYINOTICEDSTRETCHEDRIGHTACROSSTHESOUTHEASTERNHORIZ
ONAREGIONOFPURPLEVAPOURWHICHLURIDLYOBSCUREDTHEFACEOFTHESUNANDDAY
AFTERDAYISAWITSTEADILYBROODINGTHEREBUTWHATITCOULDBEIDIDNOTUNDERS
TANDITSNATUREITSORIGINREMAINSOFCOURSENOTHINGBUTMATTEROFCONJECTUR
EFORITLEAVESNOLIVINGTHINGBEHINDITNORGODKNOWSISTHATOFANYMOMENTNOW
TOUSWHOREMAINTHERUMOURTHATITISASSOCIATEDWITHANODOUROFALMONDSISDE
CLAREDONHIGHAUTHORITYTOBEIMPROBABLEBUTTHEMOROSEPURPLEOFITSIMPEND
INGGLOOMHASBEENATTESTEDBYTARDYFUGITIVESFROMTHEFACEOFITSROLLINGAN
DSMOKYMARCH

KUPCLBXBBVEPQNGYODCKZXWYYSIJYELAYJFSLRGKWJAOUALAYNINUKQOICOCYNZN
ZDIZJNHDYTSXUKVOYNWQWQZXVTKGBDZALRWAQQKAVONZHTZAWRQZSJLZVJHCOHMZ
YXLCNCWYHWDICEIBLFNMIGTPCWAXBCQWNQIYZHZEGJXBTCFZGXYEHXLXXZSCTKJP
XVBXGUVPOVCOMRBGNQSOUONRIPAWZUSUSVVVCHMDFAWFQWXNEEGOLIXSIXHEVURE
QZVLMNUHHKGEQWRVODIHMZGZWRPSCYATYCUFLVOSAYALHLDPHCDUJMLBWZNTSDEV
CGOEIQEVEXECSPJMHPZHGQYDZREAWGUMQNWZUWSDCIMSYKTRHFMFNJPIWOFUAZWM
RYKSWFWJFIEIGBVXEGOYHEDEBAMTPFLBZPVWAQUNYOLSSDOKWCSYNERPWRLRWDRN
AAQRGEWBLPJDCUJEMROLWCMXRWMYSPEFSXDQAVNLZPWUPFAPENPKJWFWJDOOOAEH
PTGXFPMPAQENNBKZVHTDBFVZMJKNHPHTRICYBEEVUYYFZBALLZWHYSJZMUAPXLFE
HNPWGYOSQTJ
```

Unit 187
Purple

The *Purple* machine was a cipher machine used by the Japanese during World War II. It gets its name from the purple binder in which American cryptanalysts kept their notes; its Japanese name was Kyunana-shiki obun injiki (九七式欧文印字機), which means "'97 European-character machine" ('97 denoted the year 2597 in the Japanese calendar). It was also called *Type B Cipher Machine*, as it was the successor to the Red (Type A) machine. The machine is different from the rotor and teleprinter machines: it uses switches to choose permutations, and therefore when a switch steps the new permutation is unrelated to the previous one.

The Japanese language is based on a syllabary, where every syllable ends in a vowel or N. To preserve this structure and so that ciphertexts can be pronounceable by cipher clerks, vowels and consonants are encrypted separately. Japanese uses two copies of the syllabary and thousands of Chinese characters; this makes the decision to use Latin letters seem sensible, and Red and Purple both encrypt messages transcribed into the twenty-six European letters. The vowels, `AEIOUY`, called "sixes" because there are six of them, are passed through one switch (the "sixes switch") that permutes them. The consonants (the "twenties") pass through three switches that permute them. See the tables at the end of this unit for the permutations of each switch, and Figure 187.2 for a diagram of the machine.

Figure 187.1: Largest surviving piece of a Purple machine. Photo from United States Air Force.

The way the machine works is as follows. The operator presses a button on the keyboard for the plaintext letter. This sends a signal on the wire for that letter. If it is one of the sixes, then it enters the sixes switch. Based on the position of that switch, the signal leaves the sixes switch on another line. The permutations for each of the twenty-five positions of the switch are in Table 187.1. If the plaintext letter is one of the twenties, then the signal is permuted by three switches. First is the “left” switch, then the “middle” switch, and finally the “right” switch. The output of the right switch leaves the machine. Each of the three twenties switches has twenty-five positions, and each position gives a different permutation of the twenty letters; see Tables 187.2, 187.3, and 187.4 for the permutations. After each letter is enciphered, the sixes switch advances to the next position. Before operation, the three twenties switches are assigned to the rôles of “fast” switch, “medium” switch, and “slow” switch. Only one of the twenties switches advances after each letter is enciphered. Whichever is the fast switch advances by one after each letter is enciphered, except when one of the other two advances. When the sixes switch advances from 25 to 1, the medium twenties switch advances by one and the fast switch does not advance. When the sixes switch advances from 24 to 25 and the medium switch is at 25, the slow switch advances by one and the others do not. Any switch that advances from position 25 comes around back to position 1.

Purple inherited its split of the alphabet into sixes and twenties from its predecessor, Red. However, the feature becomes moot with the introduction of a plugboard that permutes all twenty-six letters on the way into the machine and inversely on the way out. The full machine (P) is a conjugation of the bare unplugged machine (U) is between the monoalphabetic substitution (S) and its inverse:

$$c = P\,p = S^{-1}\,U\,S\,p$$

If it is not obvious, we tell you that the substitution shuffles which letters are the sixes and which are the twenties.

The key for Purple consists of these things:

- the initial position of the sixes switch (one number)
- the initial positions of the twenties switches (three numbers)
- the choices of which of the twenties switches are slow, medium, fast
- permutation on the plugboard (simetimes split into sixes and twenties)

For example, our key might be

sixes:	21				
left:	4	middle:	18	right:	9
fast:	middle	medium:	right	slow:	left
plugboard:	`TQCLNR PSBYGJZMFOWUDKVHAEXI`				

If we split the plugboard into sixes and twenties this way, then what we mean is the inverse permutation, so that `TQCLNR` → `AEIOUY` and `PSBYGJZMFOWUDKVHAEXI` → `BCDFGHJKLMNPQRSTVWXZ`. We should always try to be clear on exactly what we mean when we give a plugboard permutation for Purple. With this key, let us encipher the text

`THIS MESSAGE ENCRYPTED WITH PURPLE`

The first letter, `T`, is translated to `A` by the plugboard. This is one of the sixes, and from Table 187.1, we see that a signal on line 1 emerges from the sixes switch on line 3, so this letter is enciphered to `I`. The plugboard translates it to `C`. After the first letter is enciphered, the sixes switch advances to 22 and the fast twenties switch (middle) advances to 19. The second plaintext letter, `H`, is mapped to `T` by the plugboard, which is on line 16 entering the left twenties switch. From Table 187.2 we see that it emerges on line 6. The signal enters the middle switch and emerges on line 16. This enters the right switch and emerges on line 19. The plugboard translates this to `X`. The full ciphertext is

`CXEEEKVXMSXBNNRDEQEZEHLAAUCOTE`

Without the plugboard, we would get

`CBURDAWMOCAODJHURHEVBODBLEHFXE`

Figure 187.2: Diagram of the Purple machine.

switch position	**input** 1	2	3	4	5	6
1	2	1	3	5	4	6
2	5	4	2	6	3	1
3	1	5	6	3	2	4
4	4	3	2	1	6	5
5	3	6	1	4	5	2
6	2	1	5	6	4	3
7	5	4	6	3	2	1
8	3	6	1	4	5	2
9	6	3	5	2	1	4
10	5	4	3	1	2	6
11	2	1	6	3	4	5
12	6	5	4	2	1	3
13	2	3	1	5	6	4
14	4	2	5	1	3	6
15	1	3	4	6	5	2
16	5	6	3	2	1	4
17	6	2	4	5	3	1
18	4	1	2	3	5	6
19	1	2	3	6	4	5
20	2	5	1	4	6	3
21	3	4	6	5	2	1
22	1	5	2	4	6	3
23	4	6	5	2	3	1
24	3	4	6	1	5	2
25	6	2	4	3	1	5

Table 187.1: Permutations of the sixes switch of the Purple machine.

switch position	**input**																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	4	7	13	6	17	1	8	11	10	5	16	18	9	3	15	12	20	14	2	19
2	6	17	9	1	2	18	20	10	19	15	12	13	14	5	8	3	4	11	16	7
3	2	19	12	17	20	4	13	15	18	11	6	8	3	14	5	9	1	10	7	16
4	14	9	1	4	13	5	17	7	12	16	15	10	18	2	19	6	11	20	8	3
5	19	16	10	8	6	2	15	3	20	9	18	14	5	13	12	11	17	7	1	4
6	20	1	8	18	19	7	5	12	3	13	2	11	10	4	14	15	16	9	6	17
7	8	10	19	12	11	3	2	17	5	6	13	20	7	16	18	1	9	4	14	15
8	1	20	14	15	7	12	3	13	16	10	17	5	11	6	9	4	18	8	19	2
9	9	14	20	17	12	15	7	4	2	18	3	16	19	8	11	10	1	6	13	5
10	17	13	5	7	10	16	11	2	4	8	20	1	15	9	19	14	3	12	18	6
11	2	5	13	8	16	17	18	9	7	11	4	19	12	15	10	3	6	14	20	1
12	18	4	16	2	1	7	12	11	17	14	19	9	5	10	3	8	13	15	6	20
13	16	6	11	20	17	19	10	8	9	3	7	15	14	12	1	5	2	13	4	18
14	13	11	4	9	12	8	3	5	14	17	1	2	20	18	6	7	19	16	15	10
15	10	3	18	5	9	15	4	6	12	20	11	1	17	16	7	2	14	19	8	13
16	4	17	15	16	18	20	14	1	13	19	6	5	11	8	2	10	7	3	12	9
17	15	13	2	19	3	14	1	20	11	12	10	17	6	9	16	18	5	4	7	8
18	12	7	6	3	19	13	16	18	15	1	9	14	2	4	17	20	8	5	10	11
19	11	12	16	14	15	10	2	19	3	8	13	4	1	7	20	6	18	17	9	5
20	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
21	7	18	12	11	4	20	9	14	1	5	16	3	8	19	10	13	6	15	17	2
22	5	3	17	18	8	11	6	16	13	7	14	15	4	20	2	19	10	1	9	12
23	9	16	19	10	7	14	13	20	8	4	5	11	12	17	18	1	15	2	3	6
24	3	8	10	13	1	9	19	2	6	18	20	7	16	11	4	15	12	17	5	14
25	19	15	7	3	14	12	18	4	5	2	8	6	20	1	13	17	9	16	11	10

Table 187.2: Permutations of the left twenties switch of the Purple machine.

switch position	**input**																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	3	8	7	18	4	15	20	10	2	9	11	13	16	19	1	14	5	12	6	17
2	17	4	20	5	11	2	14	7	6	15	12	1	19	18	3	8	10	16	9	13
3	6	14	18	1	3	17	15	4	16	19	13	7	11	10	8	5	12	2	20	9
4	14	3	10	19	12	1	6	11	17	13	2	8	20	5	9	18	15	7	16	4
5	9	2	4	6	17	13	1	18	5	20	14	10	3	8	16	15	7	11	19	12
6	11	12	14	15	1	19	4	9	6	5	17	16	10	3	20	13	2	18	7	8
7	18	7	3	2	20	16	19	1	8	11	4	9	6	13	14	10	12	17	5	15
8	2	17	11	16	10	20	12	6	18	4	8	19	13	7	5	3	14	9	15	1
9	10	6	17	7	5	12	3	2	1	16	11	20	8	15	4	19	9	14	13	18
10	17	20	12	13	9	10	16	5	11	1	3	2	15	18	19	6	8	4	14	7
11	8	16	20	4	13	7	2	19	14	18	1	11	9	3	12	15	17	5	10	6
12	15	1	2	20	16	14	9	17	3	4	10	12	5	6	7	8	18	13	19	11
13	4	13	6	9	15	11	10	12	8	2	16	17	7	14	3	1	5	19	18	20
14	19	10	5	14	18	9	15	20	7	8	13	4	6	12	17	2	11	3	1	16
15	5	19	17	16	11	8	6	13	12	7	9	3	18	2	10	20	15	1	4	14
16	20	4	2	5	6	18	16	15	9	10	7	14	17	8	13	12	19	11	3	1
17	13	5	1	3	8	2	11	16	15	17	18	6	19	4	14	7	20	9	12	10
18	7	12	19	11	1	14	13	18	4	5	15	10	9	16	2	17	6	20	8	3
19	12	9	15	8	5	16	17	7	13	18	20	19	3	1	10	11	4	6	14	2
20	3	18	10	17	7	4	19	1	20	11	2	13	12	6	15	16	9	8	5	14
21	4	15	11	18	19	3	8	17	10	14	12	5	13	20	6	9	1	16	2	7
22	1	20	13	12	2	5	10	8	9	15	6	3	14	7	19	18	16	17	11	4
23	16	14	9	10	8	6	18	2	19	3	5	20	4	17	11	1	13	15	7	12
24	8	9	16	20	14	12	7	3	13	6	19	15	2	11	18	4	17	10	1	5
25	2	11	8	6	10	20	5	14	16	12	15	18	1	9	4	7	3	13	17	19

Table 187.3: Permutations of the middle twenties switch of the Purple machine.

switch position	**input** 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	6	20	4	15	17	8	1	13	14	7	3	10	12	18	19	9	11	16	2	5
2	9	4	8	12	20	18	14	7	11	13	15	5	6	3	1	17	2	19	10	16
3	5	1	14	7	19	11	15	18	9	8	2	4	13	10	12	16	20	17	6	3
4	16	10	2	5	11	7	20	12	4	15	14	3	19	13	17	1	18	9	8	6
5	18	12	6	4	15	9	13	11	5	14	20	1	8	17	7	3	19	2	16	10
6	19	13	18	17	3	4	6	5	2	12	15	7	1	9	16	20	8	10	14	11
7	11	3	13	1	8	15	2	9	18	6	19	12	14	16	4	10	5	20	7	17
8	12	17	20	3	9	2	19	7	1	4	18	10	15	8	14	6	11	5	16	13
9	13	14	11	16	1	12	9	10	17	18	7	8	5	2	6	19	4	3	20	15
10	10	13	5	14	7	16	18	17	3	9	1	6	19	11	8	2	12	4	15	20
11	7	9	3	18	6	20	16	2	19	10	8	11	17	5	4	12	15	13	1	14
12	1	8	15	10	11	13	6	16	5	20	12	2	4	7	9	14	3	17	18	19
13	5	7	1	2	19	14	12	8	16	3	20	4	10	9	15	13	17	6	11	18
14	17	12	20	6	4	3	11	19	1	5	2	13	9	15	10	18	7	14	8	16
15	15	18	10	13	17	19	8	1	12	9	16	6	2	3	11	4	14	20	5	7
16	16	5	19	4	14	9	18	17	15	20	10	8	7	13	3	2	6	1	12	11
17	2	15	16	11	13	5	3	20	10	17	14	9	6	1	18	7	12	8	19	4
18	3	6	9	8	12	17	5	10	16	11	4	14	18	20	13	15	1	7	2	19
19	8	3	12	20	18	6	7	14	13	1	5	19	11	4	2	9	16	15	17	10
20	20	7	14	9	8	4	10	3	2	16	6	5	17	12	15	11	13	19	18	1
21	13	19	17	12	16	10	15	4	18	6	1	20	3	8	11	5	14	7	9	2
22	9	11	10	5	2	14	17	15	20	3	13	18	16	19	7	1	8	6	4	12
23	7	16	19	10	5	1	13	18	6	2	11	17	15	14	20	4	9	12	3	8
24	4	18	15	17	3	12	2	1	7	19	9	16	20	6	5	8	10	11	13	14
25	14	2	7	19	10	13	4	6	8	12	17	15	1	5	16	11	3	18	20	9

Table 187.4: Permutations of the right twenties switch of the Purple machine.

Reading and references

Mark Stamp and Richard M. Low, *Applied Cryptanalysis: Breaking Ciphers in the Real World*, Hoboken: Wiley, 2007, section 2.3.

William F. Friedman, Preliminary Historical Report on the Solution of the “B” Machine, https://cryptocellar.org/files/purple-history.pdf

Barjol Lami, Gledis Kallco, Nicholas Guo, and Sean Shi; “Cryptanalysis of Purple, Japanese WWII Cipher Machine,” 2019, https://courses.csail.mit.edu/6.857/2019/project/24-Lami-Kallco-Guo-Shi.pdf

Wes Freeman, Geoff Sullivan, and Frode Weierud, “Purple Revealed: Simulation and Computer-Aided Cryptanalysis of Angooki Taipu B,” *Cryptologia* 27:1 (2003) 1-43, DOI: 10.1080/0161-110391891739

Kenneth J. Bures, “Cracking PURPLE: cryptanalysis of the Angooki Taipu B switch tables,” *Cryptologia* 45:1 (2021) 1-43, DOI: 10.1080/01611194.2019.1706064

Kenneth J. Bures, “Cracking PURPLE: the identification of homologs in the cryptanalysis of the Angooki Taipu B cipher machine,” *Cryptologia* 47:5 (2023) 436-448, DOI: 10.1080/01611194.2022.2064200

Wikipedia, “Type B Cipher Machine,” https://en.wikipedia.org/wiki/Type_B_Cipher_Machine

David Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Simon & Schuster, 1967, pages 18-24 and elsewhere, https://ia600606.us.archive.org/30/items/B-001-001-264/B-001-001-264.pdf, https://archive.org/details/B-001-001-264

Programming tasks

1. Simulate the Purple machine cipher. Check that it can reproduce the examples in this unit.

Exercises

1. Encipher this text from with the example key given in this unit (with the plugboard).

   > The flowers were white, with streaks of golden orange upon the petals; the heavy labellum was coiled into an intricate projection, and a wonderful bluish purple mingled there with the gold. He could see at once that the genus was altogether a new one. And the insufferable scent! How hot the place was! The blossoms swam before his eyes.
   >
   > (from “The Strange Orchid” by H. G. Wells)

2. Decipher this ciphertext with the example key given in this unit (with the plugboard).

```
DBVCUQNJMCHFYRNSOLJFPTLIRTUIAHWXTNPVIVFMSVSCESFRBSHVCCGHNWSQEPYL
ORQUSFUFSXLPPTZMRFYRVQEALHZTJNMGIKLDWUHZPJHQECYCLFFLTDTITRGBTACC
ANWKGLODVLSVPCXFPTLIKWTGYQELUTSDKZMQZGARYDCSPJYFTOSIYTKAVLRWKXZP
SJDENHMFBDHZRUCUDSRUGLWJCYUOWRSTCPVJJTZZOYACBYTGDTABLQPNPUQKLSBQ
KMTJYCJVONEJEDXXNKAJTEZPQARNSQRMXSITWLWQNJBCALDCJEXTTPEIXZVBTXLE
YARMGXJVCYOXLVPQ
```

Unit 189
Attacking Purple with statistics and hill-climbing

In this unit we discuss how the use of statistics can help in analyzing a ciphertext that was encrypted with the Purple machine.

The weakest part of the Purple cipher machine is the division of the letters into sixes and twenties, and the handling of the sixes is the weaker of the two. Now, because of the split, the letters assigned to the sixes share their frequencies among themselves, while the twenties share their frequencies together. As an extreme example, suppose that we have chosen the six most frequent letters in English (A, E, H, N, O, T) as the sixes. When we encipher a typical piece of text with over 500 letters, we see the following distribution in the ciphertext. The sixes are clearly distinguished.

If instead we choose infrequent letters for the sixes, such as B, J, K, Q, X, Z, the distribution of letters in the ciphertext looks like this:

Here, the sixes are again easily distinguished by their low frequencies. Unfortunately, if the plugboard settings are chosen at random, the effect is washed out and it can be impossible to discriminate. But with a long enough text, it sometimes becomes possible again.

Here is a somewhat lengthy ciphertext with an almost randomly generated key (one—yes, just one—letter was changed in the selection of the sixes, and it makes enough of a difference). Let's see what we can do with it.

```
EFUZBKBWJVOVTVZWAQFRUFZBKXINIXIDOALXIZSUKRZHLMUFIIXWVNRVDGPNZEIK
MOXMKYUBRMRJWVYEFXRMSGTSEBKSOXAXVKEYEERLOOBEJCUEIMXNPMWFLEJBMRRQ
DPNSBZOXAGYZYQDHPBTDVWCORNMTHXCZMSOIPTDEPRAWFMMRJKNSZIKSMYARLFPD
DVRGCHNBODNUMUFPDDGCNTMMQWHGYQLUIFKRKTSYHMPMGPXQKDOGEYLXUUTGWPED
WOWVBBPOMTKCTMMUOGDEBLVPREPMISKPGCRLCWTGFPIPWZYIXMYIDJEYHEGYCJLG
EEGRYAQTQEDSCYJYNTHYEYOJZDDDYTSIUZWOQZENALNXCRHQPWKKHEIPPCNMRNEQ
IIEYKAQDPAEVRUKGNNMJYRMVSMBFQQPRYIRGWZLJOXKNAZGOTSXDZSMMCRIMPUNX
KZAAKCMMYVFPZUREMQRGFODSJCBLZRYHUGEADMFMZUDATEHSQXZRXZDDYMAEJMMN
TYICTIFDDAEHMHLEIOLDHXWKDZOUTPRBZVRRYFAKUCMODRDHFZLPMSTQMDNRREOJ
INOURWYKVCEOEXQMJBENTTSSQJCIOMMEPNPIVCMYRMSMQRREFSIFJURVVGFGEPCR
RNYZKLDAUMFILJTZDZDCQBYGTLVFXLNODYZYRHPADJZZUSFRKXZJBNCDFSYMZVJM
WIFTDUMJDFOFHKLAVRBAPXIIFRROBJQNQRTXWWSOIGNYEXXHGLNBNUZPYFVKJEUC
EZQOOZDUERVEVTQSIBVMXTUGIXAOFNEQEJPPXRPKDNNEDCRSMMBHGWHEMFDZISKJ
RTTXNIISXANQTLRTGKTWVULRNFVEKEIVSDDDUQMWXYILFLAKRIEXLVEINXPPCCOJ
YQCFJFUORLHEGKZQQAVOIFKNTZDXRKEBMAAXYNSNFGVHFIHLQXEFGBCDEIOUXPNI
EAIKEWBIYQMFOIOSRUPWLAIVASSUAMEXOMQHSASMPCNIXPZKVIXHRDXGGVIRWJGE
RNFRPAVDCUHTEHIGVBCJJLRJZKXIWLRVOGNLIEBZMNGNKMMWLZHBDEILWSZDGXBC
IGHSCMRCNFPYGFEXMEXREHLTGMGKUKEVYSDMMAILSOMBSPWFJRZVWBIZDDVESMST
ETMDPLAOIILAYUBFANTRYZNXIIHLZXERNCVRBXVCIRHFAUMIJUVQGRIEFDTVSINQ
ISUCOKDHYZWCDWHBEPXBURMHRFYMDYPBPMDDSMCVSINCQWUHXOFDMEYOCPJCVMQT
GXXDHIZQBXZVHWGQRJBAYDZMENEBDYNRGXIDBVQCZXMVFJLBFOSXUZLXRMHUPDII
YUWDGHDIMIRZLNDDFBHZMMDKXHZAXLTEJQGNWXUQQBKXUKSDLGDEEDXGOXRPFMWB
LBZEIATERIBYKVUORUHEIRZHYKTGEIXBVRIEQPZYWFBWRJHBXQEVAQYFXYNQTMMA
WMGMWWIZUZEPJVZZBQHDGGAMXZKXOPWFGGXZMXPXSPQLDIFIIBDURYFMOMUDXNCR
LNWQADQGWNHDMGTIRFKRIXDPIJKZXWNFJYWCVEUEAGYOSDCQFTXROZMURZNMXDXB
CLHBUDUGPKSWRBPEVNMRNIHMMDDSMHBJIBCX
```

Indeed, we can pick out the sixes:

The letters in the sixes have a wide range of frequencies in English, from seldom-used X to most-used E. This variation may help us in the next stage—finding the setting of the sixes switch.

The sixes switch advances by one for every letter that is enciphered, so let us bin the sixes letters in the ciphertext into twenty-five buckets, each for one position of the switch.

position in text (modulo 25)	ciphertext letters
0	DDDEEEEEEMRRXXXXXXXX
1	DDDDDDDEEEIIIIIIIIIIIRRRX
2	DEEEEEEIIMMMMRRRRRRRRR
3	DDDIIIIIIIIIIIMMMMMX
4	EEEEEMMMMMMMMRRRRRRXXXX
5	DDDDDEIIMMMMMMRRRRRRRRRRR
6	DDDDDDDDEEIIIIMMXXXXXXXXXX
7	EEEEEEEEEEEIIIIRRRRXXXX
8	DDDDDDDDDDDEEEMMMMRXX
9	DDDDEIIIIIMMMXXX
10	DEEIIIIIMMMMMRXXXXXXX
11	DEEEEEEEEEEEEIIIRRXX
12	DDIIIIIIIIMMMRRRRRRRRRX
13	DEEEEIIIIIMMMMXXXXX
14	EIIMMMMMMMMMMRRXXXXXXXXXXX
15	DDDEEEEEMMMRRRRRX
16	DDDDDDDDDDDIIIMMRRRRRRXX
17	DDDDEIIIIIIIIIIIMMMMRR
18	EEEEIIMMMMMMMRRXXXX
19	DDDDDEEEEIIMMMMRRRRRR
20	DDDDDEEIMMMMMMMMXX
21	DDEEEEIRRRRRRRRRXXX
22	DDDDDDDDDDDEEMRRRRXXXXX
23	EEEEEEEEEIIIMMMMMMMRXXX
24	DDIIIIIMMMMMMMMMMMRRRRRRXX

Notice that some letters are missing, such as I in bin 0. It is likely that I is the encipherment of X by the sixes switch for that bin, and since X is very infrequent, I is missing in that bin. But we can be more general than that, and consider all the letters of the sixes. Convert the above lists of letters into frequencies in each bin:

position in text (modulo 25)	frequencies					
	D	E	I	M	R	X
0	0.15	0.3	0	0.05	0.1	0.4
1	0.28	0.12	0.44	0	0.12	0.04
2	0.0455	0.2727	0.0909	0.1818	0.4091	0
3	0.15	0	0.55	0.25	0	0.05

4	0	0.2174	0	0.3478	0.2609	0.1739
5	0.2	0.04	0.08	0.24	0.44	0
6	0.3077	0.0769	0.1538	0.0769	0	0.3846
7	0	0.4783	0.1739	0	0.1739	0.1739
8	0.5238	0.1429	0	0.1905	0.0476	0.0952
9	0.25	0.0625	0.3125	0.1875	0	0.1875
10	0.0476	0.0952	0.2381	0.2381	0.0476	0.3333
11	0.05	0.6	0.15	0	0.1	0.1
12	0.0833	0	0.3333	0.125	0.4167	0.0417
13	0.0526	0.2105	0.2632	0.2105	0	0.2632
14	0	0.0385	0.0769	0.3846	0.0769	0.4231
15	0.1765	0.2941	0	0.1765	0.2941	0.0588
16	0.4583	0	0.125	0.0833	0.25	0.0833
17	0.1818	0.0455	0.5	0.1818	0.0909	0
18	0	0.2105	0.1053	0.3684	0.1053	0.2105
19	0.2381	0.1905	0.0952	0.1905	0.2857	0
20	0.2778	0.1111	0.0556	0.4444	0	0.1111
21	0.1053	0.2105	0.0526	0	0.4737	0.1579
22	0.4783	0.0870	0	0.0435	0.1739	0.2174
23	0	0.3913	0.1304	0.3043	0.0435	0.1304
24	0.0769	0	0.1923	0.4231	0.2308	0.0769

The frequencies of the sixes letters in American English are

D	0.03961
E	0.12497
I	0.07303
M	0.02541
R	0.06136
X	0.00198

The action of the cipher for each bin is a simple substitution on six letters. That substitution is determined by the portion of the plugboard that sets the sixes and by the position of the sixes switch. If the initial position of the sixes switch is s, and the permutation by the plugboard is π, then the true substitution for bin n is

$$S(n) \;=\; P^{-1}(\pi)\, S_6(s+n)\, P(\pi) \qquad (189.1)$$

where $s+n$ is evaluated modulo 25. There are 6! = 720 possible permutations and 25 possible initial positions of the sixes switch, so there are 18,000 configurations that we have to check for the portion of the Purple machine that handles the sixes. Because we have computers, this is not a large number. We can score a pair (π, s) for bin n by shuffling the English frequencies in a way determined by Equation 189.1 and comparing the result to the frequencies observed in that bin. The comparison can be made by looking at the cosine of the angle between the sets of frequencies in a six-dimensional vector space (see Unit 9). For example, let's score the permutation π = (2 1 3 5 6 4) and switch setting 7. When a D enters the machine, it is mapped to the second line to the sixes switch. From Table 187.1, we see that

for setting 7 it emerges on line 4, which is mapped by π^{-1} to `R`. Likewise, `E` is enciphered to `X`, `I` to `M`, `M` to `E`, `R` to `I`, and `X` to `D`. The vector of frequencies is shuffled to

$$(0.00198, 0.02541, 0.06136, 0.07303, 0.03961, 0.12497)$$

In terms of permutations,

$$\begin{aligned}(2\ 1\ 3\ 5\ 6\ 4)^{-1}\,(5\ 4\ 6\ 3\ 2\ 1)\,(2\ 1\ 3\ 5\ 6\ 4) &= (2\ 1\ 3\ 6\ 4\ 5)\,(5\ 4\ 6\ 3\ 2\ 1)\,(2\ 1\ 3\ 5\ 6\ 4)\\ &= (6\ 4\ 5\ 3\ 1\ 2)\end{aligned}$$

so the first number in the vector is was originally last in the list of English frequencies, the second was originally fourth, etc. Take the scalar product of that vector with this vector from bin 0 in the table:

$$(0.15, 0.3, 0, 0.05, 0.1, 0.4)$$

Divide the result by the product of the lengths of the two vectors to get the score: 0.7478. The score for a pair (π, s) can be taken as the average of the scores of the twenty-five bins. A score close to 1 is considered a good match to the data extracted from the ciphertext, and may mean that we have found the true configuration of the machine so far as the sixes are concerned. We generated a score for every (π, s), and here is an histogram of the results:

It is difficult to see the winning score, so here is an enlargement of the tail of the distribution:

The winner is π = `MRIXDE` and s = 3, with a score of 0.9484. The average is 0.6168, and the standard deviation is 0.0358, which puts our winner more than nine standard deviations above the mean. This gives us a high confidence in the correctness of the result.

At this point, if we want to, we can decrypt all the sixes in the ciphertext. We might even see the hints of words in there.

```
I..................I.....ER.ERMI...ED....R...E..IDE...D.I....ER.E.
DI....RRI....I.EDR....R....R.M..E.IDE....E...ERDI..I...I..EDI.I...
...E......R....E....E.E..R..I..I..ED.I...RED.....R..D..E...ER.I...
...E..E...ER....ER........D..E.....R.I..E..E..D..I......ID........
E....ME...MI....EM.RE.....I.......I....ERI.IE.D..R......RI.M.....I
D.........E....IDE...R......I....E.I.......ED....EE.E.ERE....E..D.
E.....I..ED..D.....I.ED......E........IX..DE.REE...I......EI......
ERE.R...I......E....E.RI.M..D..I...I.IE.DI.I.E.DI...D..E.RI.M.E..R
E..I.D..I.....M...ER......E.IDE.....E...ER.ERE..R...E.....E.RI.M..
D........E.IDE...D..E.RI.M.ERE..R...E.....E..RI.....D..E.R....I.E.
........R...E...I...D......E.I.....I...E...R.M..E.I.D........E...E
RM.DE......E.I....E...ERE..............E..I.....M.DE.I....E..ME...
ER....E.I...RE..E..ED.R.MI.....EE.E.E...D..E.RI.M.....E........E..
.M.ER..DER..E.I.D.....ERED..ER.I...............D..E........I.....ED
I.D.R..E.........I...MI....ERE..E..ED.R.M..E..E..I..I.....I......E
ED.E.....E...ER....EE.EMI...MI...EI..E...I....E.I.......E...ER..D.
....RE..E....ME.....ERE....E.E..I....EI.......RDEREDI....D....I...
ERE.R...I......E....E.RI.M.E..R.ED....RD.........E...ERM...EEM...E
.I..ED....RD.....ERE.R...I..I.....E.....I...E.I..ED.I..ER....ERE.R
...I......I..RED.......I...ERE.R...I......E....E.RI.M.E..R.EDD....
.RD........E...ERM...EEM...E..RRIED...ER....ERE.R...I..I.....E....
.I...E..RRIED..ME..I.....ER..ERE......I..RED......ERE..REI........
E...E.I.....I....ME..R.M..E...E........E...ER..R......E.RI.M....EE
.ED.E.I..I.E.IR..M.....E.....ER..RE..ERRE.R...I........E.I.....I..
..ME..R.M..ERED......D......E..E..EI.M.RERE.R...I..E
```

Now things get harder. The sixes switch had only one choice for its configuration, and the portion of the plugboard relevant to the sixes was small. But the twenties switches have $3! \cdot 25^3 =$ 93,750 possible configurations. And there are 20! (a big number) possible ways to configure that part of the plugboard. So we are going to resort to hill-climbing. The algorithm is almost identical to the one in Unit 36, and we will use tetragram fitness as the function to maximize. The differences are that we are only varying twenty letters of the key, that we have to try climbing as many as 93,750 hills, and that 1% of the time we are going to allow the routine to take the step even if the fitness is as much as 5% lower (we introduced this idea in Unit 90 when attacking a transposition cipher). The implementation is very fast, but checking all 93,750 configurations is slow, but we eventually find success when the configuration is

sixes = 3, left = 24, middle = 7, right = 12
fast = middle, medium = left, slow = right

The hill is climbed and at the peak is the twenties alphabet: `AQJYLZCKSPOHGBVNUWTF`. The full key for our example is

sixes = 3, left = 24, middle = 7, right = 12
fast = middle, medium = left, slow = right
plugboard: `MRIXDE AQJYLZCKSPOHGBVNUWTF`

The plaintext is from *Opticks, or, a Treatise of the Reflections, Refractions, Inflections, and Colours of Light* by Isaac Newton:

> I took a black oblong stiff Paper terminated by Parallel Sides, and with a Perpendicular right Line drawn cross from one Side to the other, distinguished it into two equal Parts. One of these parts I painted with a red colour and the other with a blue. The Paper was very black, and the Colours intense and thickly laid on, that the Phænomenon might be more conspicuous. This Paper I view'd through a Prism of solid Glass, whose two Sides through which the Light passed to the Eye were plane and well polished, and contained an Angle of about sixty degrees; which Angle I call the refracting Angle of the Prism. And whilst I view'd it, I held it and the Prism before a Window in such manner that the Sides of the Paper were parallel to the Prism, and both those Sides and the Prism were parallel to the Horizon, and the cross Line was also parallel to it: and that the Light which fell from the Window upon the Paper made an Angle with the Paper, equal to that Angle which was made with the same Paper by the Light reflected from it to the Eye. Beyond the Prism was the Wall of the Chamber under the Window covered over with black Cloth, and the Cloth was involved in Darkness that no Light might be reflected from thence, which in passing by the Edges of the Paper to the Eye, might mingle itself with the Light of the Paper, and obscure the Phænomenon thereof. These things being thus ordered, I found that if the refracting Angle of the Prism be turned upwards, so that the Paper may seem to be lifted upwards by the Refraction, its blue half will be lifted higher by the Refraction than its red half. But if the refracting Angle of the Prism be turned downward, so that the Paper may seem to be carried lower by the Refraction, its blue half will be carried something lower thereby than its red half. Wherefore in both Cases the Light which comes from the blue half of the Paper through the Prism to the Eye, does in like Circumstances suffer a greater Refraction than the Light which comes from the red half, and by consequence is more refrangible.

The last thing we should mention is that if we cannot pick out the sixes at the beginning of our analysis, then we need to hill-climb the entire plugboard. This does not add much to the effort.

Reading and references

Wes Freeman, Geoff Sullivan, and Frode Weierud, "Purple Revealed: Simulation and Computer-Aided Cryptanalysis of Angooki Taipu B," *Cryptologia* 27:1 (2003) 1-43, DOI: 10.1080/0161-110391891739

Programming tasks

1. Write a script or program that tests the 18,000 possible configurations for the sixes and finds that portion of the plugboard and the initial setting of the sixes switch. This assumes that we have been able to identify the sixes.

2. Write a script or program to perform the hill-climbing attack on the portion of the plugboard relevant to the twenties. Use tetragram fitness as the function to be maximized. Remember that this attack is performed for each configuration of the twenties switches (initial settings and speeds).

3. Write a script or program to perform the hill-climbing attack on the entire plugboard, for the cases in which we are unable to clearly identify the sixes.

Exercises

1. Give it a try.

```
WJQOITWTVPHJCNTYVTLZFHJQBTTQUWYFTHEQPXQIYWWCYWESSUEQSKLROUOCNONY
XFEADSNFJSLHTPYTTGYNICGQCXJRTLIAXSANSKQNJRUGYHGPECHMDXDUTRSLECHA
HNDRYYCHGJKYITMNQCAQRXCJWQLYXJAXCECSWNRCNOGVCSYDQLSKWXWAZAILWYMP
PRJALPTYOLUORCDLDUOEXRRCCPFCARHQPOCQIPXDEPSGYJZDADNONSDDNCSPXHDW
VWCCUIEJWWIODYJVTXCJASTIVPAPAHCRHSJJODJGRUSWXUSNJEHVNSCWTWLJTEUG
NXXUWREQBYHCPNPOICTLCAURQJXIATLSZUSAFJNWDBWLOEEPXYPHUYEJESDXPXWP
LICTZRNCRQNESVAPOHVXTIIXTQWLGYQLIUDZIKDJGCFYYCOYNADRXYELSSARWGNI
NNBTDPNTIIETGCYHIHIRPPYEDAUATHOPMSPMIJRLIRTDCHEIVXWPHATTVYXNJHDQ
EQHIPCOHGDNSNJXDBUKLTPNMHYLWOSOUOCPRGWOQHIRHCLCUDDETTPWLVNPHPJSD
HOJAGQRHRMOLCBPDONQAAHNUUTCWDWJMRCPWEJVJCJDRCHLGIQQKGRJDLLTDNVYT
WXEAWFLDNIPDWYKWYSEPVHJSGSYTPRWPEROYRTWLXDRIOJSIJGSASRCXGEXYCWIA
HJFYXGLLGYWOJQGOVQQAJAXSOIGNAGRZIAENFVWJNZTAENYPYCLCXXPIAPLQHMOB
NRWPPKTXMAKGROETCGIIDCBTEJHEOGUFRHTGSMWYEOLPYTUMPEAEFEYRGWPTUYAL
XBLOQLFARPHUYIGIHGTJAWGONJGLIJNATOMSQWSSRXOCWASEJRTXFNKFXIIXUDDN
XITBDXPPOGYYHJXPHUXJWWAUUGTMCREQHEUZIINQJCDHPNUZAXTXXIUPGVXPNDAE
DUUENGWJDGUGTAOCJHGMSECAANXTIECACCSDYPISASEDGILUOPEWGGCYJQFDHEJI
FCDSIALCOQISSUCPIDXTHINLIXYULWZQQYRERITANWNGOWTWNTRSNSTCPQLIOVHH
CZQUORJKXLDDSYNLINPYQJCDWDEHPTPAWSEANLAJYCKKOINBAUZEXHWOFCEYLLJX
SQQDULBRSDPCNWLUTAOXRWIHHTDUSHLISWFXPPQHENDCPPGUTRLEVTLAZZHQONTA
ASAGINYQYALHTPQIDLOSCOGPAVLIFWIQYWLDCSXRQXAXOGRSOJHGCINKEOSYYHUC
TSRCNXERESNXTUAPALGPEECWJXLWKPICAMDQQNPCHTAILVNLVSAWGUCRPOLSYSZG
LWUYUWLMIIHNNUSCJLXAUWJWPNALIMIIUAQRDFQOHYOSULECUPBNAGJAYNODSEXQ
PPTRPMGJYZJESSQRTQPQUOJSNQTCYCYGOINVIPAIPITPYPRGAZPANLAPQEPSCHDE
UNFQRSXIACHPOTWLQPQHPWANIDSHAAXUWJUCAAUTPEDDBADNHPKHWIGIWENXDOSO
IEJLIRIRIEAEEQQTRHIUFWHRICHESDSUTDHXVKOGDGCRNHUJWCVAQLOITCYGWJAP
AICDVRYTOTURAPNGHAGDRWYJVPABDEJHAHYOCITNRGQGQIQRRLNTDVRUAYEWELKZ
OJUXOQPPNXSXWGJFICMUXLYVDCPSXHTYOYGOHMPNXEDORYGGIX
```

2. Now another:

```
AAVHHBJWBYVYYJPWYYQSKAYALUDDTFBCHNLROWGHYYTTBMJIGHMXFHAOMRLWEYPP
LEJWAGBKAPDPWRLYHERVHIIYNRZTYSBBHAWNPNITVAAAZYYLNIEIHFFIMHAMBLJQ
VTNHVBXFSRVXWCJQHQUSUSTHORXRUPHTNUKCBOFCHRDCHLTGLTIQEOLQRQWENHYF
JUINBAGBNWMFSFITHSOAFYJLBPRFNWATVQVKUCWBKCOQIIRREMDVWJMFVEREHMVU
```

```
DHGQCTTJLBUQNOZAHUCZSWLTFCRWAYKCQHZNHOMVTTZXTEFOTWNJAVAVTXAMCZAO
DGJMHAUYWJWNYBZNYZKHGQUYTNZTSQFNVODWTJQSHNBYRLRKJUAYLVIRIVAALCIW
HQOFNHGAPRDJRNMTTOHTWSTTGUWTTEVHIWHJRTTQXJWSWWRARIIPOOVZAWRHBWGH
AMAHMXKAMGJNVNVNSAWHFUIHGSTANZWJLRDTSVWUNHWQNAMBAOGUPFBYMGNWPRHH
BCAHKYDCQATAYXXPMCKRTVDJODTUMRTBHMNYNLWNXHHCPKAPZTITRNZFYIUZNBNP
WFEVNMWEXGTTUBWGYREGRNSVTFUHAJCASVYEJVACWSLVVFJQICWNTUXWPIOLNREH
HYSBXWVBHXMIAWSHUWRXQTAXJPFTHWVPMLPDFBMIMDRXUAENSYUHTTUVASXWYNHV
OAAMDFWDRJNZRQSEWXDWNLLTFQLBMHVWXEXRZOMHRFWWNXJWTDEWBYPFFBSVPOVD
HTNJTADNMURDIKQLIPTEPNAWUZRNQBANWOYNANHLITCBLBRFARRIAURJXQWDNGQH
JSTNRRIRVWGEOLHFHVYSXPIRLFIAHYC
```

3. And this one:

```
CEJUQDBFZBCRZGPMACSVGOAWTKOTTEPVAQBJYBCNFCGEAADCJJROVJMYUHQOUMID
WXWKDQRJXVOOKOVZEUGFIUNERKLYJPIZJTKRKPVHVKACCRENEHYPWKRVYQBJUARP
HEZOPKPKGFRSQFFHWVCBGJHQXYNAYYPPYDSULKVFSDUHDOJGECDIXTBNRVFDLGYJ
BTCOGHPKVCMRMZXRJBHXRISBPMMRXSKLPEISUEWABJAWPWSNBJBWIYKTXJIJNBFZ
NQNTYFDBLYSHVFJBLSFKOZJHFWDJRQRUBCDHOHWDWIMTQWZMJRUKFXYOHQNIFHEW
UDEDCBZFIRANKLYGZADRLOZESVKBQDSIQJQRUFZEDQKFPQXMSRIYBFIGFCNKIUFZ
GQGXDYGIOYZRJXFJYBAOPCSOPRIMMIKSCGGRMMKTAEOKYMZRYNRULPQNTWCJUGRD
LZHDFRAPUKSMRKNKNFHBVCAONJEUZWQKW
```

There is a challenge at MysteryTwister (https://mysterytwister.org) that is similar to these examples. If you would like to try it, you can find it at https://mysterytwister.org/media/challenges/pdf/mtc3-stamp-08-purple1-en.pdf.

Afterword

What now? You have learned everything there is to know about classical cryptography. Just kidding. Keep learning. There are still more ciphers we have not covered; many can be found on the American Cryptogram Association's website at www.cryptogram.org/resource-area/cipher-types. When you feel the urge, find one that is new to you. Study it, understand it, and implement it. Find a weakness, then a way to break it. Sometimes, the best you can do is a brute-force attack, but for classical ciphers that is still an achievement when you have modern computers to help you. Furthermore, new classical ciphers are invented still, usually as challenges to other enthusiasts. Several of the ciphers in the section on miscellaneous ciphers were invented for that reason.

After the classical era comes the mechanical era, and we have seen a few of the machines from that era. But there are more, and Wikipedia (https://wikipedia.org) is a good place to get an idea of what's out there. The Crypto Museum (https://www.cryptomuseum.com) has enough detail about a machine that you can usually create a simulation. Another avenue of study is to examine the techniques used by cryptanalysts during World War II. For example, the Bombe was a device that, together with a crib, was used to recover the key for messages encrypted with Enigma. It exploited cycles in the permutations of the machine.

We have not mentioned *Kerckhoffs's principles* yet. He summarized some basic properties of a good cryptographic system, such as its ease of use and portability. His second principle is that the security of the system should not depend on the secrecy of the algorithm (or mechanical device), but rather on the secrecy of the keys. One's enemy can often find and steal the details of the system or device, so we should not rely on keeping them hidden. Instead, the system should be strong enough that an enemy cannot break it without knowing the keys. As you know by now, none of the classical ciphers are secure in this light. Furthermore, when a new classical cipher comes along, we can often figure out the scheme and break it without prior knowledge of it. In the modern era, however, Kerckhoffs's second principle is taken very seriously.

The modern era is characterized by the use of computers. With them comes a level of complexity that makes it impossible to break modern cryptographic systems with pen and paper. But on the other hand, new structures and uses for them arise. The ideas that you should carry forward from the classical into the modern era are substitution, transposition, and fractionation; these ideas are used, albeit in more complicated ways. Here are some of the major differences you will see as you study the cryptography of the modern era:

- Algorithms are far more complex, and therefore...
- We must use computers
- Algorithms are publicly known. In fact, there are competitions for new algorithms. The current standard, *AES* ("*Advanced Encryption Standard*") employs the algorithm that won a competition held by the U.S. National Institute of Standards and Technology (NIST). Such

competitions take years to complete, as the community evaluates each algorithm and tries to find its weaknesses. Since everyone knows the algorithms, security rests in the secrecy of the keys (Kerckhoffs's second principle).

- *Asymmetric* (*public-key*) ciphers now allow someone to encrypt a message for a recipient s/he has never met before. Such a cipher has two keys; one locks it, and the other unlocks it. The recipient published his/her locking key (the public key) for anyone to use. Messages meant for the recipient are encrypted with the public key, and only the recipient can decrypt with the other, private, key.

- Public-key cryptography also allows us to digitally sign messages. If a message is encrypted with a private key, then anyone can use the public key to decrypt it, thereby proving that it was written by the individual holding the private key. To make this process easier, we only need to encrypt a shorter text, which is a secure digest of the message; this brings us to...

- *Hash functions*: these are functions that take a message and produce a large number, called the *digest* of the message. These functions are special in that it is easy to find the digest of a message, but nearly impossible to recover the message from the digest. For this reason, they are called *one-way functions*.

- The security of a cryptographic system is demonstrated by reducing it to a hard mathematical problem. For example, the *RSA* public-key system uses the difficulty of factoring large integers as the basis of its security.

The quantum era is now beginning. Quantum computers will allow us to solve some classically difficult mathematical problems easily. For example, once a quantum computer can be built that is large enough (really no larger in terms of memory and processing than computers that we now have), it will be able to factor large integers almost instantly. This will mean the death of RSA. Therefore, we need new algorithms, algorithms that are resistant to quantum computing. There is currently a competition at NIST for such things. Stay tuned.

Just keep learning.

Reading and references

American Cryptogram Association, "The ACA and You," http://www.cryptogram.org/cdb/aca.info/aca.and.you/aca.and.you.pdf, 2005 edition: http://web.archive.org/web/*/http://www.cryptogram.org/cdb/aca.info/aca.and.you/aca.and.you.pdf, 2016 edition: web.archive.org/web/*/http://cryptogram.org/docs/acayou16.pdf

Auguste Kerckhoffs, "La cryptographie militaire," *Journal des sciences militaires* IX (1883) 5-39 and 161-191, www.petitcolas.net/kerckhoffs/crypto_militaire_1_b.pdf, www.petitcolas.net/kerckhoffs/crypto_militaire_2.pdf

Turing Bombe Tutorial, www.lysator.liu.se/~koma/turingbombe/TuringBombeTutorial.pdf

Whitfield Diffie and Martin E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory* 22 (1976) 644-654, ee.stanford.edu/~hellman/publications/24.pdf

Exercises

1. Find a cipher you have never used before and learn how it works. A good place to start is www.cryptogram.org/resource-area/cipher-types. Can you find a weakness in the cipher? Can you modify an attack that you have to break this cipher? Give it a try.

2. Repeat Exercise 1 as often as you like.

3. Read about modern ciphers and hash functions, and about quantum key distribution.