Searching for a Solution

... just a bunch of boring stuff ...

Simone Sandri

Copyright © 2021 Simone Sandri

PUBLISHED BY ME

BOOK-WEBSITE.COM

Licensed under the Creative Commons Attribution-NonCommercial 3.0 Unported License (the "License"). You may not use this file except in compliance with the License. You may obtain a copy of the License at `http://creativecommons.org/licenses/by-nc/3.0`. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Digital copy, January 2021

Contents

Introduction

1. Before Reading

This book is a collection of riddles, challenges and their solutions. It follows the pseudo rule: described histories and facts are pseudo real, described techniques are pseudo documented, proofs are pseudo mathematical or follows some pseudo logical path, challenges and riddles are pseudo unique and pseudo error free and, finally, solutions are pseudo correct. Report any pseudo problem or pseudo suggestion to the github page where you retrieved this book.

1.1 What is the book about

1.2 Who should read it

1.3 Who should not read it

1.4 Riddles and Challenges

This book is about problems and their solution, so you are supposed to try to solve them. There are 2 kinds of problems:

- riddles: are problems related to the current topic, solvable by applying the same topic concept and some other simple transformations. The solution of the riddle, usually explained, is immediately after the problem statement;
- challenges: are problems that require functional brain to be solved. They not only cover the current topic, but usually a combination of techniques are involved. It can be the case that you need to go through further chapters and sections in order to solve them. Solutions are not provided. What tphe hell? Yes, solutions are not provided, have fun.

1.5 A quite important difference

Nowadays there are lots of different ways to interpret the words encoding, cryptography and steganography, some interesting, some only junk. The actual problem is that some techniques proposed here can be found somewhere under different sections and chapters.

This book follows a specific logic:

- Steganography: from Greek steganos, meaning cover. The steganography chapter will contain all techniques that, in some way, covers the plaintext message and hide it.
- Encoding: All techniques that fit the plaintext according to a specific communication channel.
- Cryptography: Similar to steganography, but with the important difference that in order to transform the plaintext some algorithms are used in combination with a specific key.

1.6 Java Code

Some techniques are enriched by Java code that implements their functionality. Why Java? Because I know it. Implementations have been tested but can contain some errors due to lack of debugging. Java code is not optimized at all but at least is readable and give you a start point to write more clever code if you need.

2. General Knowledge

2.1 An Unique Meaning?

When we write an integer number we usually concatenate a set of symbols to represent units, hundreds, millions and so on. All symbols we write belongs to the alphabet composed by digits from 0 to 9. Without any knowledge on how to interpret this sequence, it will remain a sad string on a sad white paper. But we all know that the string has a specific meaning and it represents a specific quantity, but how? It is because we are trained since children to interpret the sequence of digits as an information encoded in a particular way, and to decode it with an algorithm. What helps us in leaving this decoding procedure hidden in our brain is the universal acceptance of the encoding system and the fact that the sequence of symbols representing the encoded value is the same, and in the same order, of the sequence representing the real information, making the decoding algorithm really useless. But how is it possible?

Trust in me when I say that 1846 and 2471 have the same meaning and represents the same quantity. The only difference of the 2 numbers is that the fist has been written in base 10 (1846_{10}), while the second has been written in base 9 (2471_9). Remember when I said that numbers are building by concatenating symbols belonging to some alphabet? Bases simply represent the cardinality of that alphabet. Usually the alphabet for bases 1 till 10 contains the symbol '0', and its base-1 successors in the integer sequence so that the alphabet for base 10 is composed by symbols 0,1,2,3,4,5,6,7,8,9 and the alphabet for base 9 by 0,1,2,3,4,5,6,7,8. For bases greater then 10, other symbols are added to represents more information, but we will see this later. This, anyway, does not solve my assertion: why 1846_{10} and 2471_9 represent the same quantity?

Before answer this question, think about what a quantity is, and how it is computed. When there are 9_{10} apples on a table we know it because we count them: apple 1, apple 2, till eventually we will count apple 9. How can we do the same reasoning if we reason in base 9 and our alphabet is only composed by digits from 0 to 8? As before we start count apple 1, apple 2 till apple 8. We know that there is one more apple in the table but we cannot say apple 9 because we do not recognize 9 as a valid symbol in our system. We instead know that the numeric sequence in infinite

and it must proceed in some way. Very simple, in the same way we pass from 9_{10} to 10_{10} in base 10 system, we pass from 8_9 to 10_9 in base 9 system. So counting till 9_{10} requires the same number of steps of counting till 10_9, in the same way as counting till 1846_{10} requires the same number of steps of counting till 2471_9.

2.1.1 Base Conversion

Proving that two numbers are equivalent in different bases by comparing the number of steps required to count till reach them is really expensive. A better solution is to convert one number to the base of the other one and check if they are equals.

Base b to Base 10

I said before that we use an algorithm to give a mean to a sequence of digits, that is the same that we use to obtain the number of steps required to count till the number itself, that is the same that we use to convert a number expressed in an arbitrary base b to the equivalent one expressed in base 10. Given that the number is composed by n+1 digits $a_n, a_{n-1}, ..., a_0$ (ex. for number 256: a_2=2, a_1=5, a_0=6) the algorithm can be expressed by the following formula:

$$step_{num}(count_{till}(a_n, a_{n-1}, ..., a_0)) = \sum_{i=0}^{n} b^i * a_i$$

Where b is the origin base. Applying it to 1846_{10} we get $6*10^0+4*10^1+8*10^2+1*10^3 = 6*1+4*10+8*100+1*1000 = 6+40+800+1000 = 1846$ that means that we require 1846 steps to count from 1_{10} to 1846_{10}. It can be noticed that the sequence of digits of input and output of the algorithm remained the same, in the same order. Thats why I said before that we do not really apply the algorithm in order to give some meaning to numbers written in the universal accepted base 10. Now it is time to check if 2471_9 requires the same numbers of steps. So again we get $1*9^0+7*9^1+4*9^2+2*9^3 = 1*1+7*9+4*81+2*729 = 1+63+324+1458 = 1846$

A special base used by machines is base 2. All information stored on computers and, more in general, calculators, is stored in transistors that can assume only 2 possible states. This bits of information are typically represented as 0 and 1 in a binary system. The system follows exactly the rules described before so counting till 9_{10} is the same of counting till 1001_2. To check this let's apply the algorithm: $1*2^0+0*2^1+0*2^2+1*2^3 = 1*1+0*2+0*4+1*8 = 1+0+0+8 = 9$.

Challenge 2.1 convert the following numbers into base 10
330012221_4
53421_6
100212201_3
70245_8
44312_5
5722_9
4332_7
100110_2 ■

Solution 2.1 $330012221_4 = 246185_{10}$
$53421_6 = 33001222153421_{10}$
$100212201_3 = 7201_{10}$
$70245_8 = 28837_{10}$
$44312_5 = 3082_{10}$
$5722_9 = 4232_{10}$

$4332_7 = 1542_{10}$
$100110110_2 = 310_{10}$

Base 10 to Base b

We just learned how to convert a number expressed in an arbitrary base to the respective number in base 10. What about the inverse? Consider the following: number 14_{10} is number 24_5, 20_7 and 32_4. What have these numbers in common? they are all composed starting with 14_{10} dividing it by the respective base 5, 7 and 4 and concatenating the rest of the division: $14/5 = 2r4$; $14/7 = 2r0$; $14/4 = 3r2$. Again consider the following: number 141_{10} is number 1031_5, 261_7 and 2031_4. This time the previous assertion does not work, but works if we consider division sequentially: to pass from 141_{10} to 1031_5 we need to sequentially divide 141 by 5 and annotate the rest of the divisions. So that $141/5 = 28r1; 28/5 = 5r3; 5/5 = 1r0; 1/5 = 0r1$. The result is the concatenation of to the computed rests 1,0,3,1 (1031). To pass from 141_{10} to 261_7 we have that $141/7 = 20r1; 20/7 = 2r6; 2/7 = 0r2$. The result is the concatenation of the computed rests 2,6,1 (261). Finally to pass from 141_{10} to 2031_4 we have that $141/4 = 35r1; 35/4 = 8r3; 8/4 = 2r0; 2/4 = 0r2$. The result is the concatenation of the computed rests 2,0,3,1 (2031). The full algorithm can be expressed as follows:

- Divide the base 10 number by the output base and record the remainder.
- Divide the resulting quotient by the output base and record the remainder.
- Repeat step 2 until the quotient is zero.
- The number in output base will be composed by concatenating the remainders written in backwards order.

Challenge 2.2 convert the following base 10 numbers into the specified base
456_{10} to base 4
854_{10} to base 6
94310 to base 7
22_{10} to base 3
559_{10} to base 2
1140_{10} to base 9
287_{10} to base 8
10024_{10} to base 5 ■

Solution 2.2 $456_{10} = 13020_4$
$854_{10} = 3542_6$
$94310 = 2515_7$
$22_{10} = 211_3$
$559_{10} = 1000101111_2$
$1140_{10} = 1506_9$
$287_{10} = 437_8$
$10024_{10} = 310044_5$

Base a to Base b

Now that is clear how to convert a number written in base 10 to any other arbitrary bases, and how to get the base 10 value of a number expressed in any other base, how can we convert from and to bases that are not 10? It is always possible to convert from base A to base B by passing through base 10, but can we do better?

A particular case is when bases power of 2 are involved. Due to binary nature of base 2 num-

bers, the equivalent base 4, and base 8 numbers can be build starting from sequences of fixed length of the binary input. Consider as example the number 100111_2. The equivalent base 4 number is 213, that is composed by splitting the base 2 number in 3 buckets of 2 consecutive digits, converting to base 10, converting to base 4 and concatenating them together in the following way: $10_2 = 2_{10}; 01_2 = 1_{10}; 11_2 = 3_{10}$. Then we convert single results to base 4: $2_{10} = 2_4; 1_{10} = 1_4; 3_{10} = 3_4$ Concatenating back we obtain the number 213_4. As can be noticed we do not really have to convert single digits from base 10 to base 4, because convert couples of base 2 digits to base 10 will produce a symbol that belongs also to base 4 alphabet. In the same way also converting 3 base 2 digits to base 10 will produce a symbol that belongs also to base 8 alphabet. In this way we can convert 100111_2 to base 8 by splitting the number in buckets of 3, converting them to base 10, that will be equivalent to base 8, and concatenating back: $100_2 = 4_8; 111_2 = 7_8$. The result is 47_8.

Buckets are built starting from right to left. If the left most bucket has less digits than the others, trail zeros in order to obtain the same cardinality. Generally speaking it is possible to convert base 2 to base 2^n by splitting the number into buckets of n elements.

II

Math

3. Modular Math

3.1 The modulo operator

In math the modulo operator returns the remainder of a division between two integer numbers. This result is usually called modulus of the operation. Formally speaking the operator can be defined as follows:

"Given two positive numbers a and n, a modulo n (abbreviated as a mod n) is the remainder of the Euclidean division of a by n, where a is the dividend and n is the divisor."

For example, the result of 39 mod 7 is 11, the result of 57 mod 21 is 15 and the result of 7 mod 2 is 1. A common math expression to describe the modulo operator is the one proposed by Donald Knuth:

$$r = a \bmod n \rightarrow r = a - n * \left\lfloor \frac{a}{n} \right\rfloor \tag{3.1}$$

modulo operator has also some nice properties:

$$(a \bmod n) \bmod n = a \bmod n \tag{3.2}$$

$$n^x \bmod n = 0 \tag{3.3}$$

3.2 Congruence

We say that two integers are congruent modulo $n > 1$ if n is a divisor of their difference. This means that exists an integer k such that $a - b = k * n$. Taking in mind that the congruence symbol is $\equiv$, this statement can be expressed as:

$$a \equiv b (\bmod n) \rightarrow a - b = k * n \rightarrow a = k * n + b \tag{3.4}$$

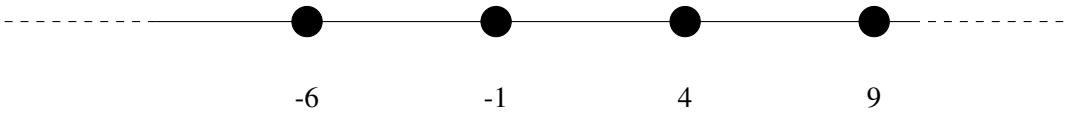
As an example we can check if $33 \equiv 5 (\bmod 7)$. Is there an integer k such that $33 = k * 7 + 5$? Yes, with $k = 4$ the equation is valid! Congruences have really nice properties:

$$a \equiv a(\bmod n) \tag{3.5}$$
$$\text{if} \quad a \equiv b(\bmod n) \quad \text{then} \quad b \equiv a(\bmod n) \tag{3.6}$$
$$\text{if} \quad a \equiv b(\bmod n) \quad \text{and} \quad b \equiv c(\bmod n) \quad \text{then} \quad a \equiv c(\bmod n) \tag{3.7}$$

3.2.1 Residue Classes and Least Residue

It can be easily seen than more than different numbers have the same remainder when divided by an integer n. For example 19 and 14 have the same remainder when divided by 5. In fact $19 \equiv 14(\bmod 5)$! From this observation we can say that 19 and 14 somehow belongs to the same class. A more formal formulation is the following:

"Given any integer a, the collection of all integers congruent to a modulo n is known as the residue class, or congruence class, of a mod n."



Residue classes can be easily plotted on a line. For example the above image shows the residue class of 4 mod 5. This because $-6 \equiv -1 \equiv 4 \equiv 9(\bmod 5)$.

Another important concept in modular arithmetic is the Least Residue. It is defined as:

The least positive residue of a modulo n is the smallest positive integer k such that $a \equiv k(\bmod n)$

Challenge 3.1 — Residue secret.
97 mod 28 134 mod 17 67 mod 7 81 mod 30 104 mod 23 99 mod 42 ■

Solution 3.1 — Residue secret.
modulo

3.3 Modular Arithmetic

When we apply operations like addition, subtraction, multiplication and division to congruences we talk about modular arithmetic.

3.3.1 Addition and Subtraction

The basic properties of congruences are the followings:

$$\text{if} \quad a \equiv b(\bmod n) \quad \text{and} \quad c \equiv d(\bmod n) \quad \text{then} \quad a+c \equiv b+d(\bmod n) \tag{3.8}$$
$$\text{if} \quad a \equiv b(\bmod n) \quad \text{and} \quad c \equiv d(\bmod n) \quad \text{then} \quad a-c \equiv b-d(\bmod n) \tag{3.9}$$

Let's immediately make an example in order to understand where this properties can be useful. We want to find the least residue of the following operation: $250+124(\bmod 50)$. There are two ways to do it: the first one is simply perform the sum between 250 and 124 and then apply the modulo operator. In this way we get that $250+124(\bmod 50) = 374 \bmod 50$. Now we need to solve 3.1 in order to obtain the remainder: $r = 374 - 50 * \left\lfloor \frac{374}{50} \right\rfloor$ that results in 24.
The second way to solve it is to notice that $250 \equiv 0(\bmod 50)$ and that $124 \equiv 24(\bmod 50)$. Thanks to 3.8 we can rewrite those statements into $250+124 \equiv 0+24 \equiv 24(\bmod 50)$. So also this way brought us to the same result.

3.3.2 Multiplications

The basic property valid for additions and subtractions also holds for multiplications:

$$\text{if} \quad a \equiv b(\bmod n) \quad \text{and} \quad c \equiv d(\bmod n) \quad \text{then} \quad a*c \equiv b*d(\bmod n) \tag{3.10}$$

This makes modular multiplications as easy as additions. Let's suppose we want to find the least residue of $407*810(\bmod 101)$. The complex way to do it is just multiply 407 times 810 and compute the least residue of 329670 mod 101, that seems hard to compute manually. The alternative is to use the just learned property: we notice that $407 \equiv 3(\bmod 101)$ and $810 \equiv 2(\bmod 101)$. Now just apply 3.10 in order to simplify things and obtain $407*810 \equiv 3*2 \equiv 6(\bmod 101)$

3.3.3 Powers

It can be noticed that, from 3.10, that if $a \equiv b(\bmod n)$ and $a \equiv b(\bmod n)$ then $a*a \equiv b*b(\bmod n)$ and so $a^2 \equiv b^2(\bmod n)$. This bring us to a more generalized statement for powers:

$$\text{if} \quad a \equiv b(\bmod n) \quad \text{then} \quad a^m \equiv b^m(\bmod n) \quad \forall m > 0 \tag{3.11}$$

3.4 Finite Fields

In classical arithmetic, we are accustomed to working with the integers, rational numbers, or real numbers. These sets are infinite and allow for arithmetic operations such as addition, subtraction, multiplication, and, in most cases, division. However, many areas of mathematics and computer scienceparticularly number theory, algebraic geometry, and cryptographyrequire working in systems where the set of available numbers is finite and arithmetic wraps around after a certain point. Such structures are known as *finite fields*. Finite fields were first systematically studied by Évariste Galois in the early nineteenth century. His revolutionary insights into polynomial solvability and group theory led to the realization that for every prime power $q = p^n$, there exists a unique (up to isomorphism) finite field with q elements. This profound result unifies much of modern algebra and underpins many applied mathematical systems, from error-correcting codes to algebraic geometry. Finite fields provide a rigorous and elegant algebraic framework for performing exact arithmetic with a fixed number of elements. Their finite nature makes them ideal for digital computation, as all results are confined to a finite range of integers, avoiding overflow or rounding errors. Moreover, their algebraic completeness guarantees that every nonzero element has a multiplicative inverse, ensuring that both addition and division remain valid operations.

3.4.1 The Structure of a Finite Field

A *finite field* (or *Galois field*) is a set of finitely many elements equipped with two operations, addition and multiplication, that satisfy the standard field axioms: closure, associativity, commutativity, distributivity, existence of additive and multiplicative identities, and existence of additive and multiplicative inverses for all nonzero elements. The simplest examples are the *prime fields*, denoted $\mathbb{F}_p$, where p is a prime number. These fields consist of the integers

$$\mathbb{F}_p = \{0, 1, 2, \ldots, p-1\}, \tag{3.12}$$

with addition and multiplication performed modulo p. That is, for any $a, b \in \mathbb{F}_p$,

$$a + b = (a + b) \bmod p, \qquad a \cdot b = (ab) \bmod p. \tag{3.13}$$

All arithmetic remains within the range $0, 1, \ldots, p-1$.

3.4.2 Existence of Multiplicative Inverses

A key property of a finite field is that every nonzero element possesses a multiplicative inverse. For $a \in \mathbb{F}_p \setminus \{0\}$, there exists $a^{-1} \in \mathbb{F}_p$ such that

$$a \cdot a^{-1} \equiv 1 \pmod{p}. \tag{3.14}$$

This inverse can be found using the *extended Euclidean algorithm*, which computes coefficients x and y satisfying $ax + py = 1$. Reducing x modulo p yields a^{-1}. For example, in $\mathbb{F}_7$, we have $3^{-1} = 5$ because $3 \times 5 = 15 \equiv 1 \pmod 7$.

The requirement that p be prime is crucial. If the modulus were composite, some elements would fail to have inverses. For instance, in arithmetic modulo 8, the number 2 has no inverse, since $2x \equiv 1 \pmod 8$ has no integer solution.

3.4.3 Examples of Computation

In the field $\mathbb{F}_7$,

$$5 + 4 \equiv 2 \pmod 7, \qquad 3 \times 6 \equiv 4 \pmod 7, \qquad 2^{-1} \equiv 4 \pmod 7.$$

These simple relationships demonstrate how all arithmetic remains confined within the set $\{0, 1, 2, 3, 4, 5, 6\}$. In this environment, both algebraic manipulation and numerical computation are straightforward and exact.

3.4.4 Extension Fields

Beyond prime fields, finite fields can also be constructed with $q = p^n$ elements, where $n > 1$. Such fields are called *extension fields* and are denoted $\mathbb{F}_{p^n}$. They are built as quotient rings of the form

$$\mathbb{F}_{p^n} = \mathbb{F}_p[x]/(g(x)), \tag{3.15}$$

where $g(x)$ is an irreducible polynomial of degree n over $\mathbb{F}_p$. In these fields, elements are represented as polynomials of degree less than n, and arithmetic is performed modulo both p and $g(x)$. Such constructions are especially useful in coding theory and digital communications, where binary extension fields $\mathbb{F}_{2^n}$ allow efficient bitwise operations.

4. Combinations and Permutations

4.1 Permutations

A permutations is a rearrangement of the elements of a set. For example all permutations of the set $S = a,b,c$ are (a,b,c), (a,c,b), (b,a,c), (b,c,a), (c,a,b), (c,b,a). A classical real world example of a permutation is the anagram of a word. "dog" is an anagram of "god" and both are different permutations of the alphabetically ordered set d,g,o.

R Technically speaking, a permutation of a set S is defined as a bijection from S to itself. This means that it is a function from S to S for which every element occurs exactly once as an image value. This is related to the rearrangement of the elements of S in which each element s is replaced by the corresponding f(s).

The first knowing example of permutations appeared in China in the I Ching as early as 1000 BC with the name of hexagrams. The first practical use of permutations can be found in the Book of Cryptographic Messages written by the Arab mathematician and cryptographer Al-Khalil around 8th Century.

4.1.1 Permutations? with repetitions

The easiest example of permutations are those that allows repetition of the elements of the original set. These are usually called n-tuples or simply, in some contexts, words over the alphabet S. As you can notice in the title of the subsection there is a question mark. This because even if some books call these sequences of symbols permutations, by definition above they are not permutation at all!

Given the set $S = a,b,c$, it is possible to extract all n-tuples of a specific length n:

- n=1 →a, b, c;
- n=2 →aa, ab, ac, ba, bb, bc, ca, cb, cc;

- n=3 →aaa, aab, aac, aba, abb, abc, aca, acb, acc, baa, bab, bac, bba, bbb, bbc, bca, bcb, bcc, caa, cab, cac, cba, cbb, cbc, cca, ccb, ccc;
- ...

More generally the following statement is true

R If the set S has k elements, the number of n-tuples (tuples with length n) over S is k^n.

This because for each position of the tuple we have k elements from which to choose. So k elements on the first position, times k elements on the second position, ..., times k elements on the nth position we obtain:

$$\underbrace{k * k * \ldots * k}_{n \quad times} = k^n \tag{4.1}$$

4.1.2 Permutations without repetitions

It is easy to see, looking at the definition, that permutations does not allow repetitions because are simply rearrangements of the original set. If all the items have to be used we refer to the ways of arranging n items into n places. The number of different ways to do it is given by $n!$. This because there are n elements that can fit the first position. Now that one element has been consumed there are only $n-1$ elements that fit the second position, and so on till in the last position only 1 element can be used:

$$n * n-1 * \ldots * 1 = n! \tag{4.2}$$

For example, if we have a set $S = a,b,c$ there are $3*2*1 = 6$ ways to arrange them: (a,b,c), (a,c,b), (b,a,c), (b,c,a), (c,a,b), (c,b,a).
As far we talked about permutations without repetitions using all the items in the original set, but what if we can build tuples that are shorter than the set cardinality? If we want to create a tuple of length k starting from n elements we can reason as follows: at position one we can use n elements, at position two we can use $n-1$ elements, ..., at position k we can use $n-k+1$ elements. More generally:

$$\underbrace{n * n-1 * \ldots * n-k+1}_{k \quad times} = \frac{n!}{(n-k)!} \tag{4.3}$$

This is the general formula to obtain the number of possible ways to extract k elements from a set of n elements, without repetitions. The notations to express the formula are different:

$$P(n,k) = {}_nP_k = {}^nP_k = P_{n,k} = P_k^n = \frac{n!}{(n-k)!} \tag{4.4}$$

5. Number Theory

5.1 Prime Numbers

By definition a prime number, prime abbreviated, is a natural number, strictly grater than 1, that is not a product of two smaller natural numbers. If a number is not prime then is a composite number. Here follows the first 100 prime numbers:

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541

The property of being prime is called primality. Since prime numbers are a fundamental brick of arithmetic their properties have been largely studied since ancient times and, around 300 BC, Euclid first demonstrated that there are infinitely many primes.

5.1.1 Fundamental Theorem of Arithmetic

We just saw that prime numbers cannot be represented by the product of two smaller numbers. This means that a prime number p can be only divided by 1 or by itself. But what about composite numbers? This doesn't hold. For example $120 = 12*10 = 6*20 = 4*30$.... But we can state more:

R Each composite number can be represented as the product of some prime numbers and this representation is unique.

In our example $120 = 2*2*2*3*5*2^3*3*5$. There no exists another composite number with the same factorization. This theorem also explain why 1 is not considered as a prime:

R If 1 is prime than $4 = 1*2^2 = 1^2*2^2 = 1^3*2^2$... and so the statement of unique representation doesn't hold anymore.

More generally speaking every positive integer greater than 1 can be represented in exactly one way as a product of prime powers:

$$n = p_1^{n_1} * p_2^{n_2} * \ldots * p_k^{n_k} = \prod_{i=1}^{k} p_k^{n_i} \tag{5.1}$$

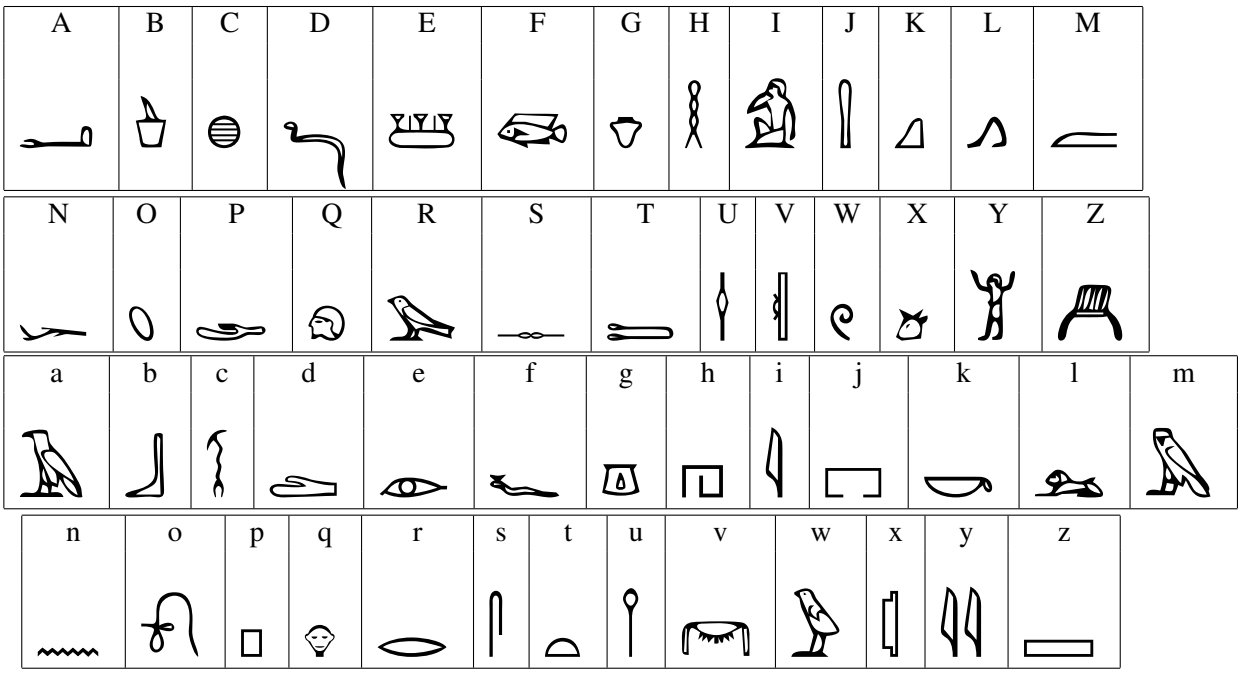
This is called canonical representation of a positive integer. Since we are not here to talk about math but hiding information, how it is possible to do so using this knowledge? Well the following technique is pretty smart. considering that the canonical representation is ordered by primes, we can use primes as placeholders and exponents as a way to encode information. For example: $52929948960000 = 2^8 * 3^9 * 5^4 * 7^5$. Getting only the exponents we have 8, 9, 4 and 5 that represents the positions of the letters h, i, d and e in the english alphabet, so 52929948960000 = hide.

Encoding

6. Ancient Languages

Sometimes different languages and unknown draws appears to us as a strange form of steganography with invented alphabets and strange graphical symbolism. Most of the times what is unknown to us is, or has been, well known to a large group of other people, with different cultures and in different epochs

6.1 Egypt (3000 B.C.)



1	10	100	1000	10000	100000	1000000

6.2 Babylonian Numbers (2000 B.C.)

In 2000 B.C. developed the first positional numeral system, where the position of each digit matters in order to interpret the whole number. The digits were composed by cuneiform symbols made by a reed stylus on soft clay.

TODO

6.3 Runic alphabets (150 A.D.)

Runes are the symbols coming from the Runic alphabet, an old notation used to write different Germanic, Scandinavian and Anglo-Saxon languages before the official adoption of the Latin alphabet.

6.3.1 Elder Futhark (2nd to 8th centuries)

The Elder Futhark (named after the initial phoneme of the first six rune names: F, U, Þ, A, R and K) has 24 runes. The Old English names of all 24 runes of the Elder Futhark, along with five names of runes unique to the Anglo-Saxon runes, are preserved in the "Old English rune poem", compiled in the 8th or 9th century.

Rune	Transliteration	Name	Meaning
ᚠ	f	fehu	"wealth, cattle"
ᚢ	u	ruz	"aurochs"
ᚦ	þ	þurisaz	"the god Thor, giant"
ᚨ	a	ansuz	"one of the Æsir (gods)"
ᚱ	r	raid	"ride, journey"
ᚲ	k (c)	kaunan	"ulcer"
ᚷ	g	geb	"gift"
ᚹ	w	wunj	"joy"
ᚺ	h	hagalaz	"hail"
ᚾ	n	naudiz	"need"
ᛁ	i	saz	"ice"
ᛃ	j	jra	"year, good year, harvest"
ᛇ	ï (æ)	(h)waz/ei(h)waz	"yew-tree"
ᛈ	p	perþ	"pear-tree"
ᛉ	z	algiz	"elk"
ᛊ	s	swil	"sun"
ᛏ	t	twaz/teiwaz	"the god Tyr"
ᛒ	b	berkanan	"birch"
ᛖ	e	ehwaz	"horse"
ᛗ	m	mannaz	"man"
ᛚ	l	laguz	"water, lake"
		ingwaz	"the god Yngvi"
ᛟ	o	þila/þala	"heritage, estate, possession"
ᛞ	d	dagaz	"day"

Challenge 6.1 — ruined. ᛏᚺᛖᚾ ᚠᚱᛚ ᛁᛊ ᚱᚢᛁᚾᛖᛞ, ᚠᛊ ᚲᚨᚾ ᛒᛖ ᛊᛖᛖᚾ ᛁᚾ ᛞᛁᚠᚠᛖᚱᛖᚾᛏ ᛈᛚᚨᚲᛖᛊ ᛁᚾ ᚨᛗᛖᚱᛁᚲᚨ, ᚠᛟᚱ ᛁᚾᛊᛏᚨᚾᚲᛖ.

Solution 6.1 — ruined. Then all is ruined, as can be seen in different places in America, for instance.

6.3.2 Anglo-Saxon Runes (5th to 11th centuries)

It is an extension of the common Elder Futhark alphabet, composed by 29 or 33 runes. As this last alphabet also the Anglo-Saxon one takes its name from the first six runes: F, U, Þ, O, R and K (futhork). The Anglo-Saxon runes are used by J.R.R. Tolkien in its opera "The Hobbit" on a map, to link the object to the dwarves.

Rune	Transliteration	Name	Meaning
ᚠ	f	feoh	"wealth"
ᚢ	u	r	"aurochs"
ᚦ	þ	þorn	"thor"
ᚩ	o	s	"god, or mouth (Latin)"
ᚱ	r	rd	"riding"
ᚳ	k	calc	"chalice"
ᚴ	c	cn	"torch"
ᚷ	g	gyfu	"gift"
ᚹ	w	pynn	"mirth"
ᚻ	h	hægl	"hail"
ᚾ	n	nd	"need"
ᛁ	i	s	"ice"
ᛄ	j	gr	"year"
ᛇ	ï, ʒ	oh	"yew-tree"
ᛈ	p	peorð	"pear-tree"
ᛉ	x	eolh	"elk"
ᛋ	s	sigel	"sun"
ᛏ	t	T, Tr	"Mars"
ᛒ	b	beorc	"birch-tree"
ᛖ	e	eh	"horse"
ᛗ	m	mann	"man"
ᛚ	l	lagu	"lay, lake"
		ðel	"ethel"
ᛞ	d	dæg	"day"
ᚪ	a	c	"oak-tree"
ᚫ	æ	æsc	"ash-tree"
ᚣ	y	r	"bow"
ᛝ	q	cweorð	"???"
ᛠ	ea	ar	"grave"
ᛡ	g	gar	"spear"
ᛣ	k	???	"???"
ᛤ	rex	???	"???"
ᛥ	stan	stone	"day"

Challenge 6.2 — destroyed. ᚠᚠᚱ ᛖᚠᚳᚻ ᛖᛉᚳᛖᚳᛏᚠᛏᛁᚠᚾ ᛏᚻᚠᛏ ᚻᛖ ᚠᚢᛚᚠᛁᛚᛚᛖᛞ, ᛏᚻᛖᚱᛖ ᚹᚠᛋ ᚠᚾᚠᛏᚻᛖᚱ ᛏᚻᚠᛏ ᚻᛖ ᛞᛖᛋᛏᚱᚠᚢᛖᛞ. ■

Solution 6.2 — destroyed. For each expectation that he fulfilled there was another that he destroyed.

6.3.3 Younger Futhark (9th to 11th centuries)

Younger Futhark, aka Scandinavian Futhark, is a reduced form of Elder Futhark with only 16 symbols, used in the Viking Age. There are mainly 2 types of Younger Futhark runes: The Long-branch (used in Denmark) and the Short-twig runes (used in Sweden and Norway). There is also another type of runes coming across the 10th and the 12th century, called Staveless or Hälsinge, first noticed in the region of Hälsingland of Sweden. The next table will summarize this runes

Long-branch	Short-twig	Staveless	Transliteration	Name	Meaning
ᚠ	ᚠ	I	f/v	fé	"wealth"
ᚢ	ᚢ	)	u/w, y, o, ø	úr	"iron"
ᚦ	ᚦ	ˈ	þ, ð	Thurs	"giant"
ᚬ	ᚭ	ˎ	, o, æ	As	"???"
ᚱ	ᚱ	(	r	reið	"ride"
ᚴ	ᚴ	ˌ	k,g	kaun	"ulcer"
ᚼ	ᚽ	ı	h	hagall	"hail"
ᚾ	ᚿ	ˋ	n	nauðr	"need"
ᛁ	ᛁ	|	i/e	ísa/íss	"ice"
ᛅ	ᛆ	ˏ	a, æ, e	ár	"plenty"
ᛋ	ᛌ	ˈ	s	sól	"sun"
ᛏ	ᛐ	ˊ	t, d	Týr	"???"
ᛒ	ᛓ	ˏ	b, p	björk/bjarkan/bjarken	"birch"
ᛘ	ᛙ	:	m	maðr	"man"
ᛚ	ᛚ	ˋ	l	lögr	"sea"
ᛦ	ᛧ	:	R	yr	"yew"

Challenge 6.3 — damaged. ᛘᛅᚾ ᛁ ᚴᚭᛏ ᛏᛅᛘᛅᚴᛂᛏ ᛏᚭᛏᛅᚢ, ᛅᚾᛏ ᛁᛏ ᚼᚢᚱᛏ ᛚᛁᚴᛂ ᚼᛂᛚᛚ. ■

Solution 6.3 — damaged. Man, I got damaged today, and it hurt like hell.

Challenge 6.4 — broken. ᚿᚭᛐᚽᛁᚿᚴ ᛓᚱᛆᚴᛌ ᛚᛁᚴ ᛆᚿ ᚽᛆᚱᛐ. ■

Solution 6.4 — broken. nothing breaks like a heart

Challenge 6.5 — disturbed. ı)ˏˈ |ˋˊ(ˋˏˋˋ ˊ|ˈˊ)(ˏˊ ˊ ıˏ(ˎ| ı(|ˋˋˎˈˈ. ■

Solution 6.5 — disturbed. He was internally disturbed to hear of her illness

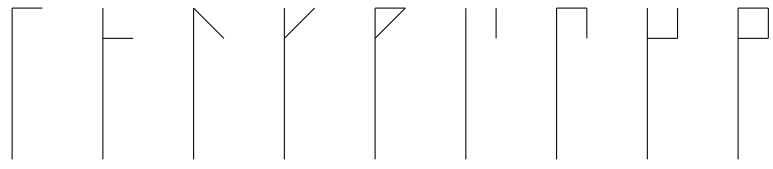
7. Numeral Encodings

7.1 Cistercian numerals (13th Century)

With the purpose to compact arabic numerals, and to represent numbers from 1 to 9999 with a single symbol, the Cistercian monastic order, by John of Basingstoke archdeacon of Leicester, developed their own numeral system. The fist attempt to create such a system was limited to numbers less than 99 and only successively expanded to numbers till 9999. Even if usually numerical number systems are also used for arithmetic operations it seems that the only documented use of the Cistercian numerals is for indicate years, page numbers, indexes, lists, tables and musical notations.

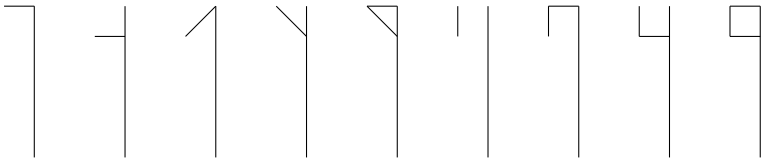
Even if the majority of the documents containing Cistercian numerals relies to the order itself, there is evidence of some outside usage. French Picardy, for example used the system in its "astrolabe of Berselius", a tratise of physics. Furthermore in 1533, Heinrich Cornelius Agrippa von Nettesheim included a description of the system in his Three Books of Occult Philosophy. The Cistercian numeral appeared here an then, in marginal way, till the early twentieth century.

Digits are based on a horizontal or vertical stave, with the position (in the four main quadrants: up-right, up-left, down-right, down-left) of the digit on the stave indicating its place value (units, tens, hundreds or thousands). These digits are compounded on a single stave to indicate more complex numbers. There ale lots of historical variation but here we will describe the common one. The following are the basics 9 digits, from 1 to 9:

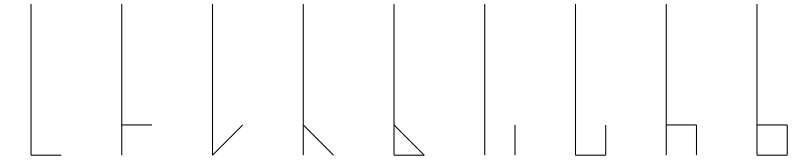


1 2 3 4 5 6 7 8 9

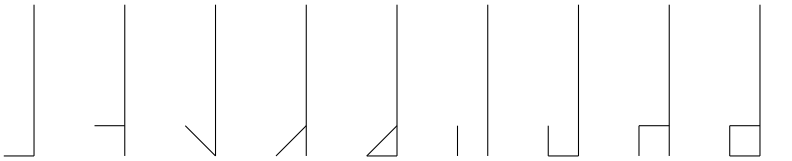
Mirroring this basic digits horizontally we obtain the tens:



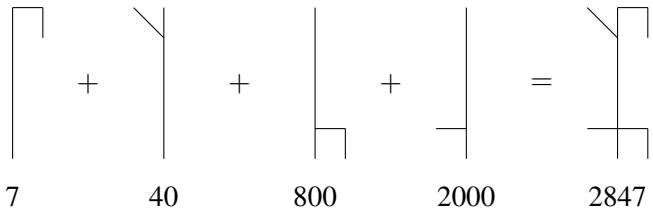
Back to the original basic units, mirroning vertically we obtain the hundreds:



Finally, mirror horizontally the hundreds we obtain the thousands:



Now that we know the basics we can combine them in order to obtain a more complex number, for example:



7.2 Binary-Coded Decimal (BCD)

Binary-Coded Decimal (BCD) is a class of binary encodings in which each digit of a decimal number is represented by a fixed number of binary bits. Unlike pure binary representation, which converts an entire decimal value into a single binary number, BCD encodes each decimal digit individually. This approach preserves the decimal digit structure, making it particularly useful in systems that require precise decimal arithmetic or easy conversion between human-readable decimal numbers and internal binary representations. In conventional binary representation, numbers are expressed as powers of two. For example, the decimal number 59 is represented in binary as 111011_2. However, in BCD, each decimal digit is treated separately:

$$59_{10} = (0101\ 1001)_{\text{BCD}}.$$

Here, the digit 5 is represented by its binary equivalent 0101, and the digit 9 is represented by 1001. Thus, two groups of four bits are used to encode the two decimal digits. The primary motivation behind BCD representation is to simplify the hardware and logic required for decimal-based operations such as those in digital clocks, calculators, financial applications, and digital instrumentation. Since each decimal digit is explicitly encoded, conversion between the displayed number and its internal representation is straightforward and requires minimal processing overhead. BCD encoding typically uses four bits (a nibble) for each decimal digit, providing $2^4 = 16$ possible binary combinations. Out of these, only ten (0000 through 1001) are valid representations of decimal digits 0 through 9. The remaining six codes (1010 through 1111) are considered invalid or unused in standard BCD systems. A BCD number containing n decimal digits requires $4n$ bits for representation. For example, the decimal number 2765 is represented in BCD as:

$$2\ 7\ 6\ 5 \Rightarrow 0010\ 0111\ 0110\ 0101_{\text{BCD}}.$$

Thus, each group of four bits corresponds directly to one decimal digit.

7.2.1 8421 BCD Code (Natural Binary-Coded Decimal)

The **8421 BCD code**, also known as the *natural BCD code*, is the most commonly used form of Binary-Coded Decimal representation. The name *8421* originates from the positional weights assigned to each bit in a four-bit group: 8, 4, 2, and 1, respectively. Each decimal digit from 0 to 9 is represented by its corresponding binary number in this weighted system. Each 4-bit combination represents one decimal digit, using the same binary weights as in natural binary numbering. The most significant bit (MSB) has a weight of 8, and the least significant bit (LSB) has a weight of 1. Thus, the decimal value of a BCD digit is calculated as:

$$\text{Decimal Value} = (8 \times b_3) + (4 \times b_2) + (2 \times b_1) + (1 \times b_0),$$

where $b_3b_2b_1b_0$ are the four bits representing the digit, with b_3 being the MSB. Table 7.1 shows the 8421 BCD code assignments for decimal digits 0 through 9.

The remaining six combinations from 1010_2 to 1111_2 are **invalid** in 8421 BCD representation and are typically unused or reserved for control purposes. To represent a multi-digit number in 8421 BCD, each decimal digit is replaced by its 4-bit equivalent:

$$459_{10} = (0100\ 0101\ 1001)_{\text{BCD}}.$$

This means:

$$4 \rightarrow 0100, \quad 5 \rightarrow 0101, \quad 9 \rightarrow 1001.$$

Thus, the full 8421 BCD representation of 459_{10} is 010001011001_2.

Table 7.1: 8421 BCD Code Representation

Decimal Digit	Binary Weight Representation (8421)	BCD Code
0	$(0\times 8)+(0\times 4)+(0\times 2)+(0\times 1)$	0000
1	$(0\times 8)+(0\times 4)+(0\times 2)+(1\times 1)$	0001
2	$(0\times 8)+(0\times 4)+(1\times 2)+(0\times 1)$	0010
3	$(0\times 8)+(0\times 4)+(1\times 2)+(1\times 1)$	0011
4	$(0\times 8)+(1\times 4)+(0\times 2)+(0\times 1)$	0100
5	$(0\times 8)+(1\times 4)+(0\times 2)+(1\times 1)$	0101
6	$(0\times 8)+(1\times 4)+(1\times 2)+(0\times 1)$	0110
7	$(0\times 8)+(1\times 4)+(1\times 2)+(1\times 1)$	0111
8	$(1\times 8)+(0\times 4)+(0\times 2)+(0\times 1)$	1000
9	$(1\times 8)+(0\times 4)+(0\times 2)+(1\times 1)$	1001

7.2.2 Aiken Code (2421 BCD Code)

The **Aiken code**, also known as the **2421 BCD code**, is a *self-complementing weighted code* widely used in digital systems for simplifying arithmetic operations, particularly subtraction using the nines complement method. The code derives its name from the positional weights of its bits, namely 2, 4, 2, and 1. In the Aiken code, each decimal digit from 0 to 9 is represented by a unique four-bit binary combination according to the assigned weights 2, 4, 2, and 1. The value of a coded digit is obtained by summing the weights corresponding to the bits set to 1:

$$\text{Decimal Value} = (2\times b_3)+(4\times b_2)+(2\times b_1)+(1\times b_0),$$

where b_3 is the most significant bit (MSB) and b_0 is the least significant bit (LSB). Because two of the bit positions share the same weight (2), the 2421 code is not a *purely weighted* code in the strict sense; however, the weight pattern gives it useful structural properties, such as self-complementation. The 2421 (Aiken) code assignments for decimal digits 09 are shown in Table 7.2.

Table 7.2: Aiken (2421) Code Representation

Decimal Digit	Weighted Sum (2-4-2-1)	Aiken Code
0	$(0\times 2)+(0\times 4)+(0\times 2)+(0\times 1)$	0000
1	$(0\times 2)+(0\times 4)+(0\times 2)+(1\times 1)$	0001
2	$(0\times 2)+(0\times 4)+(1\times 2)+(0\times 1)$	0010
3	$(0\times 2)+(0\times 4)+(1\times 2)+(1\times 1)$	0011
4	$(0\times 2)+(1\times 4)+(0\times 2)+(0\times 1)$	0100
5	$(1\times 2)+(0\times 4)+(0\times 2)+(1\times 1)$	1011
6	$(1\times 2)+(0\times 4)+(1\times 2)+(0\times 1)$	1100
7	$(1\times 2)+(0\times 4)+(1\times 2)+(1\times 1)$	1101
8	$(1\times 2)+(1\times 4)+(0\times 2)+(0\times 1)$	1110
9	$(1\times 2)+(1\times 4)+(0\times 2)+(1\times 1)$	1111

It can be observed that the first half of the table (digits 04) corresponds roughly to standard binary representation, while digits 59 deviate in order to achieve the self-complementing property. A key advantage of the Aiken code is that it is **self-complementing**. This means that the 9s complement of a decimal digit can be obtained simply by taking the bitwise complement of its 2421 code.

Mathematically, if $C(D)$ denotes the 2421 code of digit D, then:

$$C(9-D)=\overline{C(D)}.$$

For example:

$$C(3) = 0011, \quad \text{and} \quad C(9-3) = C(6) = 1100 = \overline{0011}.$$

This property simplifies subtraction in decimal arithmetic circuits, as subtraction can be performed using 9s complement addition without additional conversion logic. To encode the decimal number 375_{10} in Aiken code:

$$3 \to 0011, \quad 7 \to 1101, \quad 5 \to 1011.$$

Thus:

$$375_{10} = (0011\ 1101\ 1011)_{\text{Aiken}}.$$

7.2.3 Excess-3 Code (XS-3 Code)

The **Excess-3 code**, also known as the **XS-3 code**, is another widely used *self-complementing* BCD code. It is derived directly from the 8421 BCD system by adding a constant bias of 3 (i.e., 0011_2) to the standard 8421 binary representation of each decimal digit. This offset introduces useful arithmetic properties, particularly for complement and error detection operations. In Excess-3 encoding, each decimal digit D (where $0 \leq D \leq 9$) is represented by a 4-bit binary number equal to $(D+3)_{10}$ expressed in binary. That is,

$$\text{Excess-3 Code for Digit } D = \text{Binary Representation of } (D+3).$$

For example:

$$D = 4 \Rightarrow D+3 = 7 \Rightarrow (0111)_2.$$

Thus, the decimal digit 4 is represented in Excess-3 as 0111. The complete Excess-3 code assignments for decimal digits 0 through 9 are shown in Table 7.3.

Table 7.3: Excess-3 (XS-3) Code Representation

Decimal Digit	(Digit + 3) in Binary	Excess-3 Code
0	$3 \to 0011$	0011
1	$4 \to 0100$	0100
2	$5 \to 0101$	0101
3	$6 \to 0110$	0110
4	$7 \to 0111$	0111
5	$8 \to 1000$	1000
6	$9 \to 1001$	1001
7	$10 \to 1010$	1010
8	$11 \to 1011$	1011
9	$12 \to 1100$	1100

A notable feature of the Excess-3 code is that it is **self-complementing**. This means that the 9s complement of a decimal digit can be obtained simply by taking the 1s complement of its Excess-3 representation.

Formally:

$$C(9-D) = \overline{C(D)},$$

where $C(D)$ is the Excess-3 code for digit D, and $\overline{C(D)}$ denotes its bitwise complement.

For example:

$$C(2) = 0101, \quad C(9-2) = C(7) = 1010 = \overline{0101}.$$

This property greatly simplifies subtraction operations using complement methods in digital circuits. Let us encode the decimal number 407_{10} in Excess-3 code:

$$4 \to 0111, \quad 0 \to 0011, \quad 7 \to 1010.$$

Hence:

$$407_{10} = (0111\ 0011\ 1010)_{\text{XS-3}}.$$

7.2.4 Gray Code (Reflected Binary Code)

The **Gray code**, also known as the **Reflected Binary Code (RBC)**, is a non-weighted binary code in which any two successive code words differ by only one bit position. This property known as the *unit-distance property* minimizes ambiguity and errors during transitions between adjacent values, making Gray code especially useful in systems involving analog-to-digital conversion or mechanical position encoding. Although the Gray code is not a true BCD code in the strict sense, it is often included in discussions of BCD-like encodings due to its binary digit structure and its application in decimal and digital hardware systems. The Gray code can be constructed recursively by reflecting and prefixing bit sequences. Starting with a single-bit sequence $\{0, 1\}$, the next higher order Gray code is formed by:

1. Writing the existing sequence in direct order and then in reverse order.
2. Prefixing each entry of the first (direct) half with a 0.
3. Prefixing each entry of the second (reversed) half with a 1.

For example:

1-bit Gray code: 0, 1
2-bit Gray code: 00, 01, 11, 10
3-bit Gray code: 000, 001, 011, 010, 110, 111, 101, 100

This reflection process ensures that consecutive codes differ by exactly one bit. Table 7.4 presents the 4-bit Gray code corresponding to decimal digits 0 through 9. Only the first ten entries are typically used in decimal-based applications.

Table 7.4: 4-Bit Gray (Reflected Binary) Code Representation

Decimal Digit	4-bit Gray Code
0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101
7	0100
8	1100
9	1101

Each consecutive entry differs from the previous one by exactly one bit the defining characteristic of Gray code. To represent the decimal number 37_{10} in 4-bit Gray code, each digit is replaced by its corresponding Gray code equivalent:

$$3 \rightarrow 0010, \quad 7 \rightarrow 0100.$$

Thus,

$$37_{10} = (0010\ 0100)_{\text{Gray}}.$$

The Gray code possesses several unique properties that distinguish it from other BCD systems:

- **Unit-distance property:** Only one bit changes between successive values, minimizing transition errors.
- **Non-weighted:** No positional weights are assigned to bit positions.
- **Non-arithmetic:** Arithmetic operations such as addition or subtraction are not performed directly in Gray code.
- **Self-reflecting:** The code sequence is symmetric, meaning it reads the same forward and backward in terms of transitions.

7.2.5 GrayGlixon BCD Code

The **GrayGlixon BCD code** is a hybrid form of binary-coded decimal representation that combines properties of both the *Gray (reflected binary)* code and the conventional weighted BCD systems such as the 8421 code. Developed to retain the *unit-distance* transition property of Gray code while maintaining the decimal digit correspondence of BCD, the GrayGlixon code offers improved reliability in decimal counting and display circuits, especially where transition errors must be minimized. The GrayGlixon code is constructed to ensure that:

- Consecutive decimal digits differ by only one bit (unit-distance property).
- Each decimal digit still corresponds uniquely to one 4-bit binary pattern.

Unlike the standard Gray code, which represents a continuous binary sequence, the GrayGlixon BCD code is constrained to the decimal digits 0 through 9, omitting the remaining six 4-bit combinations (1010_2 through 1111_2) that lie beyond the decimal range. The sequence is designed so that the change from one decimal digit to the next involves a single bit transition, thus reducing transient ambiguities in systems where signals must represent successive decimal values. Table 7.5 lists the standard GrayGlixon code assignments for decimal digits 0 through 9. This sequence maintains a single-bit change between adjacent decimal values, similar to Gray code, while preserving decimal order.

Table 7.5: GrayGlixon BCD Code Representation

Decimal Digit	GrayGlixon Code
0	0000
1	0001
2	0011
3	0111
4	0110
5	0100
6	1100
7	1101
8	1111
9	1011

This mapping differs slightly from the standard 4-bit Gray code (which continues to 1110, 1111, etc.), because it is specifically constrained to represent decimal digits and maintain logical adjacency among them. To represent a decimal number such as 258_{10} in GrayGlixon code:

$$2 \to 0011, \quad 5 \to 0100, \quad 8 \to 1111.$$

Thus,

$$258_{10} = (0011\ 0100\ 1111)_{\text{GrayGlixon}}.$$

The GrayGlixon code exhibits a combination of weighted and non-weighted characteristics:

- **Unit-distance property:** Only one bit changes between successive decimal digits, reducing errors during transitions.
- **Non-weighted:** The bit positions do not have fixed numerical weights, similar to the Gray code.
- **Limited decimal range:** Only ten valid code combinations exist, corresponding to decimal digits 09.
- **Order-preserving:** The sequence of codes increases monotonically with decimal digits.

7.3 Two-out-of-Five (2-of-5) Codes

The **Two-out-of-Five (2-of-5) code** is a family of fixed-length decimal codes in which each decimal digit is represented by a group of five bits, and exactly two of those bits are set to one (1) while the remaining three are zero (0). This two ones out of five positions rule gives the code its name. The 2-of-5 code is self-checking: if a received pattern does not contain exactly two 1s, it must be erroneous. Such codes have been historically important in telecommunication signaling, postal barcodes, and early identification systems, where reliability was essential. The followings are general key properties:

- Each codeword contains exactly two ones.
- There are $\binom{5}{2} = 10$ possible distinct patterns perfectly matching the 10 decimal digits (09).
- Various systems assign the ten valid patterns differently depending on hardware or application.

7.3.1 Telecommunication 2-of-5 Code (01236 Weighting)

The **Telecommunication 2-of-5 code** (also known as the *01236 weighted code*) was used in early telegraph and telephone exchanges. Weights $0,1,2,3,6$ are assigned to the bit positions (from left to right). The sum of the weights of the two set bits equals the decimal digit being represented.

Encode 507_{10} in the telecommunication 2-of-5 code:

$$5 \to 00110, \quad 0 \to 10001, \quad 7 \to 01001,$$

thus

$$507_{10} = (00110\ 10001\ 01001)_{\text{Telecom 2-of-5}}.$$

7.3.2 POSTNET (74210 Weighting)

The **POSTNET (Postal Numeric Encoding Technique)** code was used by the United States Postal Service for mail sorting. Each decimal digit is represented by a group of five bars two full-height bars (1s) and three half-height bars (0s). Weights $7,4,2,1,0$ are assigned to the bar positions, and the sum of the weights for the two full bars equals the digit. The code also includes start/stop bars (not shown in the digit table).

Table 7.6: Telecommunication 2-of-5 (01236) Code

Decimal	Weighted sum (using 0,1,2,3,6)	Code (bit positions 04)
0	fixed	10001
1	$(0+1)=1$	11000
2	$(0+2)=2$	10100
3	$(0+3)=3$	10010
4	$(1+3)=4$	01100
5	$(2+3)=5$	00110
6	$(0+6)=6$	10001
7	$(1+6)=7$	01001
8	$(2+6)=8$	00101
9	$(3+6)=9$	00011

Table 7.7: POSTNET 2-of-5 (74210) Code

Decimal	Weighted sum (7,4,2,1,0)	Bar pattern (1 = full bar)
0	$(7+0)=7$	10001
1	$(0+1)=1$	00011
2	$(0+2)=2$	00101
3	$(0+3)=3$	00110
4	$(0+4)=4$	01001
5	$(0+5)=5$	01010
6	$(2+4)=6$	01100
7	$(1+6)=7$	10001
8	$(2+6)=8$	10100
9	$(3+6)=9$	11000

Encode ZIP+4 code segment 428:

$$4 \to 01001, \quad 2 \to 00101, \quad 8 \to 10100,$$

thus

$$428_{\text{POSTNET}} = (01001\ 00101\ 10100)_{\text{POSTNET}}.$$

7.3.3 PLANET Code (POSTNET Derivative)

The **PLANET** (*Postal Alpha Numeric Encoding Technique*) code is closely related to POSTNET, but with inverted logic: positions representing 1 in POSTNET correspond to 0 in PLANET, and vice versa. It was used by the USPS for mail tracking and identification.

Encode 59 in PLANET:

$$5 \to 10101, \quad 9 \to 00111,$$

so

$$59_{\text{PLANET}} = (10101\ 00111)_{\text{PLANET}}.$$

7.3.4 Code 39 (Bar Widths 1-2-4-7-0 System)

The **Code 39 barcode** encodes alphanumeric data using patterns of *wide and narrow* bars and spaces. The digit subset of Code 39 corresponds to a modified 2-of-5 system with bar widths

Table 7.8: PLANET Code (Complement of POSTNET)

Decimal	Pattern (inverted POSTNET)	Comment
0	01110	Inverse of POSTNET(0) = 10001
1	11100	Inverse of POSTNET(1) = 00011
2	11010	Inverse of POSTNET(2) = 00101
3	11001	Inverse of POSTNET(3) = 00110
4	10110	Inverse of POSTNET(4) = 01001
5	10101	Inverse of POSTNET(5) = 01010
6	10011	Inverse of POSTNET(6) = 01100
7	01110	Inverse of POSTNET(7) = 10001
8	01011	Inverse of POSTNET(8) = 10100
9	00111	Inverse of POSTNET(9) = 11000

Table 7.9: Code 39 Numeric (2-of-5, 12470 weighting) with Bar Visualization

Decimal	Binary Code	Bar Visualization	Comment
0	10001		Two wide at edges
1	11000		Two wide at start
2	10100		Wide at 1st and 3rd positions
3	10010		Wide at 1st and 4th
4	01100		Wide at 2nd and 3rd
5	01010		Wide at 2nd and 4th
6	00110		Wide at 3rd and 4th
7	01001		Wide at 2nd and 5th
8	00101		Wide at 3rd and 5th
9	00011		Wide at 4th and 5th

following the weights $1,2,4,7,0$. Each character is composed of 5 bars and 4 spaces, but only the bar patterns are shown here for clarity.

Encoding the decimal number 204_{10}:

$$2 \to 10100, \quad 0 \to 10001, \quad 4 \to 01100.$$

Thus the full pattern is:

$$204_{10} = (10100\ 10001\ 01100)_{\text{Code39}}.$$

The pattern above shows the concatenated bar representation for the number 204, clearly distinguishing the narrow and wide bars that form the numeric subset of Code 39.

8. Fictional Languages

Literature and cinematography is full of invented languages and alphabets, just think about dwarf language in The Lord's Ring or the Galactic basic language in Star Wars. These alphabets exists and are well known and documented, but if someone never saw or read the operas in which this symbols are involved, it is very difficult to search the source. Languages like these can so also be used to hide information. In particular lots of fictional alphabets have been created in the occultism field but if they are fictional or real, is a open question. The following is only a list of some fictional alphabets with a really short description of their usages.

8.1 Occult Languages

The occult term refers to the "knowlegde of the hidden" or to the "knowledge of the paranormal", in contrast to common "knowledge to the measurable". Even if occult movements have always existed, the term was coined in France only during the 19th Century. Occultists love to hide their messages in fictional languages, claiming that they use them to talk with some divinity or ancestral force.

8.1.1 Malachim Alphabet (16th Century)

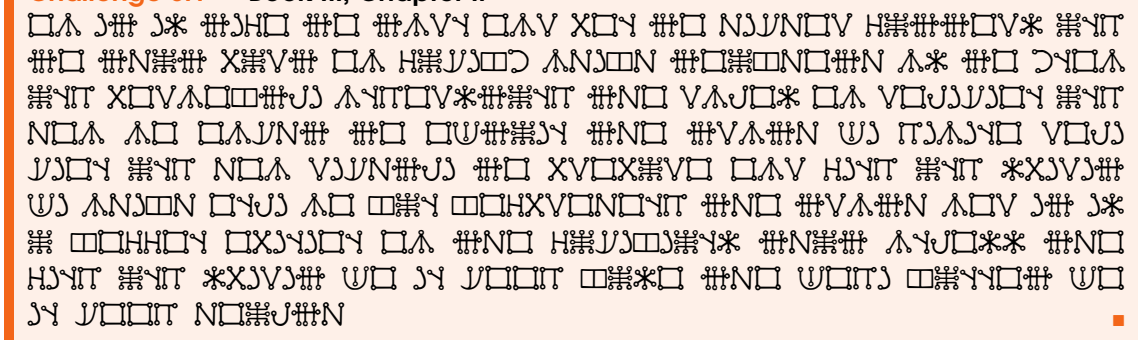
The Malachim alphabet was published for the first time by Heinrich Cornelius Agrippa, A German polymath theologian and occult writer, in 16th century, in book III, Chapter XXX of his "Occult Philosophy". Agrippa was considered the prince of black magicians, but he managed to escape the Inquisition under the protection of Hermann of Wield, the archibishop of Cologne. Due to the pressures from Inquisition, the publications of Occult Philosophy were stopped. Literally Malachim is the Hebrew word that means "angels" or "messengers" and Agrippa comments the alphabet as "of Angels or Regal". The alphabet was widely used in the Freemasons community. Curiously, in the last years of its life, Agrippa sincerely rejected the magic in its totality. In fact in the third book of Occult Philosophy he concludes:

But of magic I wrote whilst I was very young three large books, which I called Of Occult Philosophy, in which what was then through the curiosity of my youth erroneous, I now being more advised, am willing to have retracted, by this recantation; I formerly spent much time and costs in these vanities. At last I grew so wise as to be able to dissuade others from this destruction. For whosoever do not in the truth, nor in the power of God, but in the deceits of devils, according to the operation of wicked spirits presume to divine and prophesy, and practising through magical vanities, exorcisms, incantions and other demoniacal works and deceits of idolatry, boasting of delusions, and phantasms, presently ceasing, brag that they can do miracles, I say all these shall with Jannes, and Jambres, and Simon Magus, be destinated to the torments of eternal fire.

a	b	c	d	e/o	f/u/v/w	g	h	i/j/y	l
Aleph	Beth	Cheth	Daleth	Ayn	Vau	Gimel	He	Yod	Lamed
									

m	n	p	q	r	s	t	x	z
Mem	Nun	Pe	Quph	Resh	Samech	Tau	Tzaddi	Zain
								

Challenge 8.1 — Book III, Chapter I.

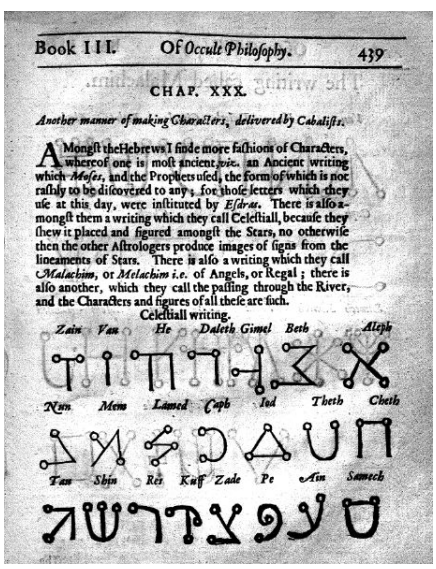


Solution 8.1 — Book III, Chapter I.

Ow it is time to turn our pen to higher matters, and to that part of Magick which teacheth us to know and perfectly understand the rules of Religion, and how we ought to obtain the truth by Divine Religion, and how rightly to prepare our mind and spirit, by which only we can comprehend the truth; for it is a common opinion of the Magicians, that unless the mind and spirit be in good case, the body cannot be in good health.

8.1.2 Transitus Fluvii and Celestial Alphabet (16th Century)

Agrippa creates other two occult languages similar to Malachim: The Transitus Fluvii and the Celestial alphabet. The first has been created around 1510 and officially published by Agrippa in 1533 in "The third book of Occult Philosophy", even if it can also be found in some priors operas like "de Balmis Peculium Abrae. Grammatica hebraea una cum latino" (1523), "Champ Fleury" (1529) and "Voarchadumia contra alchimiam" (1530). The second alphabet, also called Angelic alphabet, was invented to "communicate" with angels and was published in the same book of the Transitus Fluvii. Later it has been used in pagans, witches and wizards rituals. The following pictures are the original pages in the book where the alphabet appear:


Book III. Of Occult Philosophy. 439

CHAP. XXX.

Another manner of making Characters, delivered by Cabalists.

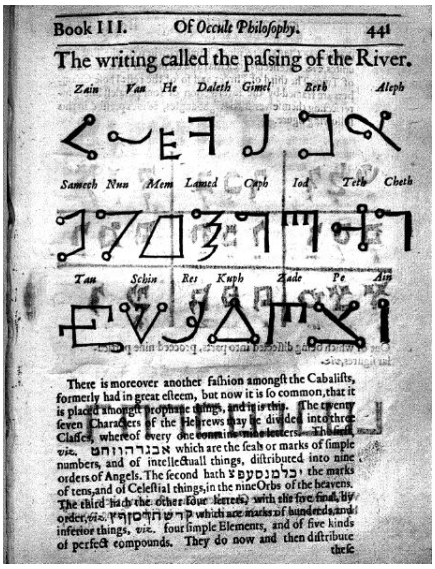
Amongst the Hebrews I finde more fashions of Characters, whereof one is most ancient, *viz.* an Ancient writing which *Moses*, and the Prophets used, the form of which is not rashly to be discovered to any; for those letters which they use at this day, were instituted by *Esdras*. There is also amongst them a writing which they call Celestiall, because they (hew it placed and figured amongst the Stars, no otherwise then the other Astrologers produce images of signs from the lineaments of Stars. There is also a writing which they call *Malachim*, or *Melachim i.e.* of Angels, or Regal; there is also another, which they call the passing through the River, and the Characters and figures of all these are such.

Celestiall writing.

Zain Van He Daleth Gimel Beth Aleph

Nun Mem Lamed Caph Iod Theth Cheth

Tau Shin Res Kuf Zade Pe Ain Samech


Book III. Of Occult Philosophy. 441

The writing called the passing of the River.

Zain Van He Daleth Gimel Beth Aleph

Samech Nun Mem Lamed Caph Iod Teth Cheth

Tau Schin Res Kuph Zade Pe Ain

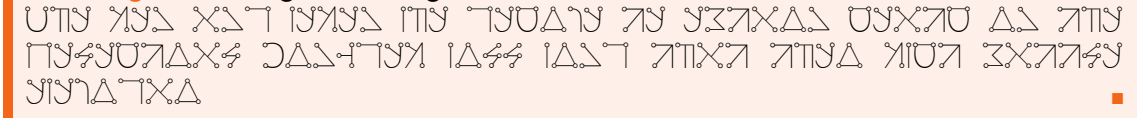
There is moreover another fashion amongst the Cabalists, formerly had in great esteem, but now it is so common, that it is placed amongst prophane things, and is this. The twenty seven Characters of the Hebrews may be divided into three Classes, whereof every one contains nine letters. The first, *viz.* אבגדהוזחט which are the seals or marks of simple numbers, and of intellectuall things, distributed into nine orders of Angels. The second hath יכלמנסעצ the marks of tens, and of Celestial things, in the nine Orbs of the heavens. The third hath the other four letters, with the five final, by order, *viz.* קרשתךמסנף which are marks of hundreds, and inferior things, *viz.* four simple Elements, and of five kinds of perfect compounds. They do now and then distribute these

Follows a table that summarize the Celestial alphabet:

a	b/v	c	d	e/o	f/u/w	g	h	i/j/y	l
Aleph	Beth	Cheth	Daleth	Ayn	Vau	Gimel	He	Yod	Lamed
									

m	n	p	q	r	s	t	x	z
Mem	Nun	Pe	Quph	Resh	Samech	Tau	Tzaddi	Zain
								

Challenge 8.2 — Brigham Young.

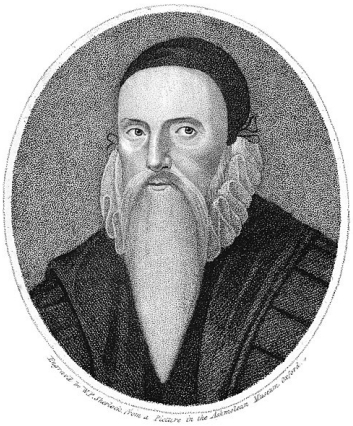


■

Solution 8.2 — Brigham Young.

The men and women, who desire to obtain seats in the celestial kingdom, will find that they must battle everyday.

8.1.3 Enochian Alphabet (16th Century)



The Enochian language, or the language of the angels, appeared for the first time in 16th century in the private journals of John Dee and Edward Kelley, an occult philosopher and a spirit medium who collaborate in magical investigations. Dee and Kelley claimed that the language came directly from the Enochian angels which Dee met during one of their scryer rituals. Enochian language is supposed to be the language used by Adam to communicate with angels and with God, but after his fall from paradise he lost that particular language and started building the Hebrew based on some recall of Enochian. The Enochian language has never been used in the known history with the exception of the patriarch Enoch (which where the name probably come from), who wrote the "Book of Loagaeth", lost during the Deluge of Noah. Starting from the 26 March 1583 Dee and Kelley, during their spirit sections, collected lots of angelic messages that became 2 books ("Liber Logareth" and "Mysteriorum Libri Quinque") and some manuscripts. The language syntax itself is very close to English and also the semantic is very similar, far away from the Hebrew, the language that it is supposed to be based on Enochian. The set of letters and their names are summarized in the following table:

a	b	c/k	d	e	f	g/j	h	i/y	l	m
Un	Pa	Veh	Gal	Graph	Or	Ged	Na	Gon	Ur	Tal

n	o	p	q	r	s	t	u/v/w	x	z
Drux	Med	Mals	Ger	Don	Fam	Gigs	Van	Pal	Ceph

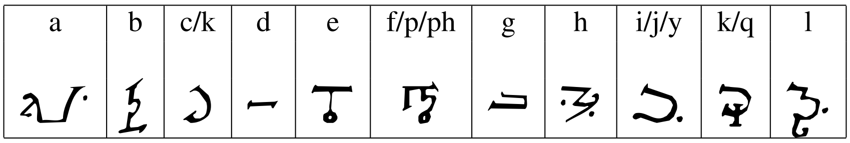
Challenge 8.3 — Nalvage Angel. 𝔏 𝔵𝔈 𝔑𝔐𝔗𝔈𝔗𝔵𝔏𝔈𝔗 𝔑𝔏 𝔏𝔇𝔗𝔙𝔈𝔄𝔅𝔑 𝔵𝔇𝔛 𝔏𝔇𝔗𝔵𝔏𝔈𝔈 𝔗𝔏𝔄 𝔵𝔅𝔅𝔏𝔈𝔛𝔏𝔇𝔍 𝔑𝔏 𝔗𝔏𝔄𝔈 𝔍𝔏𝔅𝔑𝔈𝔗𝔇𝔗 𝔛𝔗𝔉𝔏𝔄𝔗𝔈𝔗𝔛 𝔄𝔐𝔗𝔅𝔐 𝔗𝔗 𝔅𝔏𝔇𝔑𝔵𝔏𝔇𝔗𝔛 𝔏𝔇 49 𝔑𝔵𝔙𝔉𝔗𝔗. 𝔏𝔇 49 𝔄𝔏𝔗𝔅𝔗𝔗 𝔏𝔈 𝔅𝔵𝔊𝔉𝔏𝔇𝔍𝔗 𝔄𝔐𝔗𝔅𝔐 𝔵𝔈𝔗 𝔑𝔐𝔗 𝔇𝔵𝔑𝔄𝔈𝔵𝔊 𝔅𝔗𝔗𝔗 𝔑𝔏 𝔏𝔒𝔗𝔇 𝔑𝔐𝔏𝔗𝔗 𝔇𝔏𝔑 49 𝔙𝔄𝔑 48 𝔵𝔏𝔈 𝔏𝔇𝔗 𝔗𝔗 𝔇𝔏𝔑 𝔑𝔏 𝔙𝔗 𝔏𝔒𝔗𝔇𝔛𝔗 𝔍𝔵𝔑𝔗𝔗 𝔏𝔵 𝔇𝔇𝔛𝔗𝔈𝔙𝔵𝔇𝔛𝔗𝔍 𝔄𝔐𝔗𝔈𝔙𝔗 𝔗𝔏𝔄 𝔗𝔐𝔵𝔊𝔊 𝔐𝔵𝔄𝔗 𝔅𝔇𝔏𝔄𝔊𝔗𝔛𝔍𝔗 𝔑𝔏 𝔈𝔏𝔄𝔗 𝔗𝔄𝔗𝔈𝔗 𝔍𝔵𝔑𝔗 ■

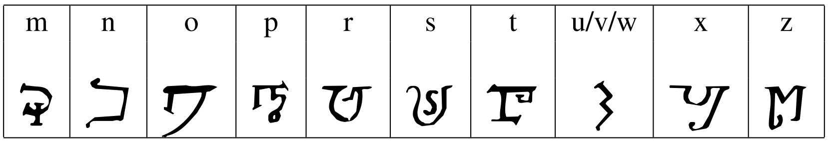
Solution 8.3 — Nalvage Angel. I am therefore to instruct and inform you, according to your Doctrine delivered, which is contained in 49 Tables. In 49 voices, or callings: which are the Natural Keys to open those, not 49 but 48 (for one is not to be opened) Gates of Understanding, whereby you shall have knowledge to move every Gate

8.1.4 The alphabet of the Magi (16th Century)

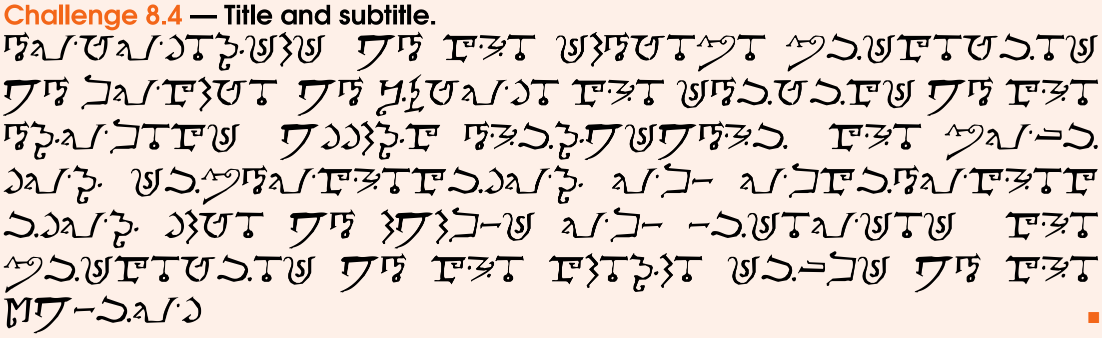
The alphabet of the Magi is a variant of the Hebrew Alphabet proposed by the italian humanist Theseus Ambrosius, one of the fathers of the Christian Kabbalah. The alphabet was invented by the swiss physician, alchemist and astrologer Philippus Aureolus Theophrastus Bombastus von Hohenheim, knows as Paracelsus, that used it to engrave names of Angels to its protection talismans. The alphabet was then cited by Claude Duret in "Thresor" (1613) under the name of "the characters of the angel Raphael", by Edmund Fry in "Pantographia" (1798) and by S.L. MacGregor Mathers in "Key of Solomon" (1888) that first named it "The alphabet of the Magi". For its opera Fry kept the name given by Duret, specifying that:

Theseus Ambrosius asserts that this character was brought from Heaven by the Angel Raphael by who it was communicated to Adam who used it in composing Psalms after his expulsion from the terrestrial paradise. Some authors pretend that Moses and the prophets used this letter and that they were forbidden to divulge it to mortal man



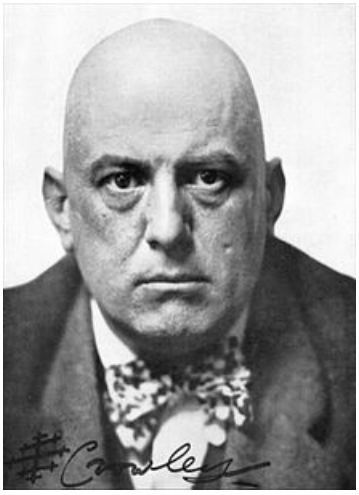


Challenge 8.4 — Title and subtitle.



Solution 8.4 — Title and subtitle.
Paracelsus. Of the supreme mysteries of nature. Of [brace] the spirits of the planets. Occult philosophy. The magical, sympathetical, and antipathetical cure of wounds and diseases. The mysteries of the twelve signs of the zodiac.

8.1.5 Daggers Alphabet (1909)

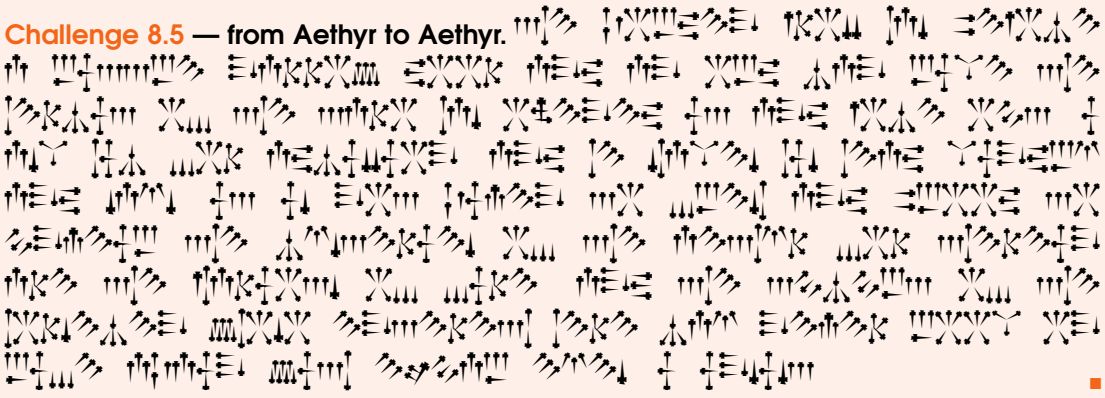


The daggers alphabet appeared for the first time in the book "The Vision and the Voice" by the famous occultist Aleister Crowley. The book was written during its trip in Mexico and algeria and follows the journey of Crowley through the 30 Enochian Æthyrs, first developed by John Dee and Edward Kelley 200 years before. The book is also a source of many of the spirictual doctrines of Thelema, the religion that Crowley invented. The alphabet is cited in the chapter named "The Cry of the 19th Aethyr" and it is introduced by:

At first there is a black web over the face of the stone. A ray of light pierces it from behind and above. Then cometh a black cross, reaching across the whole stone; then a golden cross, not so large. And there is a writing in an arch that spans the cross, in an alphabet in which the letters are all formed of little daggers, cross-hilted, differently arranged. And the writing is: Worship in the body the things of the body; worship in the mind the things of the mind; worship in the spirit the things of the spirit. This holy alphabet must be written by sinners, that is, by those who are impure.

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
												

Challenge 8.5 — from Aethyr to Aethyr. 

Solution 8.5 — from Aethyr to Aethyr. The golden cross has become a little narrow door, and an old man like the Hermit of the Taro has opened it and come out. I ask him for admission: and he shakes his head kindly, and says: It is not given to flesh and blood to unveil the mysteries of the Aethyr, for therein are the chariots of fire. and the tumult of the horsemen; whoso entereth here may never look on life again with equal eyes. I insist.

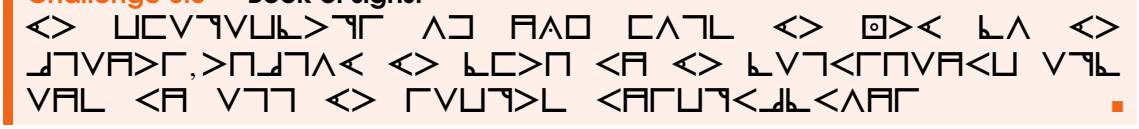
8.1.6 Nug-Soth (1922)

Howard Phillips Lovecraft, an American writer, first introduce in its short story "The Hound", a grimore book called "Necronomicon", and cited then in other histories and by other authors. The fictional history of the Necronomicon was published in 1938 by Lovecraft. It claimed that the book, written near 730 by the "half-crazed Arab" Abdul Alhazred, was originally named "Al Azif". After the mysterious death of the author in 738 the book was translated in greek in 930 by Theodorus Philetas that gave the actual name Necronomicon. The first latin translation have been done by Olaus Wormius in 1228 (even if the real Olaus Wormius born only in 1588). Both latin and greek versions of the book were banned by Pope Gregory XI in 1232 even if some versions of the book appeared later in the 15th century in Germany, in 17th century in Spain and in 16th century in Italy. In the fictional history of the book is also cited John Dee (see Enochian alphabet above) who seems to have translated it into English. Few secret copies of the book are currently around the world. The story of the book was so successful that different authors wrote their personalized version of the Necronomicon. One of those versions, published by Avon, contained the Nug-Soth alphabet, spoken by a magician that lived around 16000 A.D. whom mind, in the story "The shadows of time" of Lovecraft, was transferred in a member of the Great Race of Yith in 150000000 B.C. The alphabet contains the following symbols:

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	0	1	2	3	4	5	6	7	8	9
																	

Challenge 8.6 — Book of signs.



Solution 8.6 — Book of signs.

Ye characters of Nug hold ye key to ye planes, employ ye them in ye talismanic art and in all ye sacred inscriptions

8.2 Messages from the Space

8.2.1 Aurebesh (Star Wars)

Aurebesh is the official writing system used to represent the spoken Galactic Basic language in the Star Wars franchise. In the Star Wars chronology the alphabet was introduced by the Rakatan Infinite Empire during its reign and was widely used during the Alsakan Conflicts 17000 years before the Battle of Yavin. It is documented also that an identical alphabet was in use by the inhabitants of the Settled Worlds 25793 years before the Battle of Yavin. The font is inspired from the shapes designed by Joe Johnston for the original trilogy that compare in the Return of the Jedi and later in Attack of the Clones. Starting from those symbols in 1990s Stephen Crane developed an alphabet to include in the Star Wars games to allow players to give a name to their characters. He then gives the official name to the alphabet that is composed by the name of the first two letters of the alphabet: "Aurek" and "Bash". About Aurebesh, Crane wrote:

"The Aurebesh is a lot like Boba Fett. It is a facet of the Star Wars phenomenon that had its origin as a cinematic aside, but which has come to be widely embraced, far out of proportion to its humble origins."

One of the most famous scene in which Aurebesh compare in clear sight is during The Phantom Menace, when Anakin, in the Battle of Naboo reads the following message sent by R2-D2: ANAKIN TURN THE SHIP AROUND AND GO BACK HOMIE RIGHT AWAY.

a	b	c	d	e	f	g	h	i	j	k	l	m
Aurek	Besh	Cresh	Dorn	Esk	Forn	Grek	Herf	Isk	Jenth	Krill	Leth	Mern

n	o	p	q	r	s	t	u	v	w	x	y	z
Nern	Osk	Peth	Qek	Resh	Senth	Trill	Usk	Vev	Wesk	Xesh	Yirt	Zerek

0	1	2	3	4	5	6	7	8	9	.	,	?	!	:	;	(	)	-	/

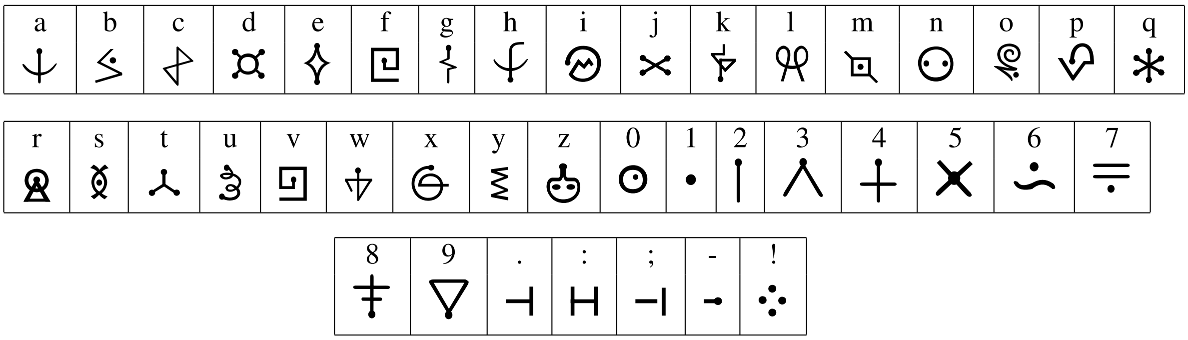
Challenge 8.7 — Light Speed? Really?. it is the ship that made the kessel run in less than twelve parsecs. i have outrun imperial starships. not the local bulk cruisers, mind you. i am talking about the big corellian ships. now. she is fast enough for you, old man. ■

Solution 8.7 — Light Speed? Really?. it is the ship that made the kessel run in less than twelve parsecs. i have outrun Imperial starships. Not the local bulk cruisers, mind you. i am talking

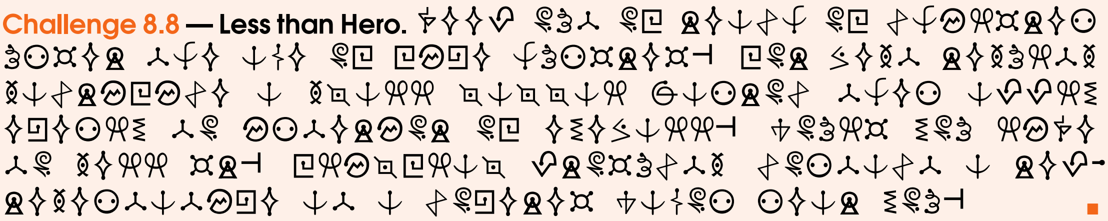
> about the big corellian ships, now. she is fast enough for you, old man.

8.2.2 Futurama Alien Alphabet (Used in Futurama cartoon)

Futurama is a fiction sitcom created by Matt Groening in in 1999 that follows the adventures of Philip J. Fry who has been cryogenically preserved for 1000 years. The earth of the future is populated by races coming from different places of the universe. In the sitcom appears, occasionally, some writes in the background, written in some alien language. Writings are usually jokes like "Rent a human" or "Tasty human burgers" or again "Don't drink the emperor" and "Obey Mom". There exists 2 different languages: AL1 and AL2. AL1 is listed below.

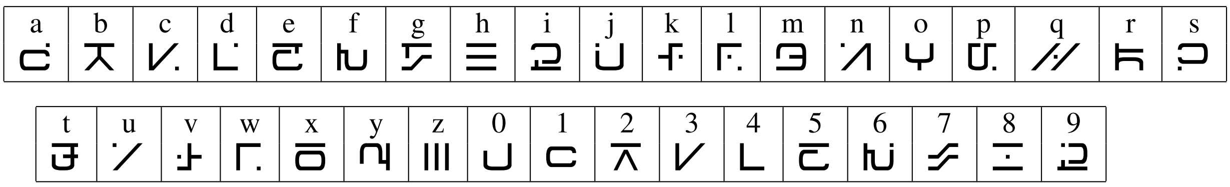


Challenge 8.8 — Less than Hero.

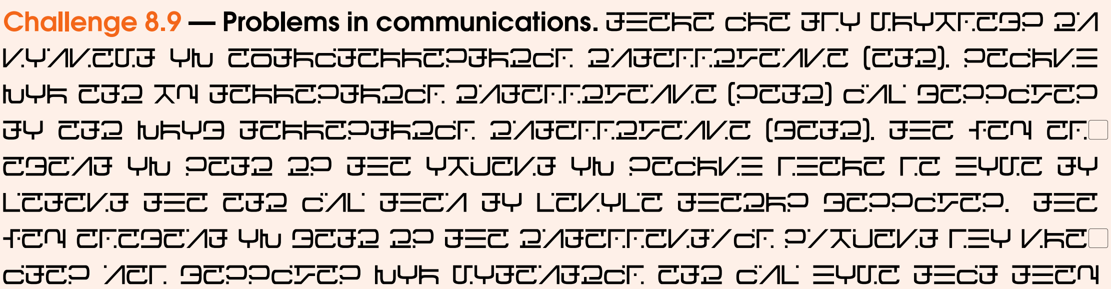


Solution 8.8 — Less than Hero. keep out of reach of children under the age of five hundred. for best results, sacrifice a small mammal xanroc, then apply evenly to interior of eyeball. would you like to sell dr. flimflam products? contact a representative at a covered wagon near you.

8.2.3 Visitor (Used in TV series of 2009)

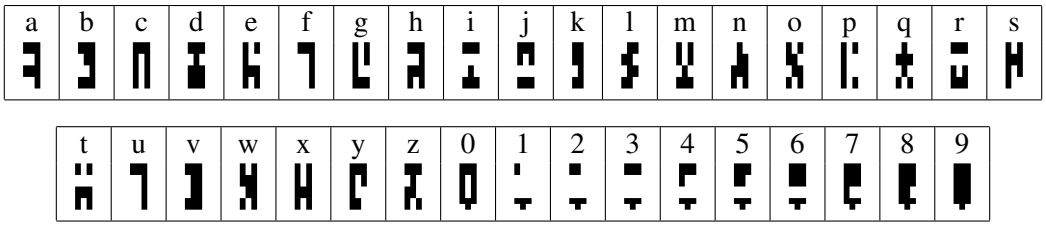


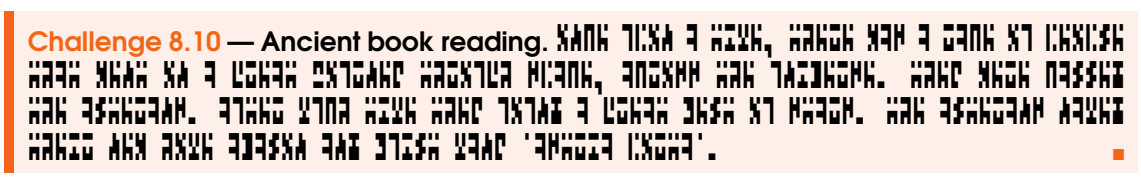
Challenge 8.9 — Problems in communications.



Solution 8.9 — Problems in communications. There are two problems in concept of extraterrestrial intelligence (ETI). Search for ETI by terrestrial intelligence (SETI) and Messages to ETI from terrestrial intelligence (METI). The key element of SETI is the Object of search where we hope to detect the ETI and then to decode theirs Messages. The key element of METI is the intellectual Subject who creates new messages for potential ETI and hope that They will detect and perceive these Messages.

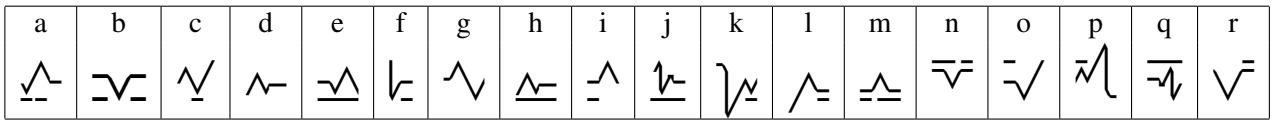
8.2.4 Ancient Alphabet (Used in TV series "Stargate")

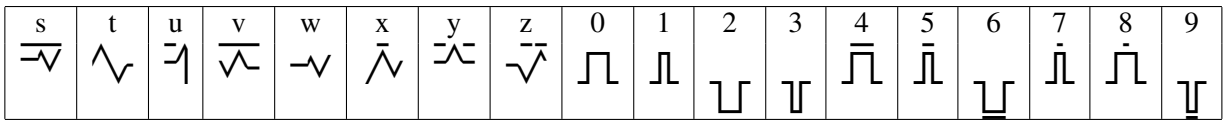


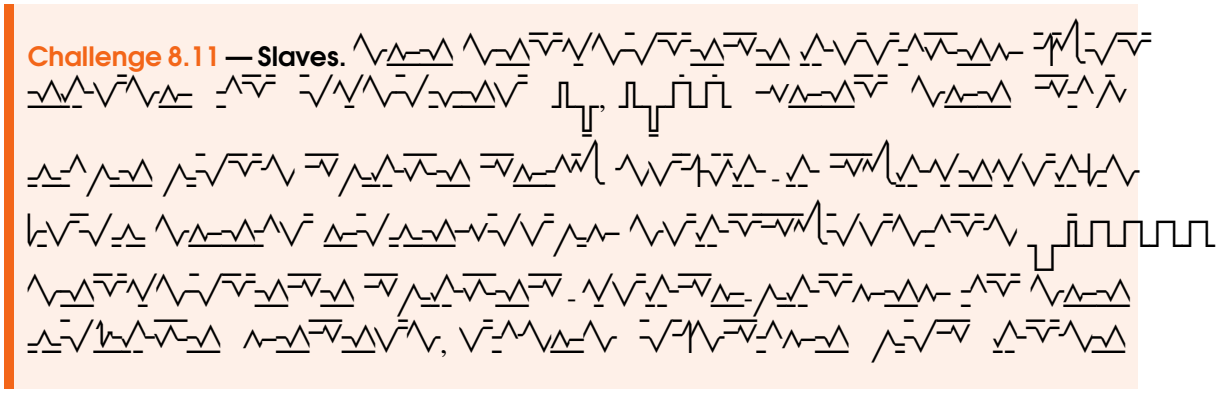


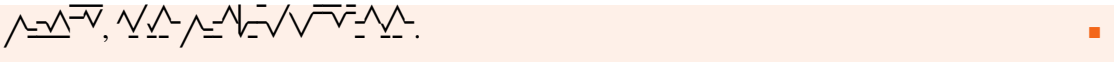
Solution 8.10 — Ancient book reading. Once upon a time, there was a race of people that went on a great journey through space, across the universe. They were called the Alterans. After much time they found a great belt of stars. The Alterans named their new home Avalon and built many 'Astria Porta'.

8.2.5 Tenctonese (Alien Nation)









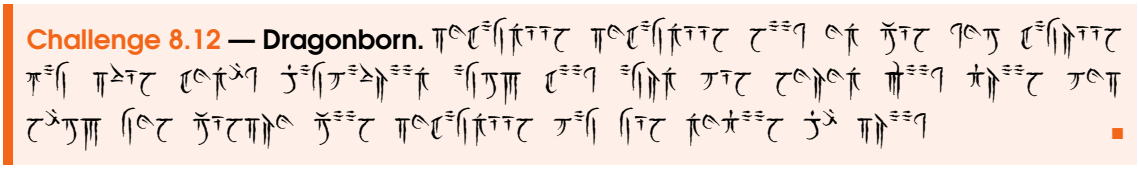
Solution 8.11 — Slaves. The Tenctonese arrived upon Earth in October 19, 1988 when the six-mile-long slave ship Gruza - a spacecraft from their homeworld transporting 250,000 Tenctonese slaves - crash-landed in the Mojave desert, right outside Los Angeles, California.

8.3 Fantasy Messages

8.3.1 Iokharic (Draconic Language)

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	0	1	2	3	4	5	6	7	8	9
															

Challenge 8.12 — Dragonborn. 

Solution 8.12 — Dragonborn. Dovahkiin Dovahkiin naal ok zin los vahriin wah dein vokul mahfaeraak ahst vaal Ahrk fin norok paal graan fod nust hon zindro zaan Dovahkiin fah hin kogaan mu draal - A piece of Dragonborn song of TES5 Skyrim

8.3.2 Daedra (Language of the Daedra Race of Oblivion)

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	0	1	2	3	4	5	6	7	8	9

Challenge 8.13 — Dragonborn.

Solution 8.13 — Dragonborn. We do not die, We do not fear death. Destroy the Body, and the Animus is cast into The Darkness. But the Animus returns. But we are not all brave. We feel pain, and fear it. We feel shame, and fear it. We feel loss, and fear it. We hate the Darkness, and fear it.

8.3.3 Matoran (From LEGO Bionicles)

The language used by the Matoran is spoken in the majority of the Matoran Universe. The Great Beings used the language as programming language when constructing the Matoran Universe, and as such, the language would not be recognized to most reading or listening to it. Most characters in the BIONICLE world have adopted a standard writing system for the Matoran language, with letters and numbers both resembling circles with lines and smaller circles inside of them. Being a fairly ancient language, there are many different variations on how the language is written. On Voya Nui, the Matoran have a slight variation on the normal alphabet, using hexagons instead of circles, and the "K" being slightly different as well. This version may have originated on the Southern Continent when Voya Nui was still part of it. Ancient Matoran texts are illegible to anyone but few who have studied such ancient texts. Most text is transcribed onto stone tablets or painted onto a surface. The only known paper in the Matoran Universe is the ancient scroll left by the Great Beings inside the Great Temple.

	a	b	c	d	e	f	g	h	i	j
Metru Nui										
Voya Nui										

	k	l	m	n	o	p	q	r	s	t
Metru Nui										
Voya Nui										

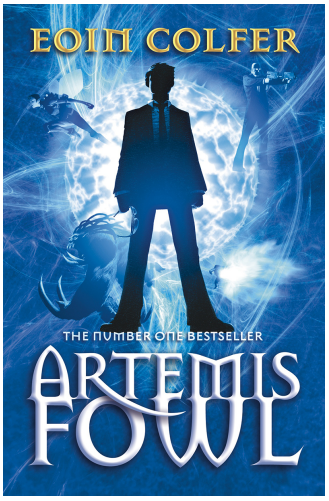
	u	v	w	x	y	z	0	1	2	3
Metru Nui										
Voya Nui										

	4	5	6	7	8	9
Metru Nui						
Voya Nui						

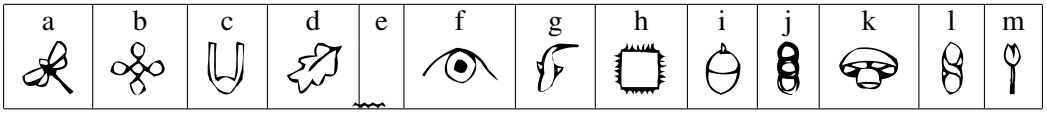
8.4 Do you read?

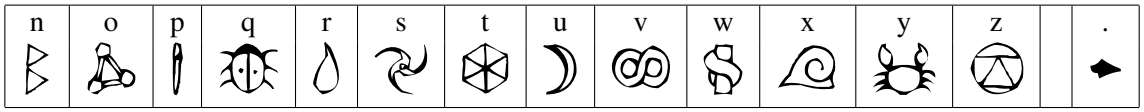
8.4.1 Gnommish (Artemis Fowl)

Gnommish is a language that appears in the Artemis Fowl series, written by Eoin Colfer. It is believed to be the language used by fairies. The script appears at the bottom of each page of the most of the books in the saga. When Artemis first analyzed the language he mentioned that some symbols can be compared to Egyptian hieroglyphs and he supposed to be much older. Gnommish is originally described as a mix of symbolic and alphabetic letters, running in spirals ("but since reading in spirals gives most fairies migraines, most modern fairy script is arranged in horizontal lines"). Each Gnommish symbol corresponds to a single letter in the English alphabet. Even if the alphabet is perfectly mapped to English words, there are three words that are Gnommish specific:



- D'Arvit, word to replace real profanity, usually not possible to translate and subject to censure;
- Cowpog, that can be translated as "moron";
- Ffurfor, that means plunderer.



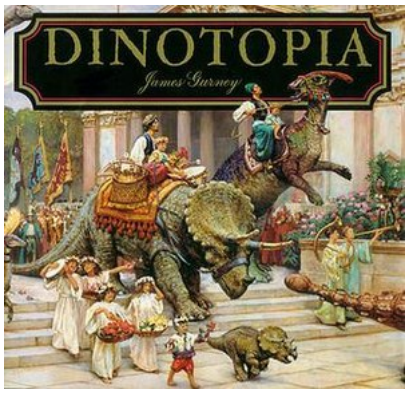


Challenge 8.14 — Book Seven.

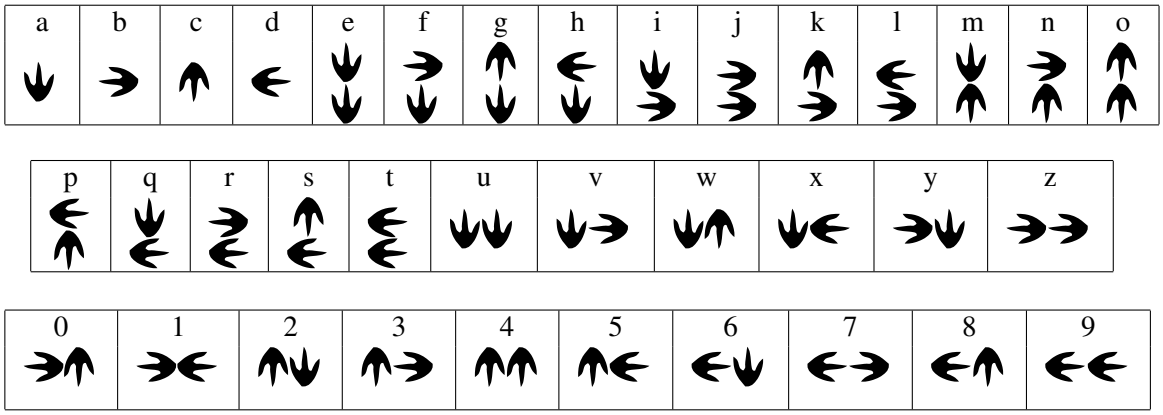
Solution 8.14 — Book Seven.

People called me a boy genius. A wunderkind. Perhaps I was a prodigy. But I will be fifteen soon and too old for that label. So what am I then? A teenage criminal mastermind perhaps. Or just a common thief. Who can a thief trust? There were a few I thought. But could I have been wrong? Is that possible?

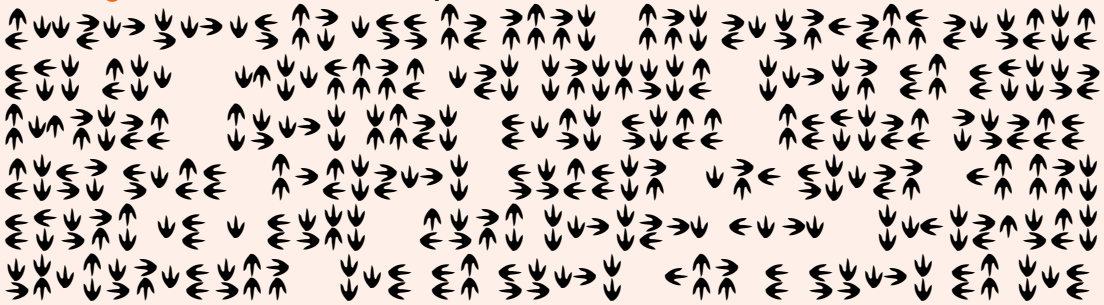
8.4.2 Dinotopian (From Dinotopia Fantasy Books)



Dinotopia is a series of illustrated fantasy books, created by author and illustrator James Gurney. It is set in the titular "Dinotopia", an isolated island inhabited by shipwrecked humans and sapient dinosaurs who have learned to coexist peacefully as a single symbiotic society. The first book was published in 1992 and has "appeared in 18 languages in more than 30 countries and sold two million copies." Dinotopia: A Land Apart from Time and Dinotopia: The World Beneath both won Hugo awards for best original artwork. Since its original publication, over twenty Dinotopia books have been published by various authors to expand the series. A live-action TV miniseries, a brief live-action TV series, an animated film, and several video games have also been released. The Dinotopian Alphabet was used mostly by three-toed dinosaurs, which were the best scribes: they wrote anything from warnings to jokes. The scribes, while "writing", act more like dancers; hopping, twisting and stepping to make the right marks



Challenge 8.15 — Code of Dinotopia.

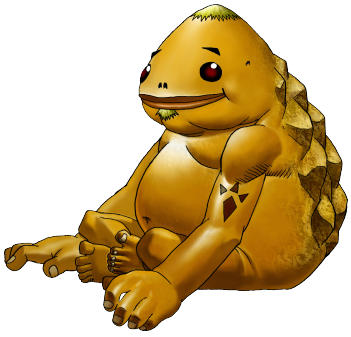


■

Solution 8.15 — Code of Dinotopia.
Survival of all or none. One raindrop raises the sea. Weapons are enemies, even to their owners. Give more, take less. Others first, self last. Observe, listen, and learn. Do one thing at a time. Sing every day. Exercise imagination. Eat to live, don't live to eat.

8.5 Do you play games?

8.5.1 Goron Alphabet



The Gorons are a recurring race of rock people inhabiting Hyrule, Termina, and several other countries, typically living in mountain ranges, in the Legend of Zelda series. Their legs are really short, especially in comparison to their much longer arms. Their skin is usually light brown, their eyes are small and they have marked lips with square teeth. Another common trait are their symbols in the shape of the Goron's Ruby tattooed to the sides of their arms, whether this is natural or not is unknown. On the back they usually have some stones and boulders and the body shape is usually rounded. Goron, in the Zelda series, is a spoken language only that combines a series of grunts and groans. While it is adapted to longer conversation, Goron is usually kept short and to the point. there is not an official alphabet among the Goron population, but some fans made it collecting some scripts found in the various Zelda games. Some of the letter of the alphabet can be encoded with a double symbol. Usually, if in the word a letter appears more than one time and if the letter has a double encoding, the both encoding are used alternates.

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	0	1	2	3	4	5	6	7	8	9	!	?	.	,

:	;	-	(	)	’

Challenge 8.16 — Ocarina of Time.

■

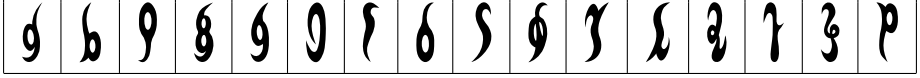
Solution 8.16 — Ocarina of Time.

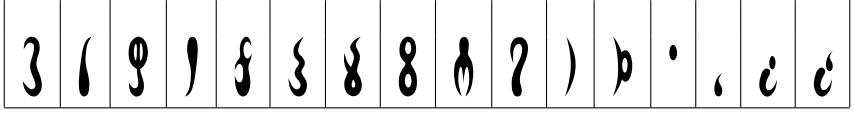
How could you do this to me? You, you're GanondOrf's servant! Hear my name and tremblE! I am Link! Hero of the GorOns!

8.5.2 Gerudo Alphabet

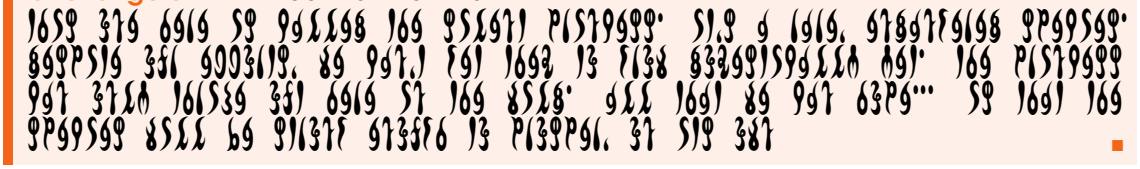
The Gerudo's alphabet is the written language of a indigenous thieves race inside the "The Legend of Zelda" series. The Gerudo lives in harsh desert and they are known to be feared and respected warriors. The race consists only in females, and only one male born into the tribe every 100 years that will play the role of the king for the whole epoch. When there is no king, the Gerudo are usually led by a chief with the position passed on from mother to daughter. The exact nature of this phenomenon has never been explained. Due to the lack of males boyfriends are taken from other human races.

The Gerudo script first appeared in "Ocarina of Time" and consists of 26 different characters that map exactly into the modern Latin alphabet. There are only two numerical symbols (1 and 5) and four punctuation marks (.,!?):

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	1	5	.	,	!	?
															

Challenge 8.17 — Breath of the Wild.

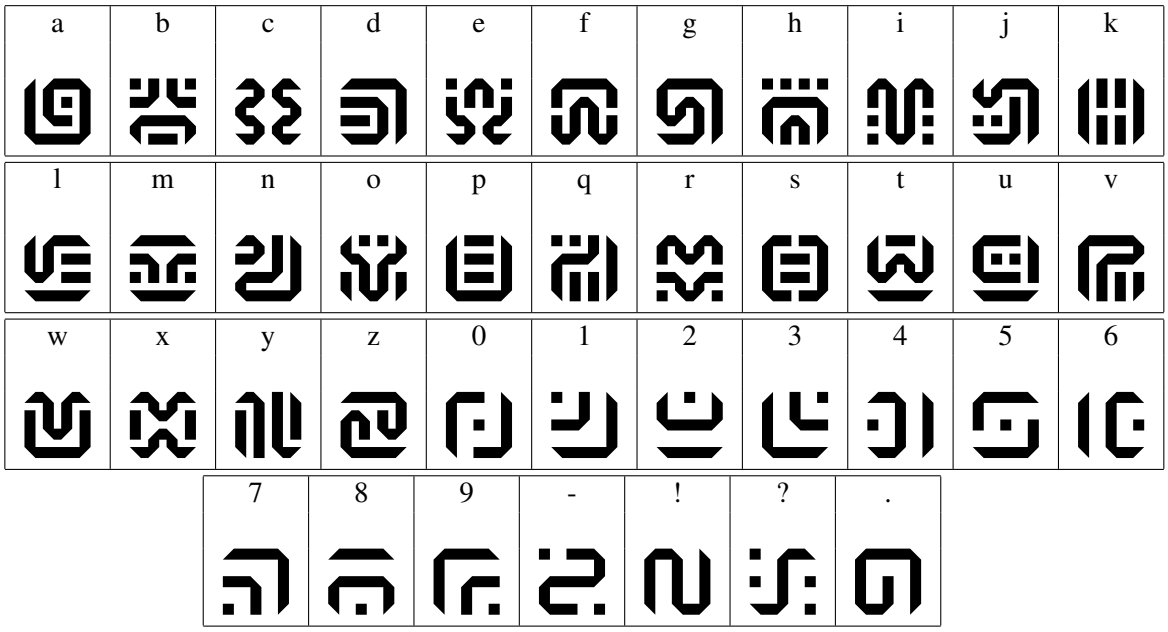


Solution 8.17 — Breath of the Wild.

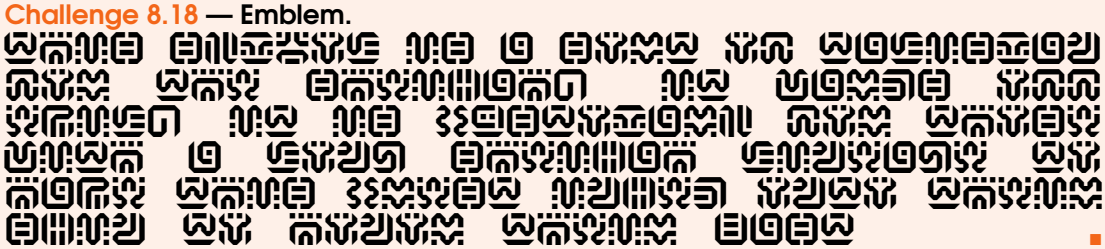
This one here is called the silent princess. It's a rare, endangered species. Despite our efforts, we can't get them to grow domestically yet. The princess can only thrive out here in the wild. All that we can hope... is that the species will be strong enough to prosper, on its own

8.5.3 Sheikah Alphabet

The Sheikah are another race appearing in "The Legend of Zelda" series. Also known as the "Shadow Folk", they are a humanoid mysterious tribe with magic abilities. Even if the race doesn't appear frequently in the games, they are known to have red eyes and pointed ears. The Sheikah have their own writing script that can be extensively found on the ancient architectures and artifacts. It is a 1 to 1 map with the Latin alphabet. Each symbol composing the alphabet present some kind of symmetry in a square shape.



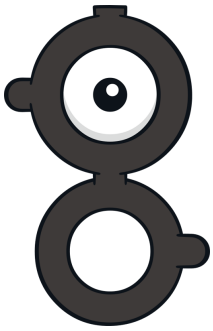
Challenge 8.18 — Emblem.



Solution 8.18 — Breath of the Wild.

This symbol is a sort of talisman for the Sheikah. It wards off evil. It is customary for those with a long Sheikah lineage to have this crest inked onto their skin to honor their past

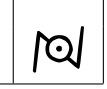
8.5.4 Unown Alphabet (From Pokemon)



All know Pokémon, The Japanese franchise created by Satoshi Tajiri in 1995. Pokèmon became so popular that the name of the characters have been used also in the scientific domain. For example there exists animals such as Stentorceps weedlei (named after the Pokemon Weedle) and Aerodactylus scolopaciceps (named after the Pokemon Aerodactyl). There is also a protein named after Pikachu, called Pikachurin. During years many different generations of different Pokémon came out, reaching more than 800 pocket monsters. A special Pokémon was created by Ken Sugimori in generation II in 1999 and its name is Unown. It's particularity is that can be found in different forms that can be easily mapped to the alphabet letters. In generation III, two more shapes have been introduced, the question mark and the exclamation mark, for a total of 28 different shapes. Unown is known to be a mysterious Pokémon that is usually found in ancient ruins in both Pokémon anime, manga and games. In these last places also some inscriptions can be found, written using the Unown alphabet. For example in the chapter 40 of Pokèmon Diamond and Pearl series the tablet below the Dialga statues is written with the Unown alphabets and says:

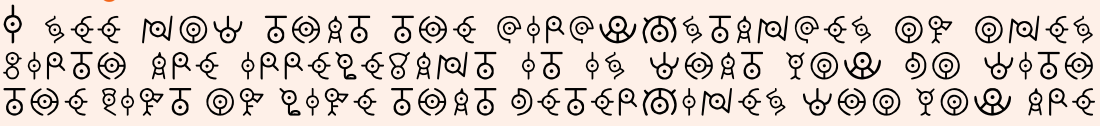
"FRIEND
SUBETE NO INOCHI HA
BETSU NO INOCHI TO DEAI
NANIKA WO UMIDASU"

That can be translated as "Friend, when every life meets another life, something will be born". The following is the full character mapping for the Unown 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	?	!
													

Challenge 8.19 — Mewtwo.



■

Solution 8.19 — Mewtwo.
I see now that the circumstances of one's birth are irrelevant; it is what you do with the gift of life that determines who you are.

8.5.5 Marker Symbols (From Dead Space)

Dead Space is a science fiction survival horror video game, set in a spacecraft and developed by EA Redwood Shores. The game has a strong science fiction atmosphere and is set in a spacecraft. During the game it is possible to encounter a bunch of symbols named "Marker Symbols". They are produced by the Markers: enigmatic double-helix shaped objects of extraterrestrial origin. The purpose of these objects are to create Necromorphs (the main antagonists of the Dead Space franchise) and finally a Moon. These scripts are frequently used by the Unitology religion that believe that the human race was created by the intelligent design of a divine alien agency, and will be reunified after death in Heaven through the power of a sacred artifact known as the Markers. The inhabitants of the spacecraft and Titan Station usually leave their thoughts written on the walls of the ship. Graffiti on the wall covered a variety of topics, such as survival tips, final messages, or often simply the survivors thoughts. Even if most of the symbols that can be found on the walls seem to be written in blood, some of them are written in marker pen.

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	1	2	3	4	5	6
															

7	8	9	10	ss	th	:	!	?
								

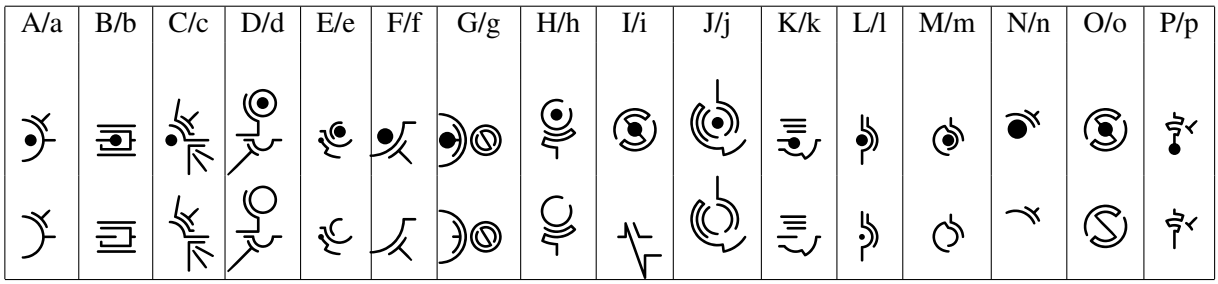
Challenge 8.20 — Believe.

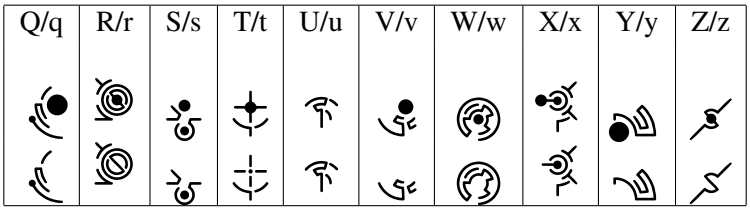
Solution 8.20 — Believe.

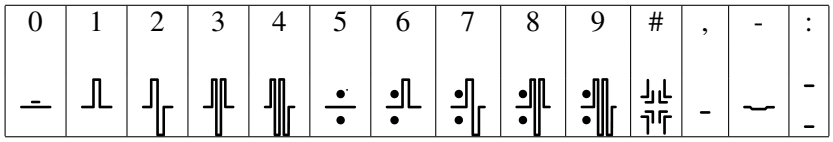
Keep us whole. Unity is forever. Death is only the beginning. keep us whole. Unity is forever. Let us be one.

8.5.6 Hymmnos Script (From Ar Tonelico)

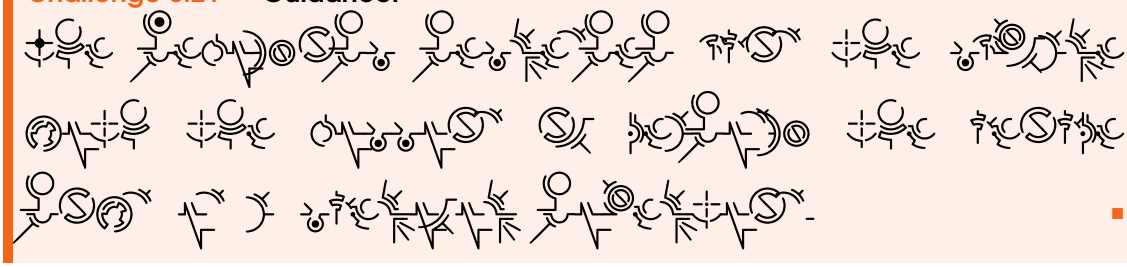
Akira Tsuchiya created the Hymmnos alphabet for the video game series Ar tonelico. According to the creator it's based in appearance upon one of the Sanskrit scripts. In the fictional world of Ar Ciel was frequent in the ancient times but it has fallen out common use. The Hymmnos language is notable for placing a heavy emphasis on the emotions of the speaker. A statement in Hymmnos can explicitly convey how the speaker feels about what they are speaking about. In Ar Ciel is mainly used by Reyvateils in order to communicate with their servers and by humans in order to control ancient machines.







Challenge 8.21 — Guidance.



Solution 8.21 — Guidance.
The Demigods descended upon the surface with the mission of leading the people down in a specific direction.

9. Useful encoding

Encoding is referred to the process of converting data into a format required for a number of information processing needs. To not be confused with the concept of encrypting that, differently, aim to hide the original information instead of transforming in order to maximize its coverage and usage.

9.1 Polybius Square (II sec. B.C.)

The Polybius checkerboard, or Polybius Square, is a simple encoding technique described in the opera "The Histories" of the Greek historian Polybius. In the opera the system was used to communicate messages from distance, like a modern telegraph system, using torches: a man that brings 5 torches in each hand is able to generate messages by rising each hand with the correct amount of torches. The method was invented to be used in Phryctoriae, towers built on the top of the mountains visible to next ones, by two Greek engineers from Alexandria: Cleoxenes and Democletus who called it pyrseia. The hidden message is build on the base of a plaintext and a 5x5 square where the alphabet letters are arranged. A modern variant of the grid is composed by 36 cells in order to represent all 26 Latin symbols and the digits from 0 to 9. Modifications of the Polybius square have been used in history to communicate, like the case of the knock code used by American prisoners in Vietnam. The polybius square is also the base of some cryptosystems like the Playfair, the ADFGX, the ADFGVX and the Bifid cipher. The followings are the classical representations in Greek and Latin alphabets

	1	2	3	4	5
1	Α	Β	Γ	Δ	Ε
2	Ζ	Η	Θ	Ι	Κ
3	Λ	Μ	Ν	Ξ	Ο
4	Π	Ρ	Σ	Τ	Υ
5	Φ	Χ	Ψ	Ω	

	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

Each letter of the message is replaced with 2 numbers that identify the row and the column in which the symbol appear on the table. So A will become 11, K will become 25, R will become 42 and POLYBIUS will become 35 34 31 54 12 24 45 43. To decode the the message is sufficient to extract from the table the symbols at row and column indicated by the numbers.

Challenge 9.1 — The Histories.
23 11 14 35 42 15 51 24 34 45 43 13 23 42 34 33 24 13 31 15 42 43 33 15 22 31 15 13 44 15 14 44 34 43 35 15 11 25 24 33 35 42 11 24 43 15 34 21 23 24 43 44 34 42 54 24 33 22 15 33 15 42 11 31 24 44 32 24 22 23 44 35 15 42 23 11 35 43 23 11 51 15 12 15 15 33 33 15 13 15 43 43 11 42 54 21 34 42 32 15 44 34 42 15 13 34 32 32 15 33 14 15 51 15 42 54 34 33 15 44 34 13 23 34 34 43 15 21 34 42 43 44 45 14 54 11 33 14 52 15 31 13 34 32 15 43 45 13 23 44 42 15 11 44 24 43 15 43 11 43 44 23 15 35 42 15 43 15 33 44 43 24 33 13 15 32 15 33 23 11 51 15 33 34 32 34 42 15 42 15 11 14 54 13 34 42 42 15 13 44 24 51 15 34 21 13 34 33 14 45 13 44 44 23 11 33 25 33 34 52 31 15 14 22 15 34 21 44 23 15 35 11 43 44 ■

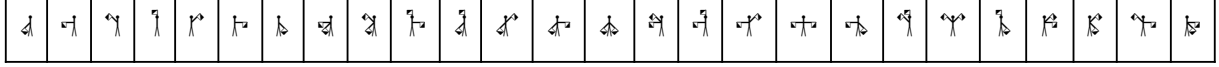
Solution 9.1 — The Histories.
Had previous chroniclers neglected to speak in praise of History in general, it might perhaps have been necessary for me to recommend everyone to choose for study and welcome such treatises as the present, since men have no more ready corrective of conduct than knowledge of the past.

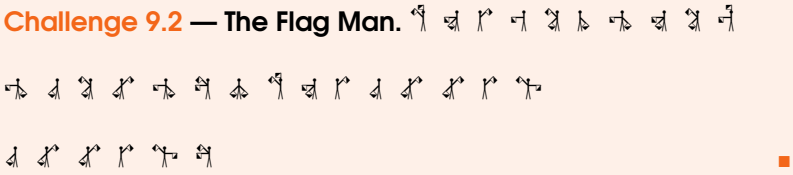
9.2 Semaphore (1794)

This is a simple method of visual encoding for communicate messages through signals, usually by pulsing lights or by flag positions. A notable communication system based on semaphores was invented by Claude Chappe in 1794. The system used a set of arms that were placed on the top of the defense towers. These arms were able to rotate around a fixed point autonomously, like the hands of a clock, allowing them to draw different figures, each with a specific meaning.

9.2.1 Flag Semaphore

In the 19th century a different version of the semaphore was used as a telegraphy system in maritime world and in particular to communicate between ships. This system reflects exactly the one invented by Chappe with the introduction of flags and human operators. Now largely abandoned it expected persons who held small flags in each hand, moving them to different angles to indicate letters of the alphabet or numbers:

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
																									

Challenge 9.2 — The Flag Man. 

Solution 9.2 — The Flag Man. The big ship sails on the alley-alley-o

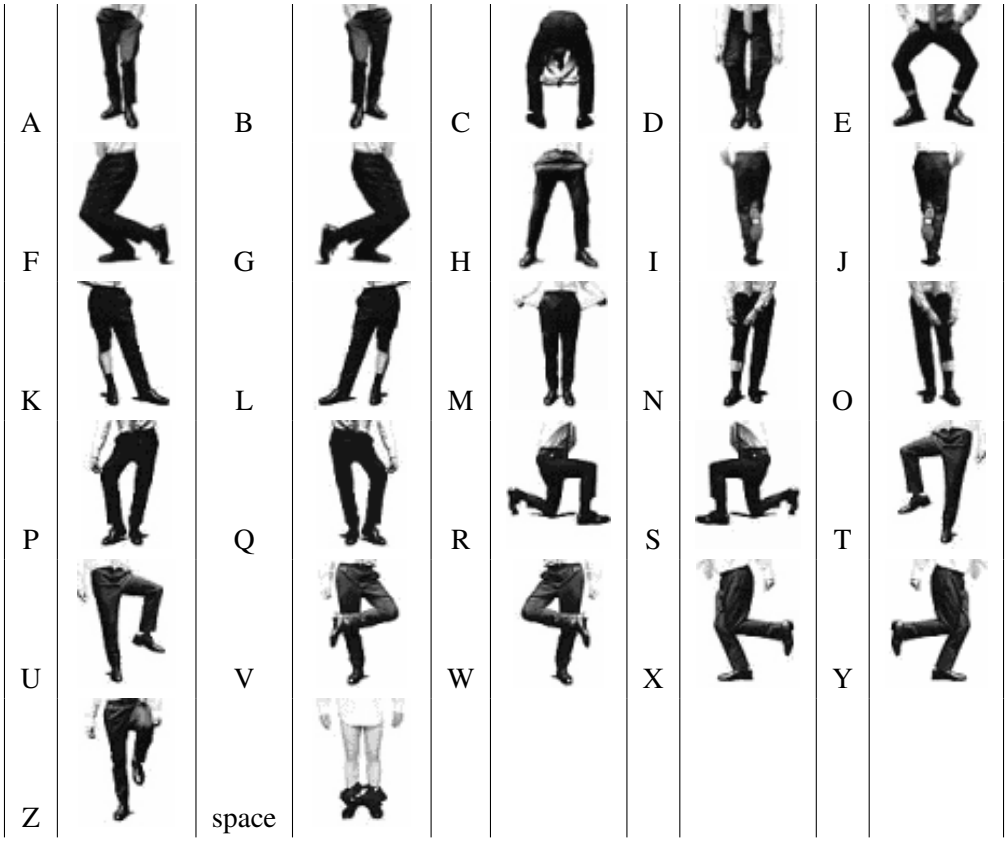
9.2.2 Trouser Semaphore

On the satiric "The Chap" magazine appeared a very nice variant of the semaphore encoding system. I will report the original article where it is descripted in a hilarious and brilliant way:

"In the previous issues of The Chap we (that is, you and I, gentle reader) have probed into the heady realm of Semiotics, wringing out and interpreting the inadvertent symbolism of body language and style affectation. This time around, we take a slightly different tack. Whilst Semiotics deals primarily with an unwitting divulgence of information, what I wish to concentrate on today is a far more calculated form or conveyance.
The acquired skill known as Trouser Semaphore is swiftly gaining currency as the only way for people of quality to communicate in an age of rapidly escalating background noise levels. Typically, at the race track or at unexpectedly rumbustious parties, attempts to make oneself heard above the general hubbub can prove exasperating and, as often as not, utterly futile. Within the space of a week and with minimal amount of application, it is possible to gain a skill of incalculable worth. Across the floor of a crowded cocktail gathering, you too would be able to convey your innermost thoughts and deepest needs to like minded individuals, using nothing more than flexibility of your

physique and the rough pliability of one's trouser cloth. Surely, there is no sight more moving than a man and three square yards of carefully tailored cavalry twill moving in perfect harmony.
After Years of practice and much patience, the carefully orchestrated movements can transcend mere practicality and approach a level bordering on art. But no matter to which level you yourself wish to pursue the practice of Trouser Semaphore, on a day-to-day practical level, you will find it an invaluable communication aid for use socially, in business and for pleausure"

Here are the fabulous "moves":



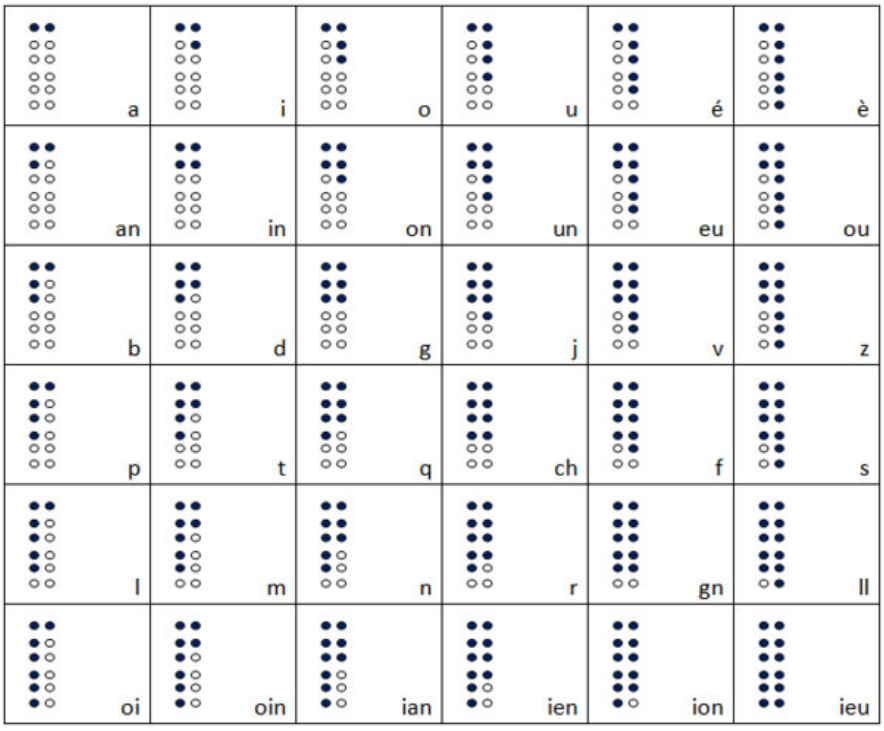
Another note at the end of the page says:

"Having practised for more time, it is perfectly possible to adapt Trouser Semaphore to meet individual needs. Not only does personal style come into the equation but inveterate practitioners often add as many as 20 to 30 unique moves to their repertoire in order to convey information in "short-leg". A great deal of physical exertion and unpleasant bending may be by-passed in this way. Overleaf are some notable examples of this individual approach."

9.3 Night Writing (1815)

Charles Barbier de la Serre, in 1815, invented a system (based on raised dots) for military use, to be able to transmit troop commands in darkness, without any light and without betraying one's own position. Only later, in 1820, the system was used, in suggestion of members of the French Royal Academy of Sciences, to allow blind people to read. Based on 12 dots disposed on 2 columns, it is the father of many other systems like Braille and New York Point. Barbier used a 6x6 Polybius square to represent symbols as a 2-digit code. For example, knowing that the symbol 'M' is placed on the 5th row and 2nd columns of the Polybius square, it will be encoded with 5 dots on the first column and 2 dots on the second column. The Polybius square used by Barbier is the following:

	1	2	3	4	5	6
1	a	i	o	u	é	è
2	an	in	on	un	eu	ou
3	b	d	g	j	v	j
4	p	t	q	ch	f	s
5	l	m	n	r	gn	ll
6	oi	oin	ian	ien	ion	ieu



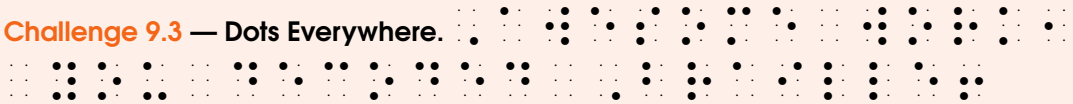
9.4 Braille (1824)

Braille, named after its inventor Louis Braille, is a tactile writing system used to allow visually impaired people to read. Based on military code "night writing" encodes symbols into rectangular blocks having a grid of 3x2 tiny bumps called raised dots. The number and the arrangement of these dots distinguish the different symbols. The arrangement may vary from language to language in order to cover different symbols and purposes. The encoding and the decoding process may also vary based on different optimization.

9.4.1 Grade 1

The simplest encoding system, where single symbols are encoded, is called Grade 1:

a/1		k		u		-	
b/2		l		v		.	
c/3		m		w		!	
d/4		n		x		()	
e/5		o		y		*	
f/6		p		z		/	
g/7		q		,			
h/8		r		;			
i/9		s		’			
j/10		t		:			

Challenge 9.3 — Dots Everywhere. 

Solution 9.3 — Dots Everywhere. Awesome work, you decoded Braille!

9.4.2 Grade 2

Practically speaking, encoding Braille needs lots of physical space. In order to try to minimize spaces, Grade 2 comes with an addition of abbreviations and contractions:

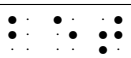
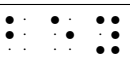
and		ing		ea	
ar		into		bb/be	
ble		of		cc/con	
ch		ou		dd/dis	
ed		ow		ff/to	
en		sh		gg/were	
er		th		his	
for		the		by/was	
gh		wh		com	
in		with		st	

Challenge 9.4 — Less Dots, Longer Words. 

Solution 9.4 — Less Dots, Longer Words. In this way lot of space has been saved.

9.4.3 Grade 3

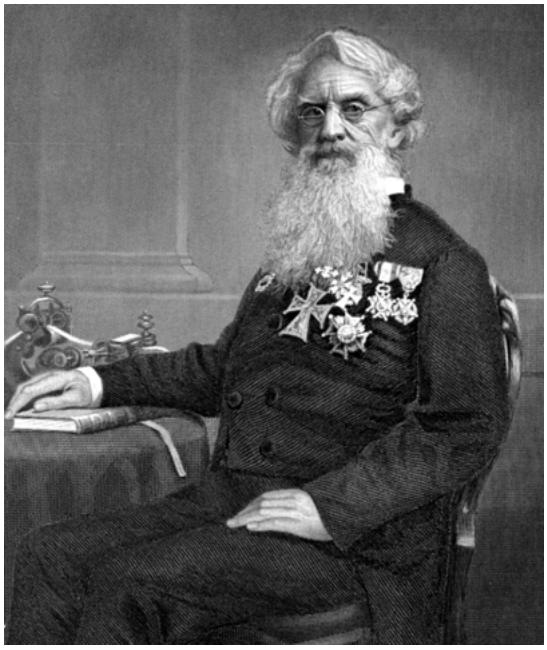
More sophisticate contractions has been developed for every language. There are no official Grade 3 encoding, but can be customized depending on the behaviors. The main idea is to use the first few letters to represent a whole word. What follows is an example of this process:

b for but		m for more		e for every	
ab for about		abv for above		ei for either	
bet for between		bey for beyond		ll for little	
td for today		tm for tomorrow		qk for quick	

Advanced Braille symbols, and combinations of them, are able to represent also the math world, the currencies and are able to distinguish between capital letters, digits and punctuation.

9.5 Morse Code (1835 - 1840)

The Morse code is one of the most common example of encoding, composed primary by dots and lines. This encoding, originally studied by Samuel Morse in 1835, but realized by his collaborator Alfred Vail starting from 1837, was used to encode information that were transferred by electrical telegraph systems.



During the following years the originally encoding schema, that was able to encode and decode the whole English alphabet, has been update to been able to manage digits and symbols coming from different languages. Morse code began to be used extensively till 1900s for early radio communication and nowadays is mostly use for aviation and marine purposes. The following is the International Morse Code (ITU):

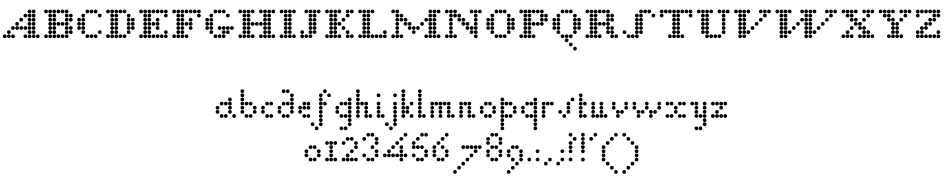
a	.-	n	-.	0	——	)	-.–.-
b	-...	o	—	1	.—-	(	-.–.
c	-.-.	p	.–.	2	..—	:	—...
d	-..	q	–.-	3	...–	,	–..–
e	.	r	.-.	4	....-	=	-...-
f	..-.	s	...	5	.....	!	-.-.–
g	–.	t	-	6	-....	.	.-.-.-
h	....	u	..-	7	–...	-	-....-
i	..	v	...-	8	—..	+	.-.-.
j	.—-	w	.–	9	—-.	"	.-..-.
k	-.-	x	-..-	&	.-...	?	..–..
l	.-..	y	-.–	’	.—-.	/	-..-.
m	–	z	–..	@	.–.-.		

Challenge 9.5 Decode: . -..- .- – .–. .-.. . — ..-. – — .-. ... . -.-. — -.. . ■

Solution 9.5 Decoded: example of morse code

9.6 Raphigraphy (1839)

In 1839 Louis Braille published his new writing system with the purpose to be used both by blind and sighted people. In this new system the shape of the letters are similar to those composing the English alphabet, making raphigraphy readable without any touch and train for sighted people. Despite this for blind people reading raphigraphy is a slow process because symbols don't have a fix size and can reach up to 10 dots in height and various dots in width. In 1841 the first Raphigraph (machine to put raphigraphy symbols into paper) was invented. Raphigraphy so became the first digital font for latin letters in history.



9.7 Moon System of Embossed Reading (1845)

Moon System of Embossed Reading, or simply Moon is a writing system for blind people developed by Dr. William Moon in 1845. Moon lost his sight at age 21 and started developing the writing system in 1943 after he noticed that his pupils had some difficulties in reading existing styles of embossed reading codes. He invented it's own system to be easy, open and clear to the touch. The system grew rapidly thanks to the funds received from the blind philanthropist Sir Charles Lowther. This allowed William Moon to set up a printing press and workshop in order to print books in Moon's system. The machine woked till the 1960 producing tons of books in different languages. It has been studied that the majority of people who go blind in later life are unable to master the small dots of the braille system and so Moon's system it's easy to learn. Nowadays is not famous as Braille's system but it's a very valid alternative even if fluent reading is not possible because lines can be difficult to understand by the blind. One reason of this is that the Moon alphabet is composed by only 14 characters used in various angles to represent the 26 characters of the English alphabet. The detection of angular degrees, slopes or arcs is difficult for a blind person. By the way it is common to use it in order to introduce blind people to the reading by touch and move then to the Braille system. In some cases it is still used as a standard writing for the blind in a few regions of the world, like Latin America.

Another historical disadvantage of Moon's system is that it was slower to reproduce compared to Braille and so literature in Moon is usually less available. The following is the conversion matrix:

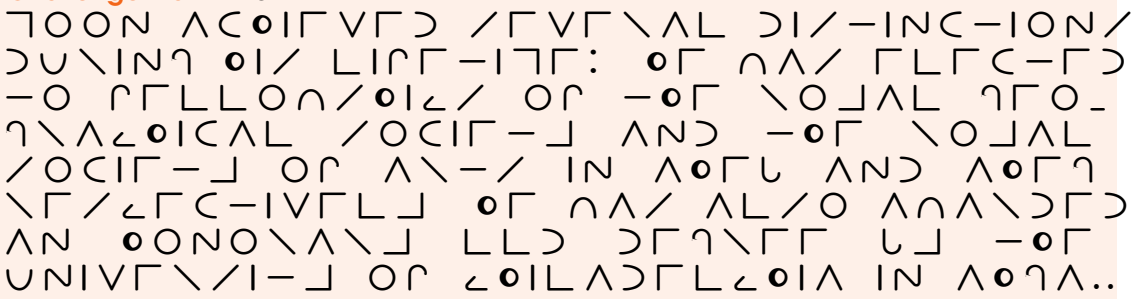
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
															

q	r	s	t	u	v	w	x	y	z	0	1	2	3	4

5	6	7	8	9	?	!	:	*	.	,	(	)	+	-	#	&

=	_	$	%	’	"	/

Challenge 9.6 — Life.

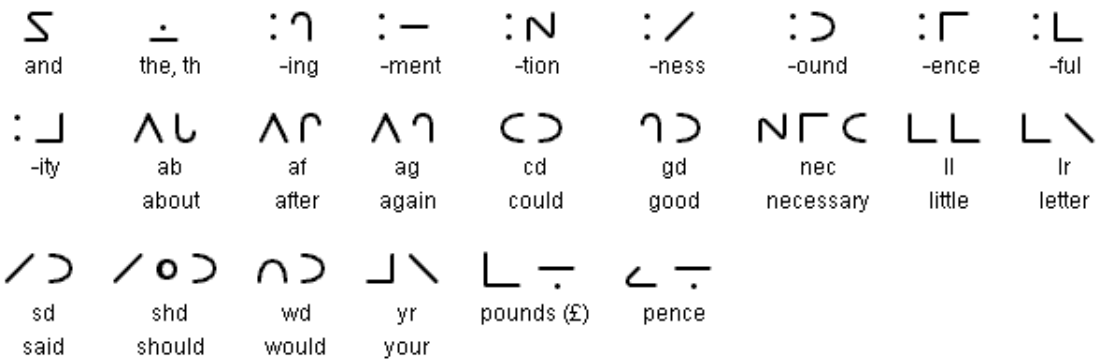


■

Solution 9.6 — Life.
Moon achieved several distinctions during his lifetime: he was elected to fellowships of the Royal Geographical Society and the Royal Society of Arts in 1852 and 1857 respectively; he was also awarded an honorary LLD degree by the University of Philadelphia in 1871.

9.7.1 Grade 2

Like Braille, also Moon’s system have a Grade 2 format containing contractions and shorthands:



9.8 Signal Flags

Signal flags are usually placed vertically each other and read from the top to the bottom. They are a communication system widely used in navy. Signal flags can be encoded in different ways:

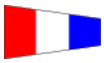
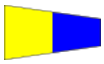
- Every flag can encode a single character, even if pratically this requires a lot of flags;
- Every flag can encode a specific concept if taken alone;
- More flags taken together can encode a more complex message;
- Can be used as an identification system.

9.8.1 International Code of Signals (1855)

The first prototype of the International Code of Signals (ICS) was developed in 1855 by the British Board of Trade and subsequently published in 1857 to help with maritime communications. The 18 proposed flags were used to create about 70000 different messages. The first revision of the code was performed in 1887 and officially modified in 1889. One of the first successful communication using ICS was during the Tsushima battle when Russians exposed the message XGA (I give up) to the Japaneses. After the First World War, in 1927, the code have been modified again in order to cover 7 different languages: English, French, Italian, German, Japanese, Spanish and Norwegian. This new version was adopted later in 1932 for the International Radiotelegraph Conference of Madrid. The new ICS included also numbers and fast medical messages. In 1947 the ICS passed under the competence of the Inter-Governmental Maritime Consultative Organization (IMCO) that revisited the code by adding two new languages, Russian and Turkish, in 1961. Some minor modifications have been done later in 1964 and 1965. The following table summarizes the current ICS flags with their main meaning:

Letter	Flag	Name	Meaning
A		Alpha	"I have a diver down; keep well clear at slow speed."
B		Bravo	"I am taking in or discharging or carrying dangerous goods."
C		Charlie	"Affirmative."
D		Delta	"Keep clear of me; I am maneuvering with difficulty."
E		Echo	"I am altering my course to starboard."
F		Foxtrot	"I am disabled; communicate with me."
G		Golf	"I require a pilot."
H		Hotel	"I have a pilot on board."

Letter	Flag	Name	Meaning
I		India	"I am altering my course to port."
J		Juliet	"I am on fire with dangerous cargo on board: keep well clear of me."
K		Kilo	"I wish to communicate with you."
L		Lima	"You should stop your vessel instantly."
M		Mike	"My vessel is stopped and making no way through the water."
N		November	"Negative."
O		Oscar	"Man overboard!"
P		Papa	"My nets have come fast upon an obstruction."
Q		Quebec	"My vessel is 'healthy' and I request free pratique."
R		Romeo	"No ICS meaning"
S		Sierra	"I am operating astern propulsion."
T		Tango	"Keep clear of me; I am engaged in pair trawling."
U		Uniform	"You are running into danger."
V		Victor	"I require assistance."
W		Whiskey	"I require medical assistance."
X		Xray	"Stop carrying out your intentions and watch for my signals."

Letter	Flag	Name	Meaning
Y		Yankee	"I am dragging my anchor."
Z		ZULU	"I require a tug."
0		Nadazero	"Represents the number zero."
1		Unaone	"Represents the number one."
2		Bissotwo	"Represents the number two."
3		Terrathree	"Represents the number three."
4		Kartefour	"Represents the number four."
5		Pantafive	"Represents the number five."
6		Soxisix	"Represents the number six."
7		Setteseven	"Represents the number seven."
8		Oktoeight	"Represents the number eight."
9		Novenine	"Represents the number nine."
1S		Substitute 1	"Repeat the first flag."
2S		Substitute 2	"Repeat the second flag."
3S		Substitute 3	"Repeat the third flag."

The list of full meanings and flag combinations is now maintained by International Maritime Organization and can be visited online. As an example of combined flags is "AC" that means "I am abandoning my vessel", or "MAA" that means "I request urgent medical advice".

9.8.2 NATO Flags

The North Atlantic Treaty Organization (NATO) uses the ICS as well but usually with different meanings. One of the biggest difference from NATO flags and ICS, are the numerical flags that are completely different. NATO also provides an additional Substitution flag to repeat the fourth flag. As ICS they follows the Phonetic Alphabet that is used widely in military communications. The phonetic alphabet, a system set up in which each letter of the alphabet has a word equivalent to avoid mistaking letters that sound alike. The pronunciation for each letter's phonetic word is contained in the name column. The following table will summarize the changes:

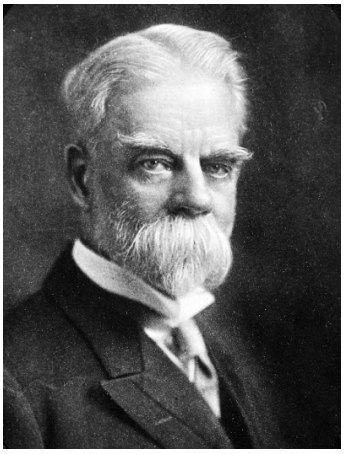
Letter	Flag	Name	Meaning
0		Nadazero	"Represents the number zero."
1		Unaone	"Represents the number one."
2		Bissotwo	"Represents the number two."
3		Terrathree	"Represents the number three."
4		Kartefour	"Represents the number four."
5		Pantafive	"Represents the number five."
6		Soxisix	"Represents the number six."
7		Setteseven	"Represents the number seven."
8		Oktoeight	"Represents the number eight."
9		Novenine	"Represents the number nine."
4S		Substitute 4	"Repeat the fourth flag."

To recognize if a code is encoded through the ICS or the NATO flag system usually NATO ships will fly the code/answer flag above the signal to indicate it should be read using the international meaning. NATO flags extends the ICS with 18 other flags of common usage. Some of them are thought to send orders to specific flotilla, groups, squadrons, divisions and subdivisions. Some of them, like Preparative, Interogative and Negative, are "governing" pennants used toalter the "sense"

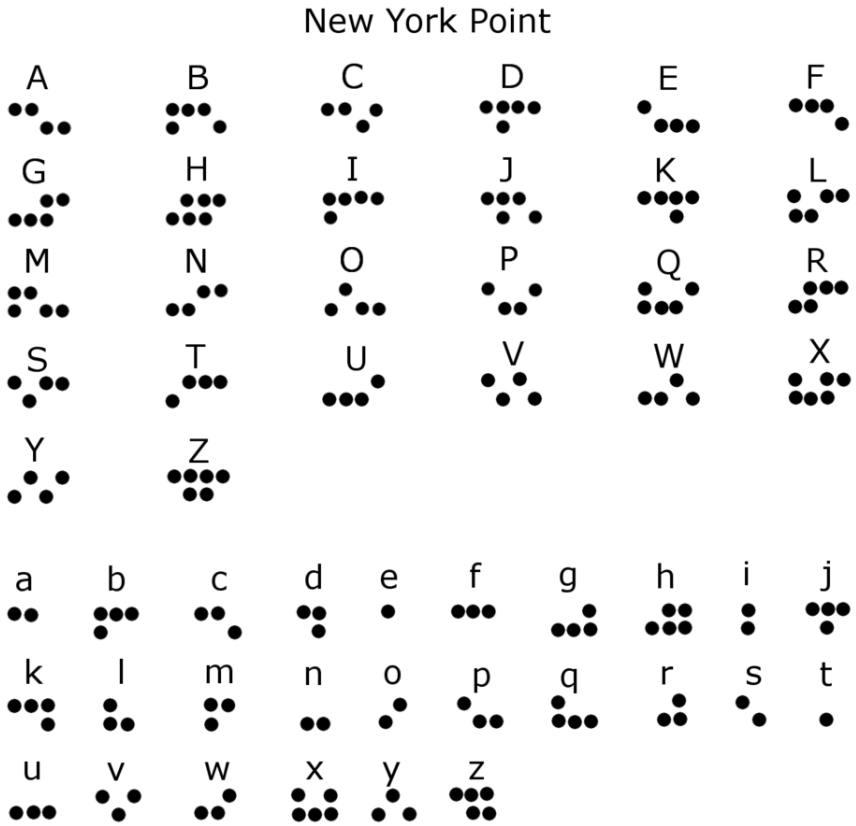
of a signal. There are also maneuvering flags: They used to order to alter speeds, to place troupes in specific formations and to do specific maneuvers, including emergency actions. Some other flags are used for emergency, to designate someone and to specify that we are replying to a question.

Flag	Name	Meaning	Flag	Name	Meaning
	ANS	"Code or Answer."		CORPEN	"Course Pennant."
	DESIG	"Designation."		DIV	"Division."
	EMERG	"Emergency."		FORM	"Formation."
	FLOT	"Group."		NEGAT	"Negation."
	PORT	"Port."		PREP	"Preparatory."
	INT	"Question."		SCREEN	"Screen."
	SPEED	"Speed."		SQUAD	"Squadron."
	STBD	"Starboard."		STATION	"Station."
	SUBDIV	"Subdivision."		TURN	"Turn."

9.9 New York Point (1868)



A braille-like system for blind person have been proposed by William Bell Wait, a teacher in the New York Institute for the Education of the Blind in 1868. Inspired by Braille system he disposed the dots in a grid composed by 2 rows and 4 columns. Each symbol can so be encoded using 1 to 4 pairs of points, each oune containing 1 or 2 dots. The idea behind this is that the most common letters are represented with less dots as possible. A special representation have been reserved for capital letters which are all composed by at least one dot for each possible pair. This is one of the weakness of the system: when capitals, hyphens, or apostrophes were used, they sometimes caused legibility problems. This was one of the main point against the system proposed by Wait and in favor of one of the Braille's system. On the other side, books written in braille are usually voluminous and the proposal to write text on 2 lines against 3 was an advantage for the system. Furthermore, the fact of using less dots for more frequent letters was another good point in favor of New York Point. In the 1910 census of reading codes used in the US, the 57.2% of the people were using the New York Point, the 28.1% the American Braille and only the 4.7% the English Braille. Despite this New York Point and other systems concurred in what is know as The War of the Dots, which lasted in the US and Great Britain for almost 80 years, declaring English Braille as the ultimate winner of the competition thanks to its superior speed of readability and the larger amount of written material available at the time. The typewriter used to write in New York Point is called Kleidograph.



9.10 Knock Code (1941)

The Knock code, or Tap code, has been first proposed by Arthur Koestler in its 1941 classic "Darkness at Noon", even if it was a primitive version of the one used by American soldiers in Vietnam war some years later. The original system was very easy: to encode a letter of the alphabet just tap n times, where n is the position of that letter in the ordered alphabet. To encode A only one tap was needed, for letter S, 19 taps in a row. This particular kind of knock code was used also by Kurt Vonnegut in 1952 in its novel "Player Piano" as a communication system between two prisoners. This could be the inspiration which brings the use a modified version of the code in World War II and later in Vietnam War. This is well documented from the memories of four prisoners of war (Captain Carlyle "Smitty" Harris, Lieutenant Phillip Butler, Lieutenant Robert Peel, and Lieutenant Commander Robert Shumaker) detainees in the Hanoi Hilton prison that used the Tap code in order to communicate each other about interrogators questions and to keep up the morale. The modified code uses a Polybius square to reduce the number of taps required to encode letters at the end of the alphabets. Now each letter is communicated by tapping 2 numbers with at most 10 taps. In comparison, Morse code is harder to communicate because it requires the ability to create "long" taps and with a precise rhythm. The square used was the following:

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

Challenge 9.7 — Prisoner Song. .. .. ■

Solution 9.7 — Prisoner Song.
If I'm underneath the table
Then pour me another drink
Oh, I don't want to remember
I don't even want to think

9.11 DTMF (1963)

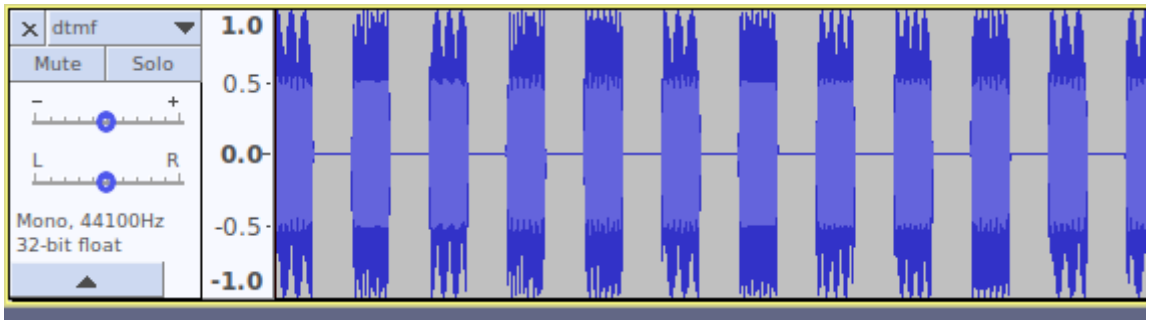
Since the invention of the telegraph in 1837 by Cooke, and next of the telephone in 1849 by Meucci, the main problem was to define an encoding system able to ensure communication between 2 communication devices. One of the first attempt, used since the second half of the 20th century was based on Pulse Dialing systems where a local loop circuit is interrupted according to a defined coding system for each signal transmitted. This system was invented in 1892 by Almon Brown Strowger and was able to send only digits. An example of this system can be found in rotary phones.

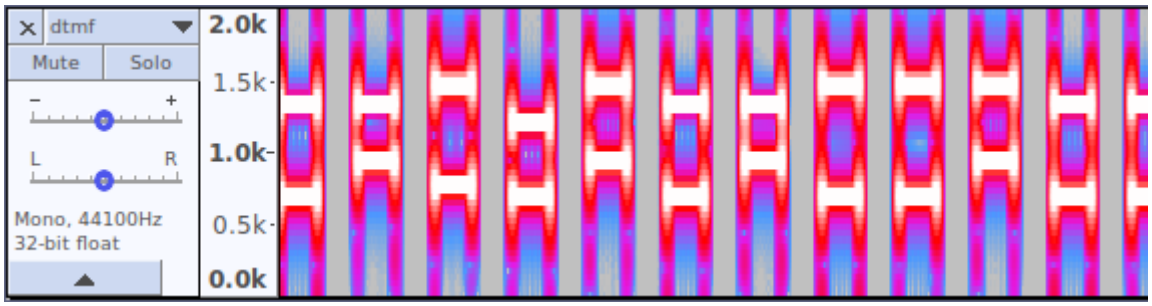
In 1963 engineers at Bell developed a new kind of technology "called Dual-Tone multi-frequency signaling (DTMF)" whose become the new communication standard. The new telephones had no rotors, but a push able 16-keys keypad, friendly called Touch-Tone. This new system allowed to make long-rage calls and for first time the communication channel was not restricted to copper cables but extended to microwave bridges and satellites.

The keypad hosts digits from 0 to 9, letters from A to D and two special characters: * and #. Each button produces a different DTMF signal, composed by a low frequency and an high frequency component. For example the key A produces a 697 Hz low tone and a 1633 Hz high tone. The next table will summarize the tone frequencies of each button:

	1209 Hz	1336 Hz	1477 Hz	1633 Hz
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*	0	#	D

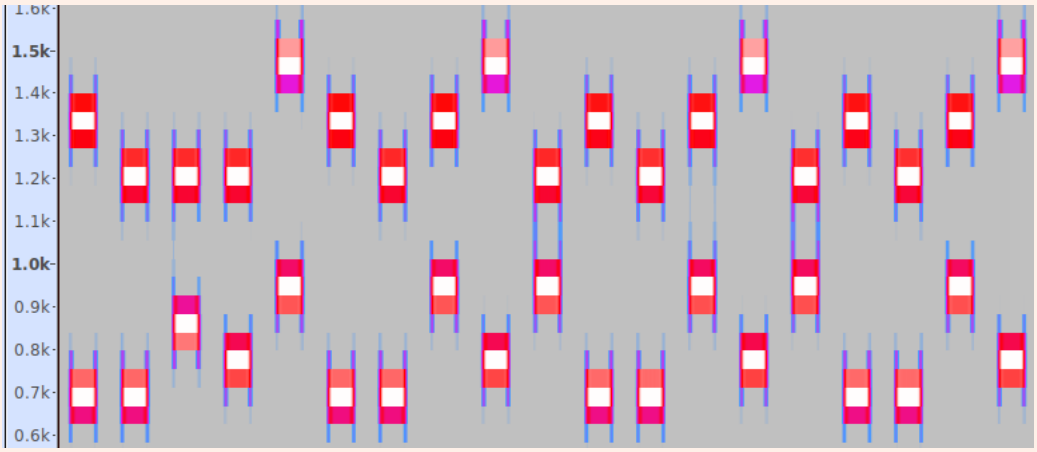
The decoding system was provided by filter banks first, and by digital signal processing then, usually using the Goertzel algorithm. Frequencies of tones of a DTMF communication can be visually seen by inspecting the spectogram of the signal itself:





Decoding the signal graphically is quite simple, just use the encoding table. For example the signal in the picture can be decoded as "2064#2033#2...". DTMF signals are used in steganography to cover binary messages and hex messages even if with some limitations (E and F are not reproducible).

Challenge 9.8 Can you decode this DTMF signal?



Solution 9.8 Decoded: 2174#2106*2106*2106

9.12 Alexander Fakoó Alphabets

Alexander Fakoo was the inventor of different alternative braille systems that blind and sighted people could use together. They were created so that communication between the blind, visually impaired and sighted people is established or enhanced. Applications of these systems can already be found today on signposts, plans, keys and many other objects.

9.12.1 Fakoo (2007)

The Fakoo script is suitable for sending messages blind to sighted, which Braille can not read, information sighted for the blind and visually impaired or as an alternative for the blind, who have not learned Braille or Braille is too complicated. Although the length of the font is double that of Braille, which make it easy readable to the sighted and the blind. Especially for late-blind people, this font is easier to learn because the Latin letters form the basis and no characters are double-occupied.

The idea of making the writing for the sighted, palpable with the help of dots, had, as we already saw, already been anticipated. For example with Raphigraphy. Fakoo extends this idea not only to letters but also to numbers and punctuation marks. Even if the initial proposal was to use a grid of 3x4 dots (Fakoo12), Alexander notices that it was unpractical and moved to a 3x3 grid (Fakoo9). Due to the consistent compliance of a maximum of 3 dots in height and width, Fakoo can be used for the alternative dot-writing with all Braille utensils and techniques.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p

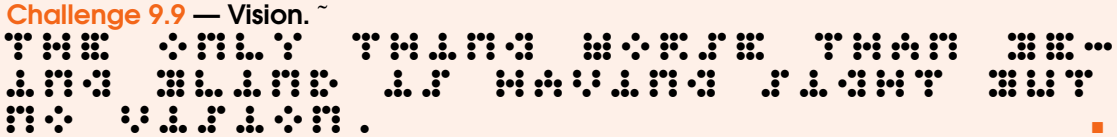
q	r	s	t	u	v	w	x	y	z	0	1	2	3	4	5

6	7	8	9	.	,	:	;	+	-	*	#	!	?	@	<

=	>	(	)	[	]	{	}	~	_	/	|	%	&	e	$

\	^	’	"

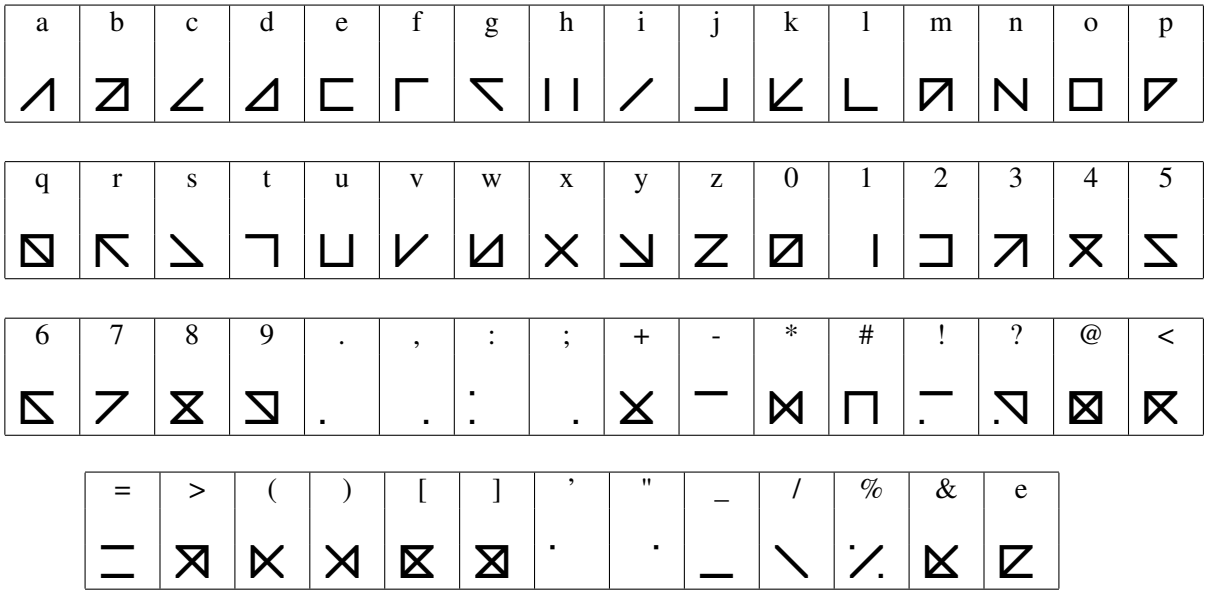
Challenge 9.9 — Vision. ˜



Solution 9.9 — Vision. ˜
The only thing worse than being blind is having sight but no vision.

9.12.2 Quadoo (2008)

The Quadoo Alphabet was developed in order to consent by the blinds to take handwritten notes. In this alphabet a single square, with its diagonals, is used in order to represent a single symbol. There are no circles or arcs in the letters and numbers that the blind can only write inaccurately, only the punctuation marks contain dots.



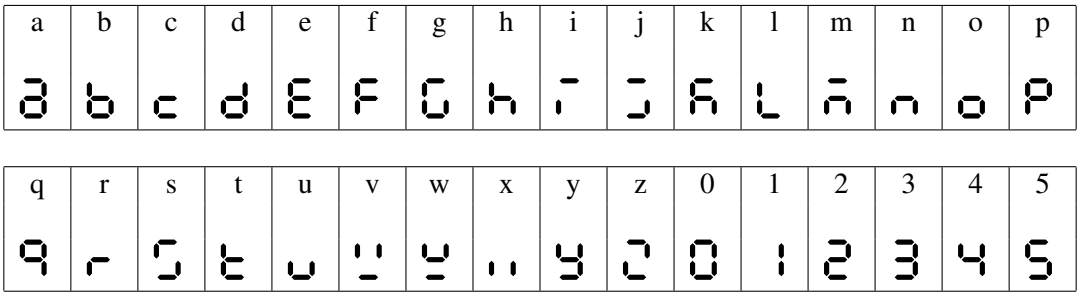
Challenge 9.10 — Handicap. ˜

■

Solution 9.10 — Handicap. ˜
Blindness is an unfortunate handicap but true vision does not require the eyes.

9.12.3 Siekoo (2012)

The last created alphabet is called Siekoo. It uses the well-known 7-segment display to realize a confusion-free alpha-numeric text representation.



6	7	8	9	.	,	:	;	+	-	*	#	!	?	@	<
6	7	8	9	.	,	:	;	+	-	*	#	!	?	@	<

=	>	(	)	’	"	_	/	%
=	>	(	)	’	"	_	/	%

Challenge 9.11 — Friendship. ˜

Love comes from blindness, friendship from knowledge.

Solution 9.11 — Friendship. ˜

Love comes from blindness, friendship from knowledge.

10. Digital Encodings

10.1 ASCII (1960 - 1967)

The American Standard Code for Information Interchange (ASCII), is a binary character encoding standard for electronic communication based on 7-bits, designed in 1960s for teleprinters, telegraphy and some computing. This means that each symbol is unequivocally identified by a sequence of 1s and 0s of length 7. ASCII codes, nowadays, represent text in computers, telecommunications equipment, and other devices. Being a binary encoding system it can also be translated in other numeric bases and usually, for challenge purposes, can be found written in base 2, base 8, base 10 and base 16. The following conversion table will summarize all this cases:

Bin	Oct	Dec	Hex	Symbol	Bin	Oct	Dec	Hex	Symbol
0000000	000	0	00	NUL	0100000	040	32	20	Space
0000001	001	1	01	SOH	0100001	041	33	21	!
0000010	002	2	02	STX	0100010	042	34	22	"
0000011	003	3	03	ETX	0100011	043	35	23	#
0000100	004	4	04	EOT	0100100	044	36	24	$
0000101	005	5	05	ENQ	0100101	045	37	25	%
0000110	006	6	06	ACK	0100110	046	38	26	&
0000111	007	7	07	BEL	0100111	047	39	27	’
0001000	010	8	08	BS	0101000	050	40	28	(
0001001	011	9	09	TAB	0101001	051	41	29	)
0001010	012	10	0A	LF	0101010	052	42	2A	*
0001011	013	11	0B	VT	0101011	053	43	2B	+
0001100	014	12	0C	FF	0101100	054	44	2C	,
0001101	015	13	0D	CR	0101101	055	45	2D	-
0001110	016	14	0E	SO	0101110	056	46	2E	.
0001111	017	15	0F	SI	0101111	057	47	2F	/
0010000	020	16	10	DLE	0110000	060	48	30	0
0010001	021	17	11	DC1	0110001	061	49	31	1
0010010	022	18	12	DC2	0110010	062	50	32	2
0010011	023	19	13	DC3	0110011	063	51	33	3
0010100	024	20	14	DC4	0110100	064	52	34	4
0010101	025	21	15	NAK	0110101	065	53	35	5
0010110	026	22	16	SYN	0110110	066	54	36	6
0010111	027	23	17	ETB	0110111	067	55	37	7
0011000	030	24	18	CAN	0111000	070	56	38	8
0011001	031	25	19	EM	0111001	071	57	39	9
0011010	032	26	1A	SUB	0111010	072	58	3A	:
0011011	033	27	1B	ESC	0111011	073	59	3B	;
0011100	034	28	1C	FS	0111100	074	60	3C	<
0011101	035	29	1D	GS	0111101	075	61	3D	=
0011110	036	30	1E	RS	0111110	076	62	3E	>
0011111	037	31	1F	US	0111111	077	63	3F	?

Bin	Oct	Dec	Hex	Symbol	Bin	Oct	Dec	Hex	Symbol
1000000	080	64	40	@	1100000	120	96	60	‘
1000001	081	65	41	A	1100001	121	97	61	a
1000010	082	66	42	B	1100010	122	98	62	b
1000011	083	67	43	C	1100011	123	99	63	c
1000100	084	68	44	D	1100100	124	100	64	d
1000101	085	69	45	E	1100101	125	101	65	e
1000110	086	70	46	F	1100110	126	102	66	f
1000111	087	71	47	G	1100111	127	103	67	g
1001000	090	72	48	H	1101000	130	104	68	h
1001001	091	73	49	I	1101001	131	105	69	i
1001010	092	74	4A	J	1101010	132	106	6A	j
1001011	093	75	4B	K	1101011	133	107	6B	k
1001100	094	76	4C	L	1101100	134	108	6C	l
1001101	095	77	4D	M	1101101	135	109	6D	m
1001110	096	78	4E	N	1101110	136	110	6E	n
1001111	097	79	4F	O	1101111	137	111	6F	o
1010000	100	80	50	P	1110000	140	112	70	p
1010001	101	81	51	Q	1110001	141	113	71	q
1010010	102	82	52	R	1110010	142	114	72	r
1010011	103	83	53	S	1110011	143	115	73	s
1010100	104	84	54	T	1110100	144	116	74	t
1010101	105	85	55	U	1110101	145	117	75	u
1010110	106	86	56	V	1110110	146	118	76	v
1010111	107	87	57	W	1110111	147	119	77	w
1011000	110	88	58	X	1111000	150	120	78	x
1011001	111	89	59	Y	1111001	151	121	79	y
1011010	112	90	5A	Z	1111010	152	122	7A	z
1011011	113	91	5B	[	1111011	153	123	7B	{
1011100	114	92	5C	\	1111100	154	124	7C	|
1011101	115	93	5D	]	1111101	155	125	7D	}
1011110	116	94	5E	^	1111110	156	126	7E	~
1011111	117	95	5F	_	1111111	157	127	3F	DEL

As can be noticed, ASCII reserves the first 32 codes for control characters, that are not printable but used for devices control (like printers, keyboards and scanners) or to provide meta-data about data streams. The remaining symbols are all printable characters representing uppercase and lower case letters, digits and most common symbols.

Challenge 10.1 Decode: something in bin ■

Solution 10.1 Decoded: example of morse code

Challenge 10.2 Decode: something in hex ■

Solution 10.2 Decoded: example of morse code

10.1.1 Extended ASCII (1980s - 2000s)

ASCII encoding is limited to 128 symbols. In order to represent symbols belonging to different languages, more modern ones like ™, ©, and scientific symbols, the 7-bit encoding has been extended to 8-bit so that at most 256 symbols could be represented. The implementation of the extended symbols is free and can vary between language and purpose. For example, the ISO 8859 maintain a standard with 16 different tables of extended symbols that cover lots of different characters. The mostly used table is ISO 8859-15, also called "Latin-9" that covers Western European languages such as English, German, Italian, Spanish, Irish, French, Danish, Finnish, Dutch and others. It also covers some other non-European languages like Indonesian, Afrikaans and Swahili:

Bin	Oct	Dec	Hex	Symbol	Bin	Oct	Dec	Hex	Symbol
10100000	240	160	A0	NBSP	11000000	280	192	C0	À
10100001	241	161	A1	ą	11000001	281	193	C1	Á
10100010	242	162	A2	ć	11000010	282	194	C2	Â
10100011	243	163	A3	č	11000011	283	195	C3	Ã
10100100	244	164	A4		11000100	284	196	C4	Ä
10100101	245	165	A5	ě	11000101	285	197	C5	Å
10100110	246	166	A6		11000110	286	198	C6	Æ
10100111	247	167	A7	ğ	11000111	287	199	C7	Ç
10101000	250	168	A8		11001000	290	200	C8	È
10101001	251	169	A9	©	11001001	291	201	C9	É
10101010	252	170	AA	ł	11001010	292	202	CA	Ê
10101011	253	171	AB	ń	11001011	293	203	CB	Ë
10101100	254	172	AC	ň	11001100	294	204	CC	Ì
10101101	255	173	AD	SHY	11001101	295	205	CD	Í
10101110	256	174	AE	ő	11001110	296	206	CE	Î
10101111	257	175	AF	ŕ	11001111	297	207	CF	Ï
10110000	260	176	B0	ř	11010000	300	208	D0	Đ
10010001	261	177	B1	ś	11010001	301	209	D1	Ñ
10110010	262	178	B2	š	11010010	302	210	D2	Ò
10110011	263	179	B3	ş	11010011	303	211	D3	Ó
10110100	264	180	B4		11010100	304	212	D4	Ô
10110101	265	181	B5	ţ	11010101	305	213	D5	Õ
10110110	266	182	B6	ű	11010110	306	214	D6	Ö
10110111	267	183	B7	ů	11010111	307	215	D7	Œ
10111000	270	184	B8		11011000	310	216	D8	Ø
10111001	271	185	B9	ź	11011001	311	217	D9	Ù
10111010	272	186	BA	ž	11011010	312	218	DA	Ú
10111011	273	187	BB	ż	11011011	313	219	DB	Û
10111100	274	188	BC		11011100	314	220	DC	Ü
10111101	275	189	BD		11011101	315	221	DD	Ý
10111110	276	190	BE		11011110	316	222	DE	Þ
10111111	277	191	BF	£	11011111	317	223	DF	SS

Bin	Oct	Dec	Hex	Symbol	Bin	Oct	Dec	Hex	Symbol
11100000	320	224	E0	à	11110000	340	240	F0	ð
11100001	321	225	E1	á	11110001	341	241	F1	ñ
11100010	322	226	E2	â	11110010	342	242	F2	ò
11100011	323	227	E3	ã	11110011	343	243	F3	ó
11100100	324	228	E4	ä	11110100	344	244	F4	ô
11100101	325	229	E5	å	11110101	345	245	F5	õ
11100110	326	230	E6	æ	11110110	346	246	F6	ö
11100111	327	231	E7	ç	11110111	347	247	F7	œ
11101000	330	232	E8	è	11111000	350	248	F8	ø
11101001	331	233	E9	é	11111001	351	249	F9	ù
11101010	332	234	EA	ê	11111010	352	250	FA	ú
11101011	333	235	EB	ë	11111011	353	251	FB	û
11101100	334	236	EC	ì	11111100	354	252	FC	ü
11101101	335	237	ED	í	11111101	355	253	FD	ý
11101110	336	238	EE	î	11111110	356	254	FE	þ
11101111	337	239	EF	ï	11111111	357	255	FF	ß

Challenge 10.3 Decode: something in oct ■

Solution 10.3 Decoded: example of morse code

Challenge 10.4 Decode: something in hex ■

Solution 10.4 Decoded: example of morse code

10.2 BaseN Encodings (1993)

10.2.1 Base64

Base64 is an encoding schema that is used to represent binary data contained in an ASCII string in another format by translating it into a string composed by a sequence of symbols coming from an alphabet of cardinality 64. Each symbol of the generated base64 string represent 6 bits of the original data, so that every 3 original symbols are represented by 4 base64 symbols. This means that length of the base64 output is always longer with respect to the length of the original message. Due to its ductility and capacity to transform binary data, it is mainly used to carry information of different sources (images, videos, media, audio...) through the World Wide Web.

The encoding procedure is the following (if starting from binary, skip first 2 points):

- Convert each input symbol to its ASCII decimal representation;
- Convert each decimal into its 8-bit representation (octets);
- Group bits into buckets of size 24
- Threat every 6 bits of each group as sextets and convert back to decimal representation
- Following the encoding table, replace each number with the corresponding symbol
- If the last bucket contains less than 24 bits add one or two padding symbols (=) to fill the bucket

The encoding table is the following:

Dec	Char	Dec	Char	Dec	Char	Dec	Char
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

Let's do an example: imagine we want to encode the word "strange" in base64. We have that:

Word	s	t	r	a	n	g	e
ASCII	115	116	114	97	110	103	101
Binary	01110011	01110100	01110010	01100001	01101110	01100111	01100101

Now the goal is to split into buckets of 24 bits:

011100110111010001110010	011000010110111001100111	01100101xxxxxxxxxxxxxxxx

It can be noticed that the last bucket contains only 8 bits and so will be filled in order to reach the 24 elements. Each bucket is now transformed according to the previous algorithm and to the encoding table. For each bucket we have:

Sextets	011100	110111	010001	110010
ASCII	28	55	17	50
Base64	c	3	R	y

Sextets	011000	010110	111001	100111
ASCII	24	22	57	39
Base64	Y	W	5	n

Sextets	011001	01xxxx	xxxxxx	xxxxxx
normalized	011001	010000	xxxxxx	xxxxxx
ASCII	25	16	?	?
Base64	Z	Q	=	=

All toghether we obtain that base64("strange") = "c3RyYW5nZQ==". The decoding process is extremely easy and will not be showed, just follw steps backward. There are some variants for the symbols at position 62 and 63 of the encoding table for different implementations that usually replace the '+' and the '/' symbols to others like '-', '_', '~', '.' and ':'. Some implementations make the padding symbol optional.

Challenge 10.5 — superego. What have Freud to say?
FrQXQgdGhlIHRpbWUgYXQgd2hpY2ggdGhlIE9lZGlwdXMgY2
9tcGxleCBnaXZlcyBwbGFjZSB0byB0aGUgc3VwZXItZWdvIH
RoZXkgYXJlIHNvbWV0aGluZyBxdWl0ZSBtYWduaWZpY2VudDs= ■

Solution 10.5 — superego. Such egocentric patients
At the time at which the Oedipus complex gives place to the super-ego they are something quite magnificent

10.2.2 Base32

Base64 has two huge limitation. The biggest one is that it cannot be used in case-insensitive systems due to the fact that the encryption table contains both a-z and A-Z symbols. Base64 cannot also be used to represent file names because the symbol '/' is included in the encryption table and for unix systems represents the path separator. A slightly simple variant of Base64 is Base32.
It is a very similar encoding scheme that encodes buckets of 5 bits instead of 6 bits producing an output, on average, 20% longer with respect to base64. The encoding algorithm is the following:

- Convert each input symbol to its ASCII decimal representation;
- Convert each decimal into its 8-bit representation (octets);
- Group bits into buckets of size 40
- Threat every 5 bits of each group as quintets and convert back to decimal representation
- Following the encoding table, replace each number with the corresponding symbol
- If the last bucket contains less than 40 bits add one or two padding symbols (=) to fill the bucket

As for base64 the most used encoding table for base32 is the one coming from RFC4648:

Dec	Char	Dec	Char	Dec	Char	Dec	Char
0	A	8	I	16	Q	24	Y
1	B	9	J	17	R	25	Z
2	C	10	K	18	S	26	2
3	D	11	L	19	T	27	3
4	E	12	M	20	U	28	4
5	F	13	N	21	V	29	5
6	G	14	O	22	W	30	6
7	H	15	P	23	X	31	7

RFC4648 suggests also another encryption table called "base32hex" with the following motivation:

"One property with this alphabet, which the base64 and base32 alphabets lack, is that encoded data maintains its sort order when the encoded data is compared bit-wise."

The new table is the following:

Dec	Char	Dec	Char	Dec	Char	Dec	Char
0	0	8	8	16	G	24	O
1	1	9	9	17	H	25	P
2	2	10	A	18	I	26	Q
3	3	11	B	19	J	27	R
4	4	12	C	20	K	28	S
5	5	13	D	21	L	29	T
6	6	14	E	22	M	30	U
7	7	15	F	23	N	31	V

Let's do the same example did before but this time with base32 encoding:

Word	s	t	r	a	n	g	e
ASCII	115	116	114	97	110	103	101
Binary	01110011	01110100	01110010	01100001	01101110	01100111	01100101

Now the goal is to split into buckets of 40 bits:

0111001101110100011100100110000101101110	01100111011001010000000000000000000000000

It can be noticed that the last bucket contains only 8 bits and so will be filled in order to reach the 24 elements. Each bucket is now transformed according to the previous algorithm and to the encoding table. For each bucket we have:

Quintects	01110	01101	11010	00111	00100	11000	01011	01110
ASCII	14	13	26	7	4	24	11	14
Base32	O	N	2	H	E	Y	L	O

Quintects	01100	11101	10010	10000	00000	00000	00000	00000
Normalized	01100	11101	10010	10000	00000	00000	00000	00000
ASCII	12	29	18	16	0	0	0	0
Base32	M	5	S	Q	=	=	=	=

All toghether we obtain that base32("strange") = "ON2HEYLOM5SQ====". It is difficult to distinguish between base32 and base64 in some cases, but can be noticed that:

- Base32 does not contain lower case characters;
- Base32 contains only letters and numbers and not other symbols (excluding the padding one);
- Base64 has at most 2 padding symbols. Base32 can have up to 6 padding symbols.

Challenge 10.6 — ego. Tell me Sigmund!
KRUGKIDFM5XSA2LTEBXG65BAONUGC4TQNR4SA43FOBQXEYLU
MVSCAZTSN5WSA5DIMUQGSZB3EBUXI4ZANRXXOZLSEBYG64TU
NFXW4IDNMVZGOZLTEBUW45DPEBUXILROFYXCAQTVOQQHI2DF
EBZGK4DSMVZXGZLEEBWWK4THMVZSA2LOORXSA5DIMUQGSZBA
MFZSA53FNRWCYIDBNZSCA2LTEBWWK4TFNR4SAYJAOBQXE5BA
N5TCA2LUFYQFI2DFEBZGK4DSMVZXGZLEEBUXGIDPNZWHSIDD
OV2CA33GMYQHG2DBOJYGY6JAMZZG63JAORUGKIDFM5XSAYTZ
EB2GQZJAOJSXG2LTORQW4Y3FOMQG6ZRAOJSXA4TFONZWS33O
HMQGS5BAMNQW4IDDN5WW25LONFRWC5DFEB3WS5DIEB2GQZJA
MVTW6IDUNBZG65LHNAQHI2DFEBUWI===

Solution 10.6 — ego. So brilliant!
The ego is not sharply separated from the id; its lower portion merges into it.... But the repressed merges into the id as well, and is merely a part of it. The repressed is only cut off sharply from the ego by the resistances of repression; it can communicate with the ego through the id

10.3 Omnicode

OmniCode is a compact symbolic system for encoding personal attributes, preferences, traits, and metadata in a machine- and human-readable form. Originally created by the UserFriendly community, OmniCode provides an extensible notation where each attribute uses a short symbol followed by an attribute value and optional modifiers. This chapter preserves the full OmniCode 0.1.7 specification for historical and archival purposes.

Each OmniCode entry consists of:

```
[symbol][attribute].[modifier].[modifier]...
```

Multiple Attributes

A single code symbol with multiple attributes uses "&":

```
ey{R}00FF00 & {L}888888
```

Reserved Characters

- `&` joins multiple attributes.
- `;` secondary joining inside a single attribute.

Formatting Rules

- Codes may appear in *any order*.
- Codes end with a space.
- Spaces in modifiers are replaced by underscores (_).
- Modifiers become more specific to the right.
- Case sensitivity is required.
- Line breaks must begin with `^` on the continuation line.

10.3.1 Universal Attribute Modifiers

These may be applied to any OmniCode symbol:

`!`	negation (does not apply)
`+`	strongly applies / higher than average
`-`	weakly applies / lower than average
`=`	average
`*`	all modifiers apply
`?`	unknown or unsure
`(n)`	level or achievement, 0–9
`^(n)`	aspiring to level *n*
`{R}` / `{L}`	right / left
`;`	secondary joining

10.3.2 Core Attributes

Gender (sx)

```
sxy      male
sxx      female
sx*      hermaphrodite
sx!      asexual
sx~      transgender
```

Height (cm)

```
cm[att].[mod]
cm180    height in centimeters
```

Weight (kg)

```
kg[att].[mod]
kg75     weight in kilograms
```

Skin Color

Hexadecimal RGB values:

```
sk[att].[mod]
Examples:
skFFFFFF        Full White
sk000000        Full Black
skEAD5C9        Lighter skin tone
sk9E684F        Darker skin tone
```

Hair Color

Hexadecimal RGB values:

```
ha[att]
```

Eye Color

Hexadecimal RGB values:

```
ey[att]
```

Eyesight (es)

```
es+ Better than 20/20
es= 20/20 vision
eso Farsighted
esO Nearsighted
es# Astigmatic
es~ Colourblind
es! Blind
```

Sexual Preference (sp)

```
sp= heterosexual
sp<homosexual
sp* bisexual
sp! non-sexual
sp!.human non-humans
spO             Use to specify something not listed (ie. spO.cars)
```

Age (Ag)

```
Ag1987     year of birth
```

Ancestry (an)

```
anA African
anC Celt
anE European
anO Asian
anP Polynesian & other Islander
anI Native (North and South) American
anU Other Aboriginal
an[attr].[mod]&[attr].[mod] More than one apply (defined by
modifier)
an? Unsure
an! nonhuman
```

Handedness (hd)

```
hdd    right-handed
hds    left-handed
hda    ambidextrous
hdn    ambi-sinistral
hd!    no hands
```

Location (Lo)

Latitude and longitude:

```
Lo49.257498,-123.181458
```

Timezone (GM)

```
There are 5 sections to this code:
1. GM
2. + (plus) or - (minus) from GMT. People -in- GMT use GM0
3. Time zone differential from Greenwich Mean Time (GMT)
4. D or no D. This determines whether your locale uses Daylight Savings
```

```
Time or
   not
5. Hemisphere: S (southern) or N (Northern) to determine your wintertime as applicable fo
   Example:
     GM+8.5DS (8.5plus from GMT, Daylight savings in a Southern
hemisphere winter.
```

Astrological Signs (Zo)

Western and Chinese zodiac:

```
ZoQ Aquarius
ZoP Pisces
ZoA Aries
ZoT Taurus
ZoG Gemini
ZoC Cancer
ZoL Leo
ZoV Virgo
ZoB Libra
ZoS Scorpio
Zos Sagittarius
Zoc Capricorn
ZoR Rat
ZoO Ox
Zot Tiger
Zor Rabbit
ZoD Dragon
ZoK Snake
ZoH Horse
Zob Sheep/Goat
ZoM Monkey
Zoo Rooster
Zod Dog
Zop Pig
```

10.3.3 Learned Attributes

things that can change over time

Religion (rl)

```
rlC     Christian
rlI     Islam
rlH     Hindu
rlB     Buddhist
rlS     Sikh
rlY     Yoruba
rlJ     Jewish
rlj     Jainist
rls     Shinto
rlN     Neo-pagan
rlL     Scientologist
```

```
rlR      Rastafarian
rlZ      Zoroastrianist
rlD      Discordian
rlG      Gravitism
rlF     Jedi
rlM     Flying Spaghetti Monsterism
rlh     Humanist
rlb     Church of the SubGenius
rl?     Agnostic
rl!     Atheist
rl<    Self
rlO.[mod] Other
```

Language (LA)

LA - Language - (Based on standard: ISO 639) For multiple languages and fluencies, mark fluencies with a '9' (1 means you know a few menu items and swear words and could probably find a restaurant and a bathroom, 9 being 'fluent').

```
LAAA Afar
LAAB Abkhazian
LAAF Afrikaans
LAAM Amharic
LAAR Arabic
LAAS Assamese
LAAY Aymara
LAAZ Azerbaijani
LABA Bashkir
LABE Byelorussian
LABG Bulgarian
LABH Bihari
LABI Bislama
LABN Bengali
LABO Tibetan
LABR Breton
LACA Catalan
LACO Corsican
LACS Czech
LACY Welsh
LADA Danish
LADE German
LADZ Bhutani
LAEL Greek
LAEN English
LAEO Esperanto
LAES Spanish
LAET Estonian
LAEU Basque
LAFA Persian
LAFI Finnish
LAFJ Fiji
```

```
LAFO Faeroese
LAFR French
LAFY Frisian
LAGA Irish
LAGD Gaelic
LAGL Galician
LAGN Guarani
LAGU Gujarati
LAHA Hausa
LAHI Hindi
LAHR Croatian
LAHU Hungarian
LAHY Armenian
LAIA Interlingua
LAIE Interlingue
LAIK Inupiak
LAIN Indonesian
LAIS Icelandic
LAIT Italian
LAIW Hebrew
LAJA Japanese
LAJI Yiddish
LAJW Javanese
LAKA Georgian
LAKI Klingon
LAKK Kazakh
LAKL Greenlandic
LAKM Cambodian
LAKN Kannada
LAKO Korean
LAKS Kashmiri
LAKU Kurdish
LAKY Kirghiz
LALA Latin
LALN Lingala
LALO Laothian
LALT Lithuanian
LALV Latvian
LAMG Malagasy
LAMI Maori
LAMK Macedonian
LAML Malayalam
LAMN Mongolian
LAMO Moldavian
LAMR Marathi
LAMS Malay
LAMT Maltese
LAMY Burmese
LANA Nauru
```

```
LANE Nepali
LANL Dutch
LANO Norwegian
LAOC Occitan
LAOM Oromo
LAOR Oriya
LAPA Punjabi
LAPL Polish
LAPS Pashto
LAPT Portuguese
LAQU Quechua
LARM Rhaeto-Romance
LARN Kirundi
LARO Romanian
LARU Russian
LARW Kinyarwanda
LASA Sanskrit
LASD Sindhi
LASG Sangro
LASH Serbo-Croatian
LASI Singhalese
LASK Slovak
LASL Slovenian
LASM Samoan
LASN Shona
LASO Somali
LASQ Albanian
LASR Serbian
LASS Siswati
LAST Sesotho
LASU Sudanese
LASV Swedish
LASW Swahili
LATA Tamil
LATE Tegulu
LATG Tajik
LATH Thai
LATI Tigrinya
LATK Turkmen
LATL Tagalog
LATN Setswana
LATO Tonga
LATR Turkish
LATS Tsonga
LATT Tatar
LATW Twi
LAUK Ukrainian
LAUR Urdu
LAUZ Uzbek
```

```
LAVI Vietnamese
LAVO Volapuk
LAWO Wolof
LAXH Xhosa
LAYO Yoruba
LAZH Chinese
LAZU Zulu
LAAL Artifical language not listed (ie Loglang)
```

Career (Cr)

Cr - Career - Replace bracketed 'n' with a number from 0-9 showing level of accomplishment. For those who wish to specify further, use a trailing period and another modifier. For example: CrE(8).Architect

```
Cra(n)     Artist
Crb(n)     Finance and banking
Crm(n)     Construction and manufacturing
Cre(n)     Education
CrE(n)     Engineering
Crs(n)     Science
Crc(n)     Computing
Crh(n)     Health
Crp(n)     Philosophy
CrM(n)     Military
Crl(n)     Law
Crg(n)     Government
Crd(n)     Media
Crn(n)     Nature
Crr(n)     Recreation and leisure
CrS(n)     Sales
Crt(n)     Transportation
CrH(n)     Home-worker
CrL(n)     Student
Cr!(n)     Unemployed
CrR(n)     Retired
Cr$(n)     Independently Wealthy
CrP(n)     Part-time or temporary "odd jobs"
CrO(n)     Other
Cr?(n)     Money arrives from somewhere
```

Education (Ed)

Ed - Education - Replace bracketed 'n' with a number from 0-9 showing level of accomplishment. For those who wish to specify further, use a trailing period and another modifier. For example: EdE(8).Architect

```
Eda(n)     Artist
Edb(n)     Finance and banking
Edm(n)     Construction and manufacturing
Ede(n)     Education
EdE(n)     Engineering
Eds(n)     Science
```

```
Edc(n)     Computing
Edh(n)     Health
Edp(n)     Philosophy
EdM(n)     Military
Edl(n)     Law
Edg(n)     Government
Edd(n)     Media
Edn(n)     Nature
Edr(n)     Recreation and leisure
EdS(n)     Sales
Edt(n)     Transportation
Ed!(n)     None
EdO(n)     Other
Ed?(n)     I think I know something but I don't know what
```

Hobbies (Hb)

```
HbSailing
HbMotorbikes
HbTV.Gilligans_Island
```

Politics (Pl)

```
PlC Communist
Plm Marxist
Pln Leninist
PlD Democratic
PlR Republican
PlL Labour
PlT Tory
PlG Green
Pll Libertarian
Pl{L} Left/liberal
Pl{R} Right/conservative
Pl{L}.Far Far left
PlM Moderate or Centrist
Pl! Anarchist
PlO Other (ie. PlO.Theocracy)
```

Transportation (Mv)

```
MvD Drive
MvW Walking
MvS Scooter
Mvs 'Segway'
MvM Motorbike
MvB Public transit
MvH Hired transport (taxi, limo)
MvL School Bus
MvA Vehicle capable of air travel
MvQ Vehicle capable of water travel
MvO Wheelchair
Mv-Other (specify with trailing .)
MvD.Car.Ford_Aspire Drive a specific vehicle
```

Relationships (Rl)

```
RlM Married
RlC Living together or Common law
RlE Engaged
RlD Dating
RlW Widow/widower
RlP Polygamous
Rlx Divorced
Rl! Single
Rl+ Seeking
Rl-Not seeking
RlD&+ Divorced and seeking
```

Children (Kd)

Kd - Children: KdnG, where n-number and G-gender (example: Kd2x1y). Further specifications may be added with a trailing period: "Kd!.Never" or "Kd2x1y.brats". Alternately:

```
Kd!      no kids
Kd?      Possible kids
Kd>      One (or more) on the way
Kd*      Too many kids
Kd+      Someone else's kids (Stepchildren)
```

Pets (Pe)

```
PeA     Amphibian (frog, salamander...)
PeB     Avian
PeN Arachnoid
PeI     Insect
PeR     Reptile
Per     Rodent
PeF     Fish
PeM     Mollusc (clam, snail, scallop, squid...)
PeP Plant
PeO     Other
PeC     Cat
PeD     Dog
PeS     Snake
PeH     Hare or Rabbit
PeT Rat
```

Myers-Briggs (MB)

MB - Myers-Briggs Personality type, These are 4 letter codes made of the opposites of Introvert-Extrovert, Intuitive-Sensing, Feeling-Thinking, Judging-Perceiving. Add modifiers with a trailing period (ie. MBINTP.Extreme_Introvert) These are all the possible combinations:

```
MBENFP
MBINFP
MBENFJ
MBINFJ
MBENTP
MBINTP
```

```
MBENTJ
MBINTJ
MBESTJ
MBISTJ
MBESFJ
MBISFJ
MBESTP
MBISTP
MBESFP
MBISFP
```

Facial Hair (FH)

```
FHb full facial beard
FHg goatee
FHp soul patch
FHm mustache
FHs sideburns
FHS stubble
FH! no facial hair
FHo other
FHm.handlebar handlebar mustache
```

Body Adornments (BA)

```
BAT Tattoo
BAP Piercing
BAB Branding
BAS Scarring (ritual or otherwise)
BAI Implants
BAl Stretching
BAO Other
```

UserFriendly (UF)

UF - Your connection to a UserFriendly character or your similarity to a UF character. Specify further with a trailing dot (ie. UFPitr.Evilness)

```
UFAJ
UFPitr
UFMike
UFStef
UFGreg
UFMiranda
UFDustPuppy
UFSmilingMan
UFChief
UFSid
UFMatt
UFCrudPuppy
UFHillary
UFErwin
UFArtur
UFPearl
```

```
UFIlliad
UFCthulhu
UF*             All of them
UF!             Dislike UF
UF?             UF? What's that? (or just omit this specification)
```

Internet Use (IN)

```
IN16.Cable
IN!             does not use internet
```

Addictions (Ad)

```
AdC     Caffeine
AdN     Nicotine
AdM     Marijuana
AdA     Alcohol
AdI     Internet
AdG     Gambling
AdS     Sex
Ad+     More than I can fit here
Ad*     Everything
Ad!     No addictions
Ad?     Addicted? Me?
AdO     Other (ie. 'AdO.Chocolate')
```

Programming Languages (Pr)

```
Pr?                     Programming? What's that?
Pr!                     I don't do programming.
Pr* I program in all languages
PrPascal Pascal
PrPascal.Turbo Turbo Pascal
PrPascal.Delphi Delphi (Object Pascal)
PrPHP PHP
PrPerl Perl
PrBASIC BASIC
PrBASIC.Visual Visual Basic
PrC C
PrC++ C++
PrC# C Sharp
PrHTML HTML
PrXML XML
PrCOBOL COBOL
PrJava Java
PrJavaScript JavaScript
PrSGML SGML
PrRuby Ruby
PrRuby.on_Rails Ruby on Rails
```

10.3.4 Examples

Simple Example

This person is a 180 cm, 75 kg male, born in 1992, with:

Light/medium skin, brown hair, and green eyes
Normal 20/20 vision
Heterosexual
They speak:
English fluently (level 9)
French at an intermediate level (level 3)
Their career field is computing, at level 6 of proficiency.
They enjoy motorbikes as a hobby.
They usually drive, specifically a Toyota Corolla.
They are currently dating.
They have a cat as a pet.
Their Myers-Briggs type is INTJ.

```
sxy cm180 kg75 skEAD5C9 ha8A6F4A ey00FF00
es= sp= Ag1992
LAEN(9)&LAFR(3) Crc(6) HbMotorbikes
MvD.Car.Toyota_Corolla RlD
PeC MBINTJ
```

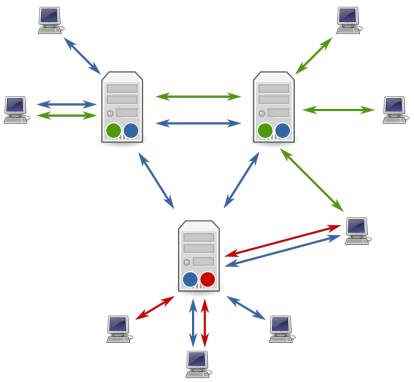
Complex Example

This person is a 165 cm, 60 kg female, born in 1985, with:
Light skin, blonde/yellow hair
Right eye green, left eye grey
Farsighted vision
Bisexual
Mixed European and Celtic ancestry
She lives at coordinates around New York City and is in timezone UTC5, observing daylight savings.
Her astrological signs are:
Gemini (Western zodiac)
Horse (Chinese zodiac)
Religion: Hindu.
She speaks:
English fluently (9)
Japanese well (5)
French moderately (4)
Her career is in the sciences, specifically biology, proficiency level 7. She has education level 8 in genetics.
Hobbies include:
Sailing
Watching Star Trek
Her political stance is moderate left-leaning.
She usually bicycles for transportation.
Relationship status: married.
Children: two girls and one boy, humorously described as brats.
Pets: cat and dog.
Myers-Briggs type: ENFP.
Body adornments:
A waves tattoo on her leg
Internet access:

Broadband fiber (speed class 5)
Programming:
Python (with Django)
C++
Addictions:
Caffeine
Tea Earl Grey, hot (a Star Trek reference)

```
sxx cm165 kg60 skF2D3C0 haFFCC00
ey{R}00FF00&{L}888888
eso sp* Ag1985 anE&anC
Lo40.7128,-74.006 GM-5DN
ZoG ZoH
rlh LAEN(9)&LAJA(5)&LAFR(4)
Crs(7).Biology EdS(8).Genetics
HbSailing&HbTV.Star_Trek
Pl{L}.Moderate
MvB RlC Kd2x1y.brats
PeD&PeC
MBENFP
BAT.leg.waves
IN5.Fiber
PrPython&Django;PrC++
AdC.Tea.Earl_Grey_Hot
```

10.4 yEnc (2001)



In 2001, Jürgen Helbing started developing a binary-to-text encoding scheme with the purpose of reducing the overhead over the existing encoding methods like uuencode and Base64. For example Base64 encodes three bytes of printable characters into four bytes of printable characters, causing an overhead of 33%. On the other side yEnc (named derived from the wordplay "Why Encode"), tends to use only one character to represent one byte of the original data, with some exceptions, causing an average overhead of only 1/2%. In 2003 yEnc became the de facto standard encoding system for binary files on Usenet.

The encoding principle is the following: each octet of input data is encoded by a single output character using the following simple formula:

$$Output = (Input + 42) \tag{10.1}$$

In this way, NULL characters (ASCII 00h) that are usually a lot in binary messages, are converted in different characters and there is no more the necessity to escape them. A drawback is that there is the possibility that some other characters will be encoded with the NULL character or some other

characters that needs to be escaped. If this happens they will be escaped with the special character "=" (ASCII 3Dh). This characters includes:

- ASCII 00h (NULL);
- ASCII 0Ah (LF);
- ASCII 0Dh (CR);
- ASCII 3Dh (=);
- ASCII 09h (TAB) →removed in version 1.2 of the protocol.

In average the probability of occurance of these 4 critical characters is about 0.4%, causing an overall average overhead of 1.6% for the entire message. The TAB has been removed from the list of the critical characters because it can be handled in a smart way using spaces (ASCII 20h).
To summarize the encoding steps are the followings:

- Fetch one character from the input stream;
- Increment it by 42 modulo 256;
- If the result character is a critical one, output the escape character and increment the critical character by 64 modulo 256;
- Output the resulting character;
- Repeat from the start for each next characters.

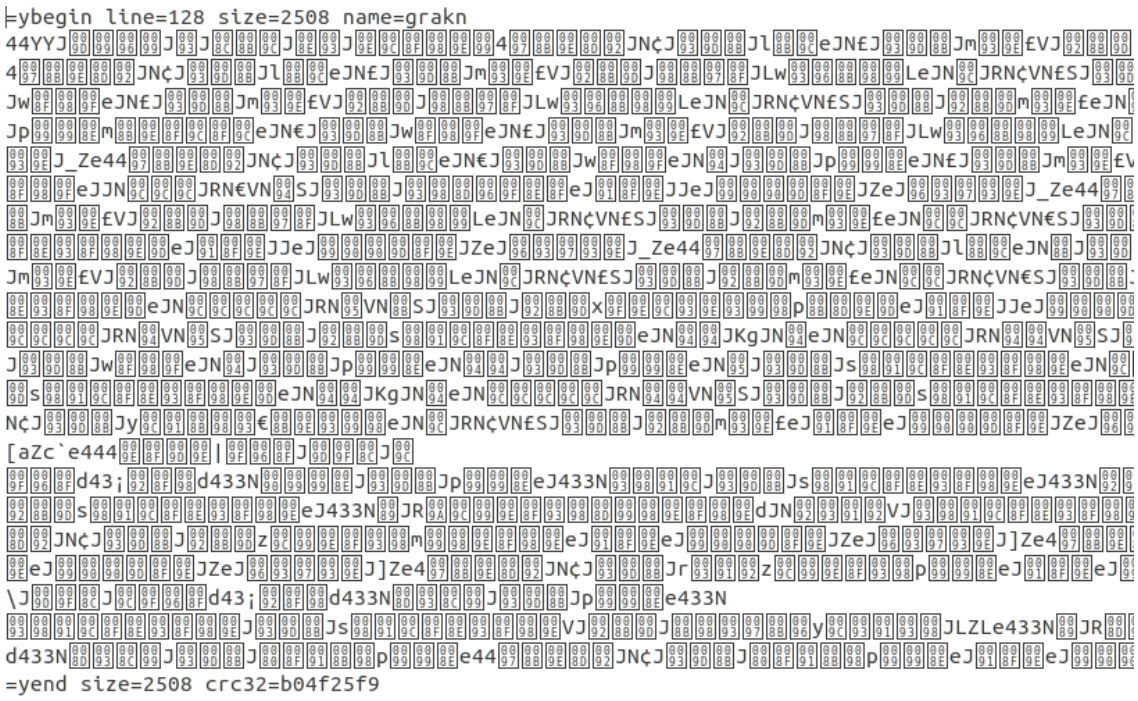
To recognize that a file has been encoded using yEnc, a special header and trailer line are attached to the file. The header line must begin with the escape character "=" followed by the character "y". Since the "y" character is not a valid critical character the decoder will be able to recognize that line as a keyword line, header or footer. Additionally the header line must begin with the string "ybegin" and contains information about the length of the line, the size and the name of the encoded file, so that a typical yEnc header looks like:

=ybegin line=128 size=123456 name=mybinary.dat

The trailer line must always begin with the string "yend" and must contain the size of the the original unencoded binary file. The size is repeated also in the trailer for redundancy checking in order to detect file corruptions. To verify the integrity of the file, the trailer can include also a 32-bit Cyclic Redundancy Check (CRC) value. A typical yEnc trailer looks like:

=yend size=123456 crc32=abcdef12

A yEnc file looks like this:


=ybegin line=128 size=2508 name=grakn
=yend size=2508 crc32=b04f25f9

IV Cryptography

11. Classical Cipher

11.1 Rot-n

One of the widest used cipher for beginning cryptography challenges, rot-n refers to a family of ciphers based on alphabet rotation. For the Rot-n ciphers family, when we talk about rotation, we mean that the second half of the alphabet represent the encryption of the first half of the alphabet. The set of symbols involved into the alphabet may differ according to different languages, and the complexity can be improved by adding also special symbols and digits to the alphabet. The most notable property of this kind of ciphers is that the encryption algorithm is the same as the decryption one:

$$Rot_n(Rot_n(x)) = x \tag{11.1}$$

Where Rot-n(x) is defined as:

$$Rot_n(x) = (x+n) \pmod{2n} \tag{11.2}$$

11.1.1 Rot-5

When the alphabet is only composed by digits from 0 to 9, the rotation cipher is called Rot-5. In this case the number 0 is encoded by the number 5, the number 1 by the number 6 and so on according to the following table:

0	1	2	3	4
5	6	7	8	9

Challenge 11.1 Decode: 297584 ■

Solution 11.1 Decoded: 742039

```
static String rot5(String text) {
   StringBuilder sb = new StringBuilder();
   for (int i = 0; i < text.length(); i++) {
      sb.append((char) ((text.charAt(i) - '0' + 5) % 10 + '0'));
   }
   return sb.toString();
}
```

11.1.2 Rot-13

When the alphabet is only composed by english letters from a to z, the rotation cipher is called Rot-13. In this case the letter 'a' is encoded by the letter 'n', the letter 'b' by the letter 'o' and so on according to the following table:

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

Challenge 11.2 Decode: guvf zrffntr unf orra rapelcgrq ■

Solution 11.2 Decoded: this message has been encrypted

```
static String rot13(String text) {
   StringBuilder sb = new StringBuilder();
   for (int i = 0; i < text.length(); i++) {
      sb.append((char) ((text.charAt(i) - 'a' + 13) % 26 + 'a'));
   }
   return sb.toString();
}
```

11.1.3 Rot-13.5

Combining the Rot-13 with the Rot-5 we obtain the Rot-13.5.

a	b	c	d	e	f	g	h	i	j	k	l	m	0	1	2	3	4
n	o	p	q	r	s	t	u	v	w	x	y	z	5	6	7	8	9

R Rot 13.5 should result a strange notation. This is a special case in which 2 alphabets are involved. Both of them are encrypted and decrypted separately. According to the Rot-n definition using a single alphabet would have produced the following encryption/decryption schema:

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	0	1	2	3	4	5	6	7	8	9

And would have been named something like Rot-18.

Challenge 11.3 Decode: guvf zrffntr unf orra rapelcgrq jvgu ebg-68 ■

Solution 11.3 Decoded: this message has been encrypted with rot-13

```
static String rot13_5(String text) {
   StringBuilder sb = new StringBuilder();
   for (int i = 0; i < text.length(); i++) {
      char current = text.charAt(i);
      if (current >= '0' && current <= '9') {
         sb.append((char) ((current - '0' + 5) % 10 + '0'));
      } else if (current >= 'a' && current <= 'z') {
         sb.append((char) ((current - 'a' + 13) % 26 + 'a'));
      }
   }
   return sb.toString();
}
```

11.1.4 Rot-47

Rot-47 takes in consideration 94 consecutive symbols of the ASCII table, from '!' at position 33 to ' ' at position 126 of the table. This alphabet will cover all English upper case and lower case letters, all digits and lots of printable symbols:

!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	0	1	2	3	4	5	6	7	8
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_	‘	a	b	c	d	e	f	g

9	:	;	<	=	>	?	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~

Challenge 11.4 Decode: %9:D >6DD286 92D 366? 6?4CJAE65 H:E9 #@E\cf ■

Solution 11.4 Decoded: This message has been encrypted with Rot-47

```
static String rot47(String text) {
   StringBuilder sb = new StringBuilder();
   for (int i = 0; i < text.length(); i++) {
      sb.append((char) ((text.charAt(i) - '!' + 47) % 94 + '!'));
   }
   return sb.toString();
}
```

12. Substitution Ciphers

A substitution cipher is a way to replace single units of the plaintext with the respective symbols of the ciphertext, according to a fixed schema; the "units" may be single letters (the most common), pairs of letters, triplets of letters, mixtures of the above, and so forth. The receiver deciphers the text by performing the inverse substitution.

12.1 A1Z26

One of the easiest way to encode a piece of text is to replace each of it's symbols with the position that it covers in the alphabet in which belongs. Considering for example the English alphabet, the letter A will be encrypted with number 1, because A is the first letter of the alphabet. Letter E will be encrypted with number 5, letter M with number 13 and so on. The encrypted word is alphabet dependent: for example in Italian alphabet (in which J and K are not present) the letter M will be encrypted with number 11 and not 13. The following is the conversion table for English alphabet:

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

t	u	v	w	x	y	z
20	21	22	23	24	25	26

```
static String atbash(String text) {
   StringBuilder sb = new StringBuilder();
      for (int i = 0; i < text.length(); i++) {
         char current = text.charAt(i);
         sb.append((current - 'a') + 1).append(" ");
      }
      return sb.toString().trim();
}
```

Challenge 12.1 Decode: 20-8-5 7-15-22-5-18-14-13-5-14-20 4-15-5-19-14'20 23-1-14-20 1-14-25 19-25-19-20-5-13 15-6 20-18-1-14-19-13-9-20-20-9-14-7 9-14-6-15-18-13-1-20-9-15-14 20-15 18-5-13-1-9-14 21-14-2-18-15-11-5-14, 21-14-12-5-19-19 9-20'19 21-14-4-5-18 9-20-19 15-23-14 3-15-14-20-18-15-12. ■

Solution 12.1 Decoded: The government doesn't want any system of transmitting information to remain unbroken, unless it's under its own control.

12.2 Atbash, Albam, Atbah

In old Sacred texts, and in particular in the Old Testament, can be found 3 examples of substitution ciphers. The first one, called Atbash, was invented by Hebrew population and it was a matter of overturning the alphabet so that letter 'a' is encrypted with letter 'z', letter 'b' with letter 'y' and so on:

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
z	y	x	w	v	u	t	s	r	q	p	o	n	m	l	k	j	i	h	g	f	e	d	c	b	a

```
static String atbash(String text) {
   StringBuilder sb = new StringBuilder();
   for (int i = 0; i < text.length(); i++) {
      sb.append((char) ((25 - (text.charAt(i) - 'a')) + 'a'));
   }
   return sb.toString();
}
```

The second one, called Albam, is another name to identify the Rot-13 cipher where the first half of the alphabet is encrypted by the second half. The last one is quite more complex. The Atbah substitution have to satisfy a numeric relation. The first nine symbols are substituted in a way in which the sum of plaintext symbol and the ciphertext symbol gives 10 (Considering a=1, b=2, ..., z=26). For the next nine symbols the same rule is valid, but the sum must give 28. For the last 8 symbols, again, the same rule is valid and the sum must give 45:

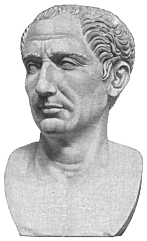
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
i	h	g	f	e	d	c	b	a	r	q	p	o	n	m	l	k	j	z	y	x	w	v	u	t	s

```
static String atbah(String text) {
   StringBuilder sb = new StringBuilder();
   for (int i = 0; i < text.length(); i++) {
      char current = text.charAt(i);
      if (current <= 'i') {
         sb.append((char) (8 + 2 * 'a' - current));
      } else if (current <= 'r') {
         sb.append((char) (8 + 2 * 'j' - current));
      } else {
         sb.append((char) (7 + 2 * 's' - current));
      }
   }
   return sb.toString();
}
```

It can be easily noticed that according to previous rules the encryption of symbols 'e' and 'n' are the symbols themselves. To avoid this some variants propose to encrypt symbol 'e' as 'n' and symbol 'n' as 'e':

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
i	h	g	f	**n**	d	c	b	a	r	q	p	o	**e**	m	l	k	j	z	y	x	w	v	u	t	s

12.3 Caesar Cipher (100 B.C. - 44 B.C.)



One of the widely known substitution cipher came to us from the "Da vita Caesarum", where the emperor was used to write secret messages replacing each letter by a letter some fixed number of positions down the alphabet. For example, with a shift of 3, letter 'a' becomes 'd', letter 'b' becomes 'e' and so on till letter 'x' returns to 'a', letter 'y' to 'b' and 'z' to 'c'. The encryption and the decryption of this simple cipher can be synthesize using modular arithmetic. For the English alphabet, which is composed by 26 letters, we have that, choosing an encryption key 'n' (the number of shifts), the encryption algorithm become:

$$E_n(x) = (x+n) \pmod{26} \tag{12.1}$$

And the decryption algorithm will be:

$$D_n(x) = (x-n) \pmod{26} \tag{12.2}$$

R It is easy to notice that with a key of 13, we obtain the Rot-13 cipher (Recall from 3.2):
$Rot_n(x) = (x+n) \pmod{2n}$
$with \quad n = 13$
$Rot_n(x) = (x+n) \pmod{2*13}$
$Rot_n(x) = (x+n) \pmod{26} \longleftrightarrow E_n(x) = (x+n) \pmod{26}$

Thus an example of encryption schema for key = 5 is:

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
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

Challenge 12.2 Decode: Ymnx mfx gjjs jshwduyji bnym Hjfxfw hnumjw ■

Solution 12.2 Decoded: This has been encrypted with Ceasar cipher

12.4 Affine Cipher

Affine cipher is generalization of the Caesar cipher, providing a bit more security but maintaining the flaws of a classical substitution cipher. As usual, letters of the English alphabet are represented by numbers from 0 to 25 but this time two different keys are used in order to encrypt and decrypt the message. The algorithms for encryption and decryption are the following:

$$E_{a,b}(x) = (a*x+b) \pmod{26} \tag{12.3}$$

$$D_{a,b}(x) = a^{-1}*(x-b) \pmod{26} \tag{12.4}$$

where a^{-1} is the modular multiplicative inverse of a modulus m (26 in this case). It can be easily computed by solving:

$$1 = a * a^{-1} \pmod{26} \tag{12.5}$$

The main problem is that the multiplicative inverse of a exists only if a and m are coprime. Choosing a that is not coprime with m makes the decryption impossible.

R With $a = 1$ and $b = n$ the affine cipher is actually the Caesar cipher:
$E_n(x) = E_{1,b}(x)$
$(x+n) \pmod{m} = (1*x+b) \pmod{m}$
$(x+n) \pmod{m} = (x+b) \pmod{m}$ Since $b = n$ the both sizes are equals. Since 1 is coprime with every possible modulus m, a is a valid key that makes the ciphertext decryptable.

We can now proceed showing an example. Suppose we want to encrypt the message "affinecipher" using keys $a = 3, b = 17$. First step is to encode the message as numbers so that "affinecipher" becomes 0 5 5 8 13 4 2 8 15 7 4 17. Then, for each number, the formula is applied resulting in 17 6 6 15 4 3 23 15 10 12 3 16. Converting back to chars we obtain "rggpedxpkmdq". The next table will summarize the process:

plaintext	a	f	f	i	n	e	c	i	p	h	e	r
numerical x	0	5	5	8	13	4	2	8	15	7	4	17
$3*x+17$	17	32	32	41	56	29	23	41	62	38	29	68
$3*x+17 \pmod{26}$	17	6	6	15	4	3	23	15	10	12	3	16
ciphertext	r	g	g	p	e	d	x	p	k	m	d	q

Now to decrypt the ciphertext, as seen before, we have to compute a^{-1}. In our case 9 is the multiplicative inverse of 3 modulus 26. In fact $3*9 \pmod{26} = 1$. The decryption algorithm is so $D_{3,17} = 9*(x-17)$. Let's reconstruct the process table:

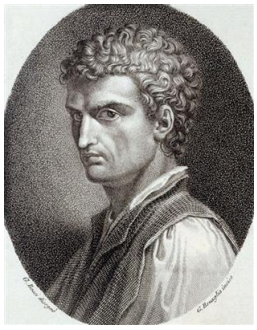
ciphertext	r	g	g	p	e	d	x	p	k	m	d	q
numerical x	17	6	6	15	4	3	23	15	10	12	3	16
$9*(x-17)$	0	-99	-99	-18	-117	-126	54	-18	-63	-45	-126	-9
$9*(x-17) \pmod{26}$	0	5	5	8	13	4	2	8	15	7	4	17
plaintext	a	f	f	i	n	e	c	i	p	h	e	r

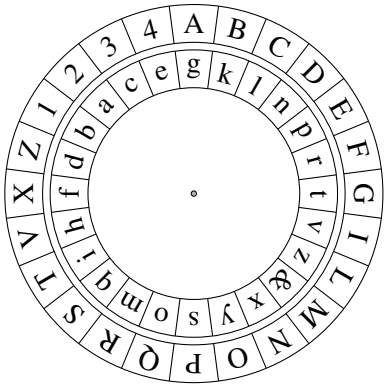
13. Polyalphabetic Substitution Ciphers

The concept behind polyalphabetic substitution ciphers is very similar to the one behind the simplest substitution ones but with the main difference that more than one alphabet is used to build the ciphertext. Al-Qalqashandi was the first to theorize this kind of encryption in the "ub al-a sha" around the end of the XIV century.

13.1 Alberti Cipher (1467)

In 1467 born, by the hand of an Italian architect Leon Battista Alberti (1404-1472), the first example of polyalphabetic cipher with a practical use. Alberti was an humanist author, artist, architect, poet, priest, linguist, philosopher and cryptographer, an important figure of the Renaissance. He described its cipher in the "De cifris" treat, proposing two concentric disks: the external one, stationary, called "Stabilis" and the internal one, movable, called "Mobilis". The whole device was called by Alberti "Formula". The Stabilis is divided into 24 blocks with a total of 20 uppercase alphabet letters and 4 digits. The Mobilis is also divided in 24 block but contains only lowercase letters. The following is an accurate representation of the Formula with the original arrangement of both Stabilis and Moilis:



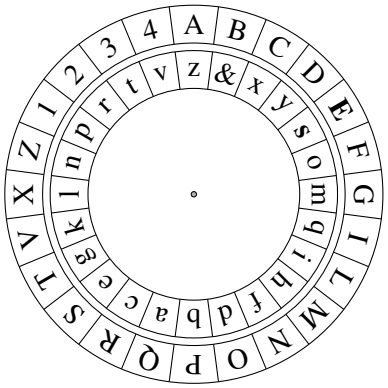


The outside digits are used to change the current encrypting alphabet. There are 2 main methods of encryption, that were also described by Alberti in his treat: fixed index and mobile index..

13.1.1 Fixed Index

In the fixed index method an uppercase letter of the Stabilis and a lowarcase letter of the Mobilis are chosen as initial key. The Mobilis is rotated till the lowercase letter matches the uppercase one that we chose. The lowercase letter is appended to the ciphertext and the encryption can begin. Each letter of the plaintext (represented by the Stabilis) is simply replaced by the corresponding one in the Mobilis.

For example let's encrypt "LEONBATTISTAALBERTI" using index 'E' and 's'. Let's move the Mobilis so that 's' matches 'E':



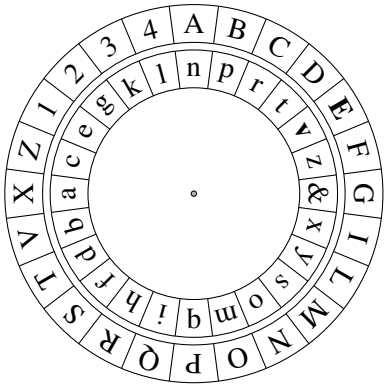
Now proceed replacing single uppercase plaintext letters with lowercase letters so that 'L' becomes 'i', 'E' becomes 's', 'O' becomes 'd' and so on. Obtaining:

_	L	E	O	N	B	A	T	T	I	S	T	A	A	L	B	E	R	T	I
s	i	s	d	f	&	z	g	g	q	e	g	z	z	i	&	s	c	g	q

So far the cipher is not far away to be a simple substitution one. Complexity come when numbers are used. Numbers are used to rearrange the Mobilis and thus building a new encryption alphabet: when a number is encountered it is ciphered as usual using the corresponding symbol in the Mobilis but later the Mobilis will be rotated till this last symbol matches the chosen index in the Stabilis. For example we want to encrypt "LEONBATTI4STAALBERTI" using index 'E' and 's'. Let's move the Mobilis so that 's' matches 'E' as before, we obtain:

_	L	E	O	N	B	A	T	T	I	4
s	i	s	d	f	&	z	g	g	q	v

We encountered a number. Let's encrypt it (4=v) and then rotate the Mobilis so that 'v' matches the Stabilis index ('E'):



No we can proceed encrypting the rest of the plaintext using the new alphabet obtaining:

S	T	A	A	L	B	E	R	T	I
f	d	n	n	y	p	v	h	d	x

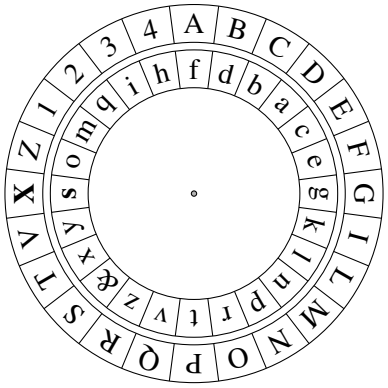
Putting all together, using 'E' and 's' as indexes we encrypted "LEONBATTI4STAALBERTI" as "sisdf& zggqvfdnnypvhdx". Considering that this time two different alphabets have been used in the encryption, the secret is much stronger than before. The decryption is really easy. Knowing the indexes just follow the reverse process mapping Mobilis lowercase symbols to Stabilis uppercase ones and rotate the Mobilis whenever a number is encountered.

13.1.2 Mobile Index

Differently from the previous one, the mobile index methods forces to change alphabet every n letters instead of when a number is encountered. We start choosing as always a pain of indexes. We start appending the uppercase letter to the ciphertext and proceed with encryption simply mapping Stabilis symbols to Mobilis ones. At the $n-th$ letter we choose another index of the Stabilis, we append it to the ciphertext and we rotate the Mobilis till the old lowercase index and the new uppercase one match. We proceed in this way until all the plaintext has been encrypted. Let's do an example: for simplicity let's take the ones that we used before: 'E' and 's'. We want to encrypt the word "MESSAGE" and rotate every 4 letters. Encrypting the first part of the plaintext is trivial:

_	M	E	S	S
E	h	s	e	e

Now we choose another index on the Stabilis, let's say 'X'. We appent 'X' to the ciphertext and rotare the Mobilis till 'X' matches our lowercase index 's'



No proceed using the new alphabet:

_	A	G	E
X	f	g	c

Finally we obtained the full ciphertext "EhseeXfgc" starting from plaintext "MESSAGE", Stabilis index 'E', Mobilis index 's' and changing to Stabilis index 'X' after enciphering 4 letters.

13.2 Tabula Recta (1508)

In 1508, Johannes Trithemius, a German monk, set up the Tabula Recta that can be defined as a square of 26 rows x 26 columns. Each row contains a different alphabet, the same as the previous one but shifted by a position to the left. The Tabula is the basis of lots of ciphers and it is so composed:

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

13.3 Trithemius Cipher (1518)

Together with the Tabula Recta, Trithemius explained the first way to use it in his book "Polygraphia" which was published in 1518 after his death. He described a polyalphabetic cipher with the same principle of work as the Alberti's cipher disk, but with the difference that the order of the letters in the alphabets is not scrambled.

It works as follows. We choose a row where to start. This row represents the firs encryption alphabet. Next we choose a rotating direction (up or down). We start encrypting the first symbol of the plaintext (always identified by the column) with the corresponding symbol in the Tabula in the chosen alphabet. Then, the next symbol of the plaintext will be encrypted using the next alphabet in sequence (or the previous one if we chose the up direction). Let's make an

example:

Suppose we want to encrypt the world "TABULA" and we choose the initial alphabet identified with letter 'M' and rotation down. We start encrypting the first symbol so that 'T' becomes 'F'. Now the alphabet changes, the new one to be used is the one identified with letter 'N'. Now encrypt the second letter: 'A' becomes 'N'. The next alphabet will be the one identified with letter 'O' sot that the third letter of the plaintext "B" will be encrypted with letter 'P'. And so on and so forth. At the end we have that "TABULA" $\rightarrow$"FOPJBR"

13.4 Vigenere Cipher (1553)

The Vigenere cipher can be seen as an improvement of the Trithemius cipher. The main problem with this previous one is that encryption alphabets follow an exact sequence, from top to bottom or from bottom to top. This means that, knowing the Tabula, it is sufficient to guess the starting alphabet and the rotation direction in order to decrypt it: a total of 52 guesses are enough! To solve this issue, the Vigenere cipher uses a key in order to identify which alphabets have to be used during the encryption. It is easy to see how guessing the key is a more complex task.

Figure 13.1: Giovan Battista Bellasio

The cipher was first described by Giovan Battista Bellaso in 1553 in his book "La cifra" but became famouse only after that Blaise de Vigenere published it in the court of Henry III of France in 1586. Even if the original author was Bellaso the cipher was attributed to Vigenere and so took his name. In reality Vigenere described a variant of the cipher called "Autokey cipher" that will be discussed in the next section.

The encryption using the Vigenere cipher works as follows: choose a key and replicate it until we reach the end of the message we want to encrypt. Now, using the Tabula Recta, encrypt each symbol of the plaintext (identified as before by columns) finding the symbol that intersect the row of the key. For example we want to encrypt the word "EXTRAORDINARY" using key "BATH". We need to extend the key until it is the same length of the plaintext. Our new key will be "BATHBATHBATHB". The encryption can start: column 'E' and row 'B' produces 'F', column 'X' and row 'A' produces 'X', column 'R' and row 'T' produces 'K' and so on.

plaintext	E	X	T	R	A	O	R	D	I	N	A	R	Y
key	B	A	T	H	B	A	T	H	B	A	T	H	B
ciphertext	F	X	M	Y	B	O	K	K	J	N	T	Y	Z

So using key "BATH" the plaintext "EXTRAORDINARY" has been encrypted as "FXMYBOKKJNTYZ".

Challenge 13.1 — 1 char key. Sx mywlsxsxq dro locd yp rsc dgy zbonomoccybc dro wshon kvzrklod yp Kvlobds kxn dro voddob-li-voddob oxmszrobwoxd yp Dbsdrowsec gsdr rsc ygx lbsvvskxd snok yp k vsdobkv uoi, ro mbokdon dro wynobx myxmozd yp zyvikvzrklodsm celcdsdedsyx ■

Solution 13.1 — 1 char key. In combining the best of his two predecessors the mixed alphabet of Alberti and the letter-by-letter encipherment of Trithemius with his own brilliant idea of a literal key, he created the modern concept of polyalphabetic substitution.

Challenge 13.2 — 3 chars key. Upk dzeqbuhzgn kuobgjvy upk ffvmitbbopv cig zxw hbtrt, wtf qt jzuo ite wtf qt xwue, lxpxvfl lsws b pohp vmiif eomt lbtr pv zim mswaol gu bnf agnm zjuk. ■

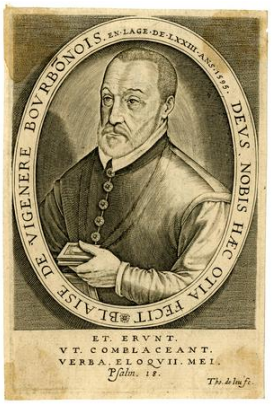
Solution 13.2 — 3 chars key. The cryptogram contains the explanation why two balls, one in iron and one in wood, dropped from a high place will fall on the ground at the same time.

13.5 Autokey Cipher (1586)

Invented in 1586 by Blaise de Vigenere is based on the same concept behind the Vigenere cipher. The only difference is ho the encryption key is build. While in the Vigenere cipher the chosen key is replicated several times to match the plaintext length, in the Autokey cipher, as the name suggests, the part key required to fit the plaintext length is generated automatically using the plaintext itself. Let's suppose we want to encrypt the word "IMMAGINATION" using the key "GREEN". The key is too short and so plaintext is added in order to make it longer: "GREENIMMAGIN". We can now proceed encrypting the plaintext as we saw in the Vigenere cipher:

plaintext	I	M	M	A	G	I	N	A	T	I	O	N
key	G	R	E	E	N	I	M	M	A	G	I	N
ciphertext	O	D	Q	E	T	Q	Z	M	T	O	W	A

The Autokey cipher is more robust than the Vigenere cipher. Generally speaking each cipher that uses a stream as a key with respect to a cipher that repeats its key several times is more robust. This due to the fact that it avoids potential patterns in ciphertext.



Challenge 13.3 — 4 consecutive chars key. Wzprdb fvpcvfegtdnp bjpjem carf gpbl af gfs hhuggdgu mcmjwqntsg cp goifutcsd ktyavfegtdnp. ■

Solution 13.3 — 4 consecutive chars key. Modern cryptography builds upon many of the concepts introduced in classical cryptography.

13.6 Quagmire

Quagmire is a family of polyalphabetic ciphers with 2 or more encryption keys. The number of substitution alphabets is given by the length of one of the keys. Another key usually define hot the alphabets are arranged.

13.6.1 Quagmire I

In the specific Quagmire I have 2 keys: the first is constructed from a keyed plaintext alphabet created from a keyword to which duplicate symbols are removed and all the remaining symbols of the alphabet are appended, ordered, at the end. In this way the keyword "RUBBER" will form the key "**RUBE**ACDFGHIJKLMNOPQSTVWXYZ".

A second keyword is used for define n ciphertext alphabets where n is the the length of the keyword. Each alphabet is a normal "A-Z" one. A final placeholder symbol is used to shift each alphabet. If for example the keyword is "ANIMAL" and the placeholder symbol is "M" the arrangement works in the following way:

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

Once the encryption alphabets are built it is possible to encrypt the message. The first symbol of the plaintext message will be encrypted using the first encryption alphabet, the second symbol with the second alphabet and so on cycling. In this way, maintaining the above configuration, the word "FESTIVAL" becomes "PYJOSOMI"

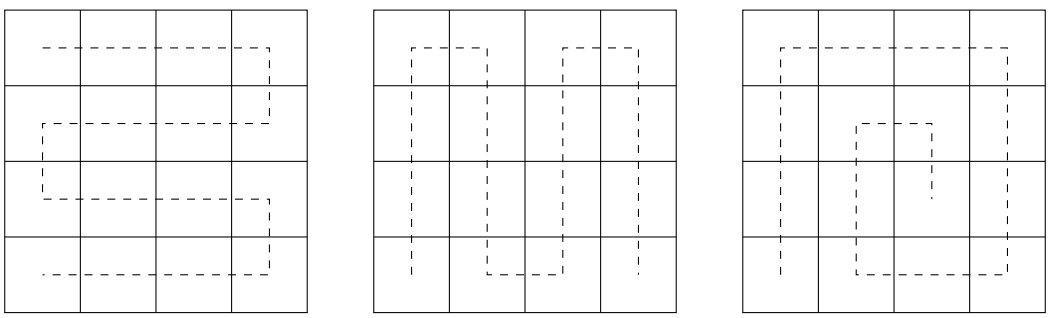
13.6.2 Quagmire II

14. Transposition Ciphers

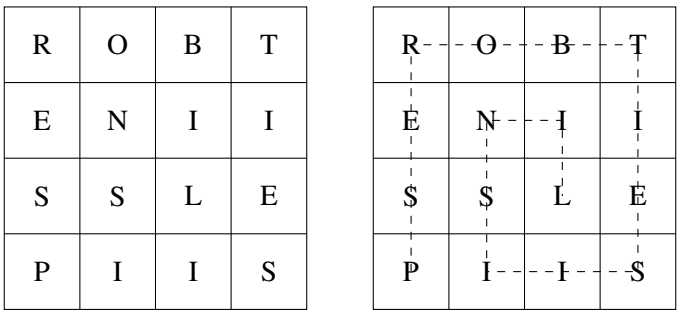
So far we talk about ciphers that are build by substituting plaintext symbols with other symbols generated by an algorithm and an encryption key. On the other side, a transposition cipher does not substitute original plaintext symbols, but it scramble them, or group of them, moving from a position to another. In this way the ciphertext is exactly a permutation of the plaintext that is obtained by applying an algorithm and an encryption key. The decryption algorithm will unscramble the ciphertext in order to reassemble the plaintext.

14.1 Route Cipher

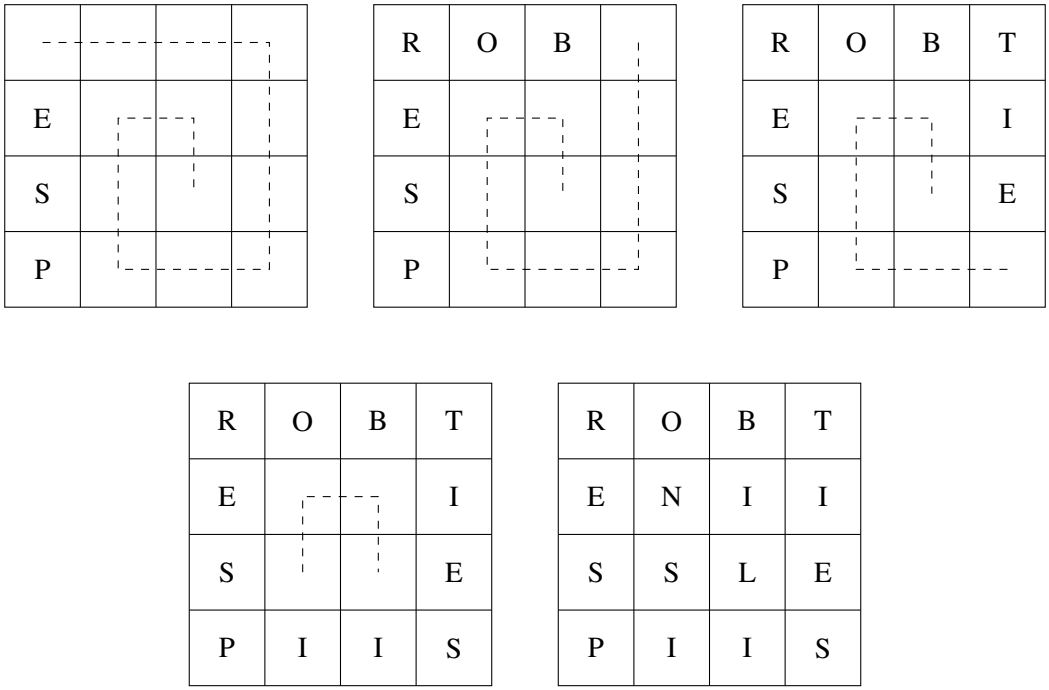
Route cipher, or Serpentine ciphers, usually refer to a family of transposition ciphers in which the ciphertext is obtained by following a continuous path on a nxm grid in which the plaintext has somehow been arranged. The decryption process consists in rebuilding the grid from the ciphertext stream and read back the plaintext message. Plaintext is usually arranged on the grid line by line or column by column starting from the upper left most cell, but more complex arrangements can be done in order to make the encryption stronger. The routes used to cover all cells of the grid for building the ciphertext can also be various, from linear ones to spiral ones. Here are showing some of the simplest possibilities:



Let's try to encrypt something using the "spiral route cipher". Usually the encryption and the decryption algorithm refers in the way the route is computed, in this case we use the "spiral" algorithm. But the algorithm alone is not enough to encrypt the plaintex neither to decrypt the generated ciphertext, we need to know the sizes of the grid. This will be our encryption and decryption key, let's say 4x4 grid. The plaintext we want to encrypt is just the word "RESPONSIBILITIES". Let's start arrange the letters in the grid column by column starting from the upper left most cell and read the spiral from the lower left most cell:



The ciphertext will be "PSEROBTIESIISNIL". How to recover the plaintext from the ciphertext? If the algorithm and the key are known it is easy to reverse the process and build the original grid. We know that the algorithm is "spiral" and the key is 4x4, so start rearrange the letters:



Reading the grid again, column by column starting from the upper right cell we obtain back the original message. Usually the length of the message cannot be arranged exactly in a chosen size nxm grid, specifically when the plaintext's length is a prime number. In these cases you can

consider to keep or remove punctuation from the message or add some placeholder characters at the beginning or at the end to fill the grid completely.

Challenge 14.1 — The american road dream.
RIXTAERDHTXOUTHEOICFREYTESMRAAXMOOIS ■

Solution 14.1 — The american road dream.
The encryption algorithm used is a column snake (similar of the one in example 2 but mirrored vertically), starting from the upper left most cell. The key is a 6x6 grid. The last 'X' symbols has been used to fill all the cells of the grid:
ROUTESIXTYSIXTHEMOTHERROADOFAMERICAX
ROUTE 66 THE MOTHER ROAD OF AMERICA

14.2 Columnar Transposition Cipher

Another transposition technique is provided by the columnar transposition cipher. As the route cipher, the plaintext is arranged into a grid, this time row by row starting from the upper left most cell. The number of the columns of the grid is given by the length of the key used to encrypt and decrypt the message. So if the key is "CAT" the message will be arranged into a nx3 grid (where n is the number of rows and 3 is the number of columns). Once the plaintext has been arranged the order of the columns will be scrambled according to the key. The shuffle follows the lexicographical order of the letters in the key, so if the key is "CAT" the column corresponding to letter "A" will become the first of the grid, column corresponding to letter "C" will be the second and the last column remains there. Then the ciphertext is obtained by reading the grid column by column starting from the upper left most cell. Also in this cipher, like the previous one, there can be situation in which the grid is not fulfilled. In these cases some placeholders can be used, or just random text.

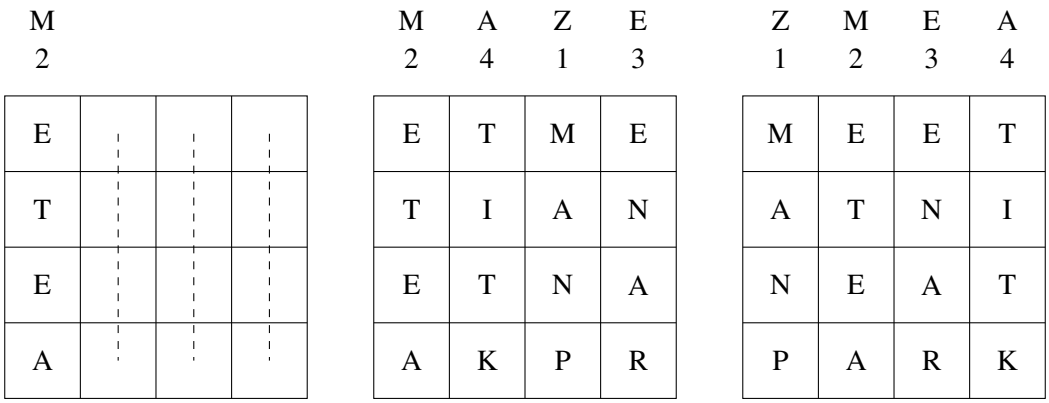
Consider the following example. We want to encrypt the message "meet at nine at park" using the key "MAZE". Start arranging the grid and shuffle according the lexicographical order of the letters composing the key:

M	A	Z	E
3	1	4	2
M	E	E	T
A	T	N	I
N	E	A	T
P	A	R	K

A	E	M	Z
1	2	3	4
E	T	M	E
T	I	A	N
E	T	N	A
A	K	P	R

Now we can proceed reading the cipherthext column by column obtaining "ETEATITKMANPENAR". Decrypting the message is very easy if you know the key. First we need to fill the grid that will have $length(key)$ columns and $length(ciphertext)/length(key)$ rows. The filling starts from

the upper left most cell and proceeds column by column. The key is put on the top of the grid as done in encryption mode but with the difference that it is ordered in reverse lexicographical order. Then we just need to shuffle the columns according to the defined order and read the plaintext message row by row:



We obtain back the message "MEET AT NINE AT PARK". Notice that every key with the same lexicographical order will produce the same result both on encryption and decryption phase.

Challenge 14.2 — Modern War.
EEROSSYOOAONAHLWHRHWFTISLRNWCODTOAFUTRNTYUEAO
Hint: the length of the key is 5 ■

Solution 14.2 — Modern War.
The key used for the cipher is PLANE. The decrypted message is so:
The war we confront today is thus solely a war of honour

Challenge 14.3 — Modern Warfare.
FVRAONINLMSITEYREIECUCFSEYOZWNTTHSEETALKEDRAHLEAIIEFC
UDALEELIVTUGNLIONIEELOHDIGWOIIODIIETOEDUAWEUALRNOVRHU
MTLBARCWDSFNIINBATAIFRNLKTLTCWDWDEOOOH
Hint: the length of the key is 9 ■

Solution 14.3 — Modern Warfare.
The key used for the cipher is FACTORIES. The decrypted message is so:
if enough civilians were killed, factories could not function and if civilians were killed, the enemy would be so demoralized that it would have no ability to wage further war

14.3 Myszkowski Transposition (1902)

A variant of the classical columnar transposition cipher has been proposed by Émile Victor Théodore Myszkowski in his book "Cryptographie indéchiffrable" in 1902. The principle is the same as the original cipher but the idea is to make more confusion and ambiguity by choosing keys that have letters that appears more than once. Suppose we want to encrypt the sentence "The enemy in arriving in Paris" with the key "RADAR". We proceed filling the grid and shuffling columns as

usual:

R	A	D	A	R
3	1	2	1	3
T	H	E	E	N
E	M	Y	I	S
A	R	R	I	V
I	N	G	I	N
P	A	R	I	S

A	A	D	R	R
1	1	2	3	3
H	E	E	T	N
M	I	Y	E	S
R	I	R	A	V
N	I	G	I	N
A	I	R	P	S

At this point, in the original columnar transposition cipher, each column will be read in order to generate the ciphertext "HMRNAEIIIIEYRGRTEAIPNSVNS". Myszkowski method works different and considers columns with the same lexicographical order as a single column where 2 or more letters are stored. In this way reading the first two columns we obtain "HEMIRINIAI", reading the third is easy "EYRGR" and the last two columns encrypt the sequence "TNESAVINPS". The new ciphertext will be "HEMIRINIAIEYRGRTNESAVINPS" that, of course, is different from the previous one. In this case the decryption procedure described above doesn't work. the new procedure works as follows: Start writing the key over the grid in lexicographical order. If more letters inside the word have the same order fill the columns together, otherwise fill the single column. Last phase is to reassemble the key moving the columns in their original position and read back the plaintext row by row:

A	A	D	R	R
1	1	2	3	3
H	E		- -	- -
M	I		- -	- -
- -	- -		- -	- -
- -	- -		- -	- -
- -	- -		- -	- -

A	A	D	R	R
1	1	2	3	3
H	E	E	- -	- -
M	I	Y	- -	- -
R	I	R	- -	- -
N	I		- -	- -
A	I		- -	- -

A	A	D	R	R
1	1	2	3	3
H	E	E	T	N
M	I	Y	E	S
R	I	R	A	V
N	I	G	I	N
A	I	R	P	S

R	A	D	A	R
3	1	2	1	3
T	H	E	E	N
E	M	Y	I	S
A	R	R	I	V
I	N	G	I	N
P	A	R	I	S

14.4 AMSCO (1939)

Variants to simple columnar transposition cipher have been figured out over time. In her "Elementary Cryptanalysis", published in 1939, Helen Fouché Gaines describe a variant proposed by A. M. Scott, a member of the American Cryptogram Association (ACA). It differs from the original by the way the plaintext is arranged in the grid. Each cell of the grid can hold both unigrams and bigrams allowing to use shorter keys to encrypt the same message. Always starting from a bigram we dispose the plaintext in the grid by alternate bigrams and unigrams. Suppose the message to encrypt is "AMSCO transposition cipher" and the key is "ROCK" the we proceed as follows:

R	O	C	K
4	3	1	2
AM	S	CO	T
R	AN	S	PO
SI	T	IO	N
C	IP	H	ER

C	K	O	R
1	2	3	4
CO	T	S	AM
S	PO	AN	R
IO	N	T	SI
H	ER	IP	C

The encrypted text will so be "COSPIOHETONRSNTPAMRASICI".

15. Polygraphic Substitution Ciphers

what is, TODO

15.1 Playfair Cipher (1854)

Charles Wheatstone invented, on 26th March 1854, the first cipher that encrypted pairs of letters, instead single ones. The name comes from Lord Playfair that financed and promoted its use. Thought to be used in telegraph communications, it was initially rejected by the British Foreign Office because of its perceived complexity. The first known use of the cipher was in the Second Boer War and successively in World War I and World War II.

Since the Playfair cipher encrypts digrams instead of single characters, it is more complex to break through frequency analysis since from 26 possible monograms we pass to 25x24 = 600 possible bigrams (in a grid 5x5 where no duplicate letters are allowed). Frequency analysis is still possible but a considerably larger cipher text is required in order to be able to break it.

The Playfair cipher uses a 5x5 keyed Polybious square as a support for the encryption and the decryption algorithm. To encrypt the message it is first broken into bigrams, for example "banana" becomes "BA NA NA", that will be substituted using the table. To messages with an odd number of letters a 'X' will be append to complete the last bigram. Each bigram forms a rectangle inside the encryption grid with the two letters as opposite corners. Considering the generated rectangles, to perform the substitution:

- If both letters of the bigram are the same, substitute the second letter with the "X" and apply

one of the following substitutions according to the shape of the new rectangle;

- If beth letters of the bigram appear on the same row of the polybius square, encrypt them with the letters to their immediate right. If one of the two letters of the original bigram is the right most of the row, substitute it with the left most of the same row;
- If beth letters of the bigram appear on the same column of the polybius square, encrypt them with the letters to their immediate below. If one of the two letters of the original bigram is the bottom one of the column, substitute it with the top one of the same column;
- If the letters composing the bigram are not on the same row and not on the same column, replace them with the letters on the 2 remaining corner of the rectangle or the square. The first letter of the encrypted pair is the one that lies on the same row as the first letter of the plaintext pair.

Imagine we want to encode the word "LETTERS" using the following keyed Polybious square:

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

Since the message has odd length, let's add an "X" to the end of it and generate the bigrams "LE", "TT", "ER", "SX". For the first bigram, both letters are on the same row, so substitute them with the ones immediately on the right: "U" and "X". The second bigram is composed by the same letter so substitute the second one with a "X" and evaluate "TX". Since letter "T" and letter "X" aren't on the same row neither on the same column they form a rectangle:

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

In this case we will encrypt the bigram "TX" using the other two corners of the red rectangle: "NL". The third digram is "ER" that follows the previous case becoming "UK". Also the last digram draws a rectangle and so "SX" become "NU". It's so that the word "LETTERS" is encoded with the sequence of symbols "UXNLUKNU"

15.2 Bifid Cipher (1895)

The Bifid cipher has been presented for the first time by Felix-Marie Delastelle in the "Revue du Genie civil" in 1895 under the name of "Cryptographie Nouvelle". Bifid is a polygraphic, in particular a digraphic, substitution cipher which combines a Polybius square with a very simple transposition cipher to provide both confusion and diffusion to the ciphertext. The encryption and the decryption key is the way in which the letters are arranged into the Polybius square. The encryption algorithm works as follows:

- Encode the plaintext letters as a simple Polybius algorithm, taking the corresponding row and column numbers. Write the row number above the column number and the column number below the row number obtaining so 2 rows of numbers.
- Like a transposition cipher read the text line by line in pairs of two numbers.
- Decode each pair using the Polybius square.
- The new obtained text will be the ciphertext.

Let's do a practical example. Suppose that the starting Polybious Square is the following:

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

We want to encrypt the phrase "crypto is fun". Let's proceed applying the Polybius algorithm producing the sequence 51 13 15 54 52 44 34 41 21 23 55. Now arrange numbers according to the Bifid rule:

C	R	Y	P	T	O	I	S	F	U	N
5	1	1	5	5	4	3	4	2	2	5
1	3	5	4	2	4	4	1	1	3	5

It is now time to read the numbers line by line obtaining 51 15 54 34 22 51 35 42 44 11 35. Now decrypt these numbers with the Polybious square to obtain the final ciphertext: CYPILCBVOEB. To decrypt the ciphertext another algorithm is provided:

- Use the Polybius square to encode the ciphertext into a sequence of numbers.
- Split the ciphertext into 2 parts putting the result into two rows.
- Read the number pairs column by column from left to right.
- Decode each pair using the Polybius square.

Challenge 15.1 — It is strange.
Altvcqndve is tpptchv aoa igymc cf llqrdzq ummusidrfmdep fq r odre fphx rhhrogalscq ntoordryhots oo fts xqfeixu ddkk labd wx siwqdrpntlon aolfqtwq, hntvndtfs caa ocacfuidn ■

Solution 15.1 — It is strange.
hint - classical polybious square
Delastelle is unusual for being an amateur cryptographer at a time when significant contributions to the subiect were made by professional soldiers, diplomats and academics

15.3 Trifid Cipher (1902)

In the book "Traité Élémentaire de Cryptographie", Delastelle proposed another cipher called Trifid Cipher. The principle is similar to the Bifid cipher invented 7 years before with some small variations that makes the system stronger and the ciphertext harder to decrypt. The first modification lies in how the plaintext is encoded. If in the Bifid Cipher Dlastelle used the Polybius Square to generate numeric bigrams starting from single plaintext letters, here he proposes to use trigrams with the following note on how to generate them:

"In order to split letters into three parts, it is necessary to represent them by a group of three signs or numbers. Knowing that n objects, combined in trigrams in all possible ways, give n x n x n = n^3, we recognize that three is the only value for n; two would only give 2^3 = 8 trigrams, while four would give 4^3 = 64, but three give 3^3 = 27."

The trigram generation is so following a 3x3x3 cube structure. Delastelle generates 3 small 3x3 polybious square, representing the 3 layers of the cube:

LAYER 1

	1	2	3
1	A	B	C
2	D	E	F
3	G	H	I

LAYER 2

	1	2	3
1	J	K	L
2	M	N	O
3	P	Q	R

LAYER 3

	1	2	3
1	S	T	U
2	V	W	X
3	Y	Z	+

Each letter is so replaced with trigram compost by the layer, the row and the column in which the letter belongs. As in Bifid Cipher the generated n-grams have to be written vertically and then divided in groups of some chosen fixed length (the last group could be smaller). The trigrams are read again (as done for the Bifid) row by row, following the generated groups (once one group has been read row by row, pass to the next one). Using the original substitution cube decode each trigram to obtain the ciphertext. Delastelle described the procedure as:

"We start by writing vertically under each letter, the numerical trigram that corresponds to it in the enciphering alphabet: then proceeding horizontally as if the numbers were written on a single line, we take groups of three numbers, look them up in the deciphering alphabet, and write the result under each column."

Notice that the key of the system is composed by 2 parts: the arrangement of the letter in the cube, and the length of each group of trigrams. Let's do an example: suppose we want to encrypt the plaintext "bifid extension" using the following cube and with group size of 5 elements:

LAYER 1

	1	2	3
1	P	Y	D
2	H	W	U
3	A	Q	J

LAYER 2

	1	2	3
1	N	B	R
2	G	I	V
3	C	M	X

LAYER 3

	1	2	3
1	L	Z	E
2	S	K	+
3	T	O	F

It is easy to see how "bifid extension" can be mapped to the trigram sequence 212 222 333 222 113 313 233 331 233 211 321 222 332 211:

B	I	F	I	D	E	X	T	E	N	S	I	O	N
2	2	3	2	1	3	2	3	2	2	3	2	3	2
1	2	3	2	1	1	3	3	3	1	2	2	3	1
2	2	3	2	3	3	3	1	3	1	1	2	2	1

proceed grouping the trigrams 5 by 5:

B	I	F	I	D
2	2	3	2	1
1	2	3	2	1
2	2	3	2	3

E	X	T	E	N
3	2	3	2	2
1	3	3	3	1
3	3	1	3	1

S	I	O	N
3	2	3	2
2	2	3	1
1	2	2	1

Now read back the trigrams, group by group, row by row, obtaining 223 211 232 122 323 323 221 333 133 131 323 222 311 221. Use the key cube to generate the final ciphertext "VNMW++GFJA+ILG"

Challenge 15.2 — I am so confused.
EIBMIXUOJEZORQRXPTXHYDLSXSQHCKHZBRPTPCMZZFPIIDMWNFSWUQQOLTB
YNLDYGGXIN+JP+PMOOYYLCES+NIFESHYLIESZGIQPEKNIDUWYILCGSLUBYX
RAIFMG+UOJEZORQRXYTUHZCNQ+QPYF+FIHVWIJSZJRLJLWDZBUFSZURQOYU
SCAKTCVYHRU+SWCRBOGFIZEINXISYZUIGTGCSMSMYRIXIZWIUFAHTFYIJSV
RIICFHUWRDFIXFEW+LX ■

Solution 15.2 — I am so confused.
hint 1 - no scrambled cube
hint 2 - group size is a prime number < 10
Confusion refers to making the relationship between the key and the ciphertext as complex and as involved as possible. Diffusion refers to the property that redundancy in the statistics of the plaintext is "dissipated" in the statistics of the ciphertext.

15.4 2-Square Cipher (1902)

The 2-square cipher, also known as double Playfair cipher, is a polygraphic substitution cipher that works similar to the Playfair one, with the biggest difference that 2 keyed Polybious squares, with 2 encryption/decryption keys, are involved. The grids can be placed close each others in horizontal or vertical way. The first letter of the bigram must be find into the first grid, while the second letter into the second grid. The message is first split into bigrams, appending a 'X' in the case of odd message length, and each bigram is then encrypted according to the following rules:

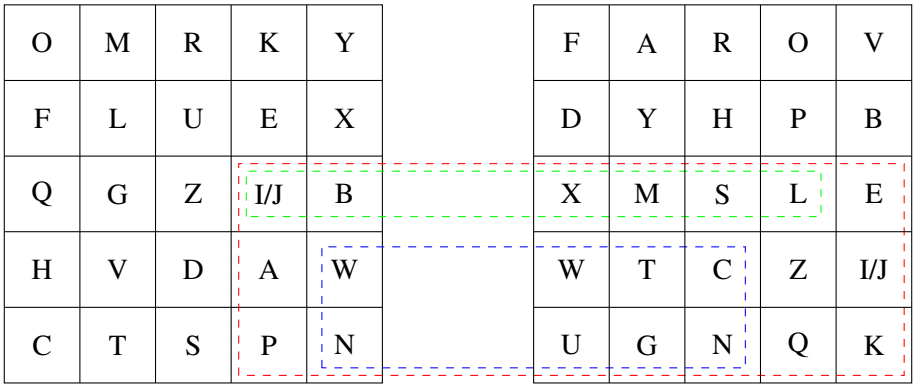
- If both of letters composing the bigram lays on the sam row (or column), do nothing;
- Otherwise build a rectangle between the two Polybious squares using the letters of the bigram as vertexes. Replace each letter with the other vertex letter on the same Polybious square.

Suppose we want to encrypt the word "PENCIL" using the following grids:

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

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

First split the word into bigrams obtaining "PE", "NC" and "IL". The first letter of the bigram have to be found in the first grid, the second letter in the second grid so that "PE" will form the red rectangle, "NC" the blue one and "IL" the green one:



It's easy to notice that the "PE" and the "NC" bigrams form a rectangle. To encrypt them we will take the letters composing the remaining two vertexes, respectively "IK" and "WN". On the other side "IL" doesn't form a rectangle because the letters composing the bigram are on the same line. In this case nothing happen and "IL" remains itself. The ciphertext will so be "IKWNIL". Since "IL" is mapped as itself it is called to be a transparent encryption. Transparent encryptions covers about the 20% of all possible bigrams making them a huge weakness of the two-square cryptosystem.

15.5 4-Square Cipher (1902)

15.6 ADFGX Cipher (1918)

During the first world war, the German lieutenant Fritz Nebel combined a variant of the polybius square with a columnar transposition cipher creating the "Geheimschrift der Funker 1918" or, shorter, the "GedeFu 18". The first person to break it was the French lieutenant Georges Painvin who gives to it the name of the only symbols that appears in the ciphertext: ADFGX. It is not sure but it seems that thanks to the work of Painvin, France was able to discover where the Germans intended to attack during the Spring Offensive and this helps for the Allies victory. The main problem in this assumption is temporal:

"Telegrams in ADFGX appeared for the first time on 5 March, and the German attack started on 21 March. When Painvin presented his first solution of the code on 5 April, the German offensive had already petered out."

Like the Trifid cipher, the ADFGX has 2 different keys. The first one is how the polybius square is arranged. The particularity of the polybius square used by Nebel is that instead of numbers from 1 to 5, the square has letters A, D, F, G, X.

The second key is the string of text used to scramble the columns of the table generated by applying the polybius encrytion method to the plaintext, as in a simple Columnar Transposition Cipher described above. The full encryption algorithm works as follows:

- Arrange the alphabet in a polybius table in a secret way;
- For columns and rows identification, replace number 1 with letter A, number 2 with letter D, number 3 with letter F, number 4 with letter G and number 5 with letter X;
- Encode the plaintext using the arranged square as a classical polybius encoding system;
- Choose a secret key and arrange the encoded plaintext into a table with *lenght(key)* columns, row by row strarting from the upper left most cell;
- Scramble columns according to a classical Transposition Cipher;
- read the table column by column to produce the final ciphertext.

Let's do an example. Suppose we want to encrypt the plaintext "happy news war is over" using the key "blocks" and the following modified Polybius square:

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

We can start encoding the plaintext using the table, producing "XF DG XG XG AX XX AA GX GA GX DG AF FG GA GG GD AA AF". Now we have to arrange a table composed by length("blocks") = 6 columns, row by row. The table is then scrambled according to a classical Columar Transposition Cipher followings the alphabet positions of the letters composing the key:

B	L	O	C	K	S
1	4	5	2	3	6
X	F	D	G	X	G
X	G	A	X	X	X
A	A	G	X	G	A
G	X	D	G	A	F
F	G	G	A	G	G
G	D	A	A	A	F

B	C	K	L	O	S
1	2	3	4	5	6
X	G	X	F	D	G
X	X	X	G	A	X
A	X	G	A	G	A
G	G	A	X	D	F
F	A	G	G	G	G
G	A	A	D	A	F

Finally the table is read column by column starting from the left most one, obtaining the final ciphertext: "XXAGFG GXXGAA XXGAGA FGAXGD DAGDGA GXAFGF". Decrypting the ciphertext is very simple, if you have doubts on how to do it, follow instructions in Columnar Transposition and Polybius Square sections.

15.7 ADFGVX Cipher (1918)

One of the biggest limitations of ADFGX is that it was not able to encode numbers. A variant of ADFGX is the ADFGVX cipher that extends the polybius square by adding one more column and one more row in order to be able to encode 36 different sumbols (26 alphabet letters + 10 digits). The encryption and decryption algorithms are the same of ADFGVX. An example of ADFGVX square is the following:

	A	D	F	G	V	X
A	E	3	Q	C	Y	J
D	P	U	0	K	X	A
F	9	O	7	I	T	2
G	D	4	N	F	V	Z
V	L	1	B	R	W	6
X	M	8	G	S	5	H

Only in 1966, almost fifty years later, Fritz Nebel learned that his system had been broken during the First World War, and he said that he had originally proposed a double column transposition, but this had been rejected in discussions by his superiors. At that time, for practical reasons, they opted for simple column transposition. In 1968, Nebel and Painvin met in person and Nebel expressed his feelings by saying that the enemies of yesterday meet as the friends of today.

It can be noticed that the security of the polybious square is really bad. In order to break it it is sufficient to perform a simple frequency analysis on ciphertext bigrams. Solving the ADFGX or ADFGVX ciphers involved only the problem of solving the columnar transposition cipher and then apply the frequency analysis on the unscrambled text. The true story:

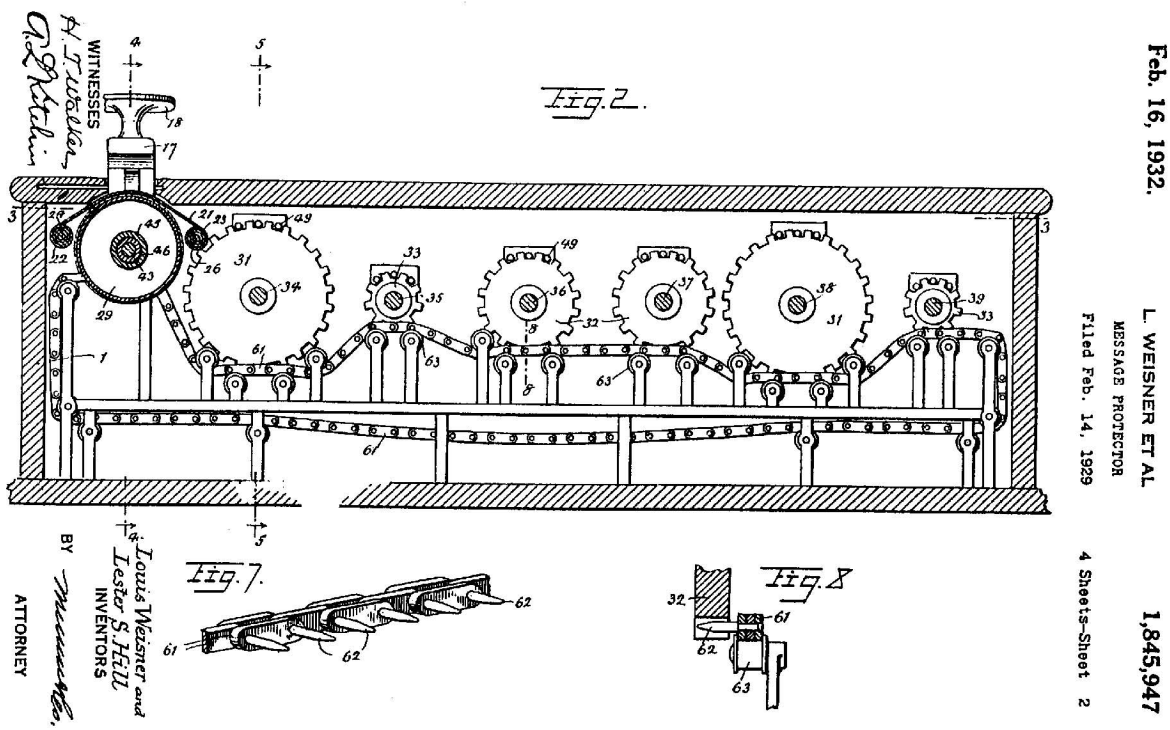
On June 1, 1918, the French radio station on the top of the Eiffel Tower caught for the first time a German radio message containing not only the letters A, D, F, G and X, but also the letter V. The radio message came from the Region Remaugies, north of Compiègne, and read:

FGAXA XAXFF FAFVA AVDFA GAXFX FAFAG DXGGX AGXFD XGAGX GAXGX AGXVF VXXAG XDDAX GGAAF DGGAF FXGGX XDFAX GXAXV AGXGG DFAGG GXVAX VFXGV FFGGA XDGAX FDVGG A

Painvin recognized this and correctly concluded that the Germans had extended the Polybius square from 5 x 5 to 6 x 6, meaning they could now encode 36 characters instead of just 25. He also correctly guessed that the 26 letters of the alphabet plus the 10 numbers (0 to 9) were used, and based his cryptanalysis on this assumption. After intensive work over 26 hours to physical exhaustion, the decipherment succeeded on June 2, 1918. The clear text was "Munitionierung beschleunigen Punkt Soweit nicht eingesehen auch bei Tag". This information was immediately forwarded to the French headquarters around Marshal Foch and convinced him that the Germans were planning a massive attack in the front section at Compiegne. He concentrated his last reserve troops around this city, which meant that the German attack that was actually taking place shortly thereafter could be repelled. On the French side, the German radio message is since then called Le Radiogramme de la Victoire (The "radio message of victory")

15.8 Hill's Cipher (1929)

The Hill's Cipher takes its name from its inventor Lester s. Hill that in 1929 developed an encryption system based on matrices. Each letter of the English alphabet is represented, with the usual convention, with numbers from 0 to 25. To encrypt the message, each block of n letters is transformed into a vector and then is multiplied by an invertible nxn matrix that represents the encryption key. To decrypt the message the same process is applied, but this time the vectors coming from the ciphertext are multiplied by the inverse of the key matrix. All operations are usually performed modulus 26 to maintain the same input notation.



Suppose we want to encrypt the word "cryptography" using the key "encodings". First we have to define the key matrix that in our case will be a 3x3 one ("encodings" has 9 letters and can be arranged into a 3x3 schema):

$$\begin{bmatrix} e & n & c \\ o & d & i \\ n & g & s \end{bmatrix} = \begin{bmatrix} 4 & 13 & 2 \\ 14 & 3 & 8 \\ 13 & 6 & 18 \end{bmatrix} \tag{15.1}$$

In the same way, the plaintext "cryptography" can be arranged into 4 vectors of size 3:

$$cryptogrphy = \begin{bmatrix} c \\ r \\ y \end{bmatrix} \begin{bmatrix} p \\ t \\ o \end{bmatrix} \begin{bmatrix} g \\ r \\ a \end{bmatrix} \begin{bmatrix} p \\ h \\ y \end{bmatrix} = \begin{bmatrix} 2 \\ 17 \\ 24 \end{bmatrix} \begin{bmatrix} 15 \\ 19 \\ 14 \end{bmatrix} \begin{bmatrix} 6 \\ 17 \\ 0 \end{bmatrix} \begin{bmatrix} 15 \\ 7 \\ 24 \end{bmatrix} \tag{15.2}$$

Now we can start multiply each vector with the encryption key:

$$\begin{bmatrix} 4 & 13 & 2 \\ 14 & 3 & 8 \\ 13 & 6 & 18 \end{bmatrix} * \begin{bmatrix} 2 \\ 17 \\ 24 \end{bmatrix} (mod \quad 26) = \begin{bmatrix} 13 \\ 23 \\ 1 \end{bmatrix} = \begin{bmatrix} n \\ x \\ b \end{bmatrix} \tag{15.3}$$

$$\begin{bmatrix} 4 & 13 & 2 \\ 14 & 3 & 8 \\ 13 & 6 & 18 \end{bmatrix} * \begin{bmatrix} 15 \\ 19 \\ 14 \end{bmatrix} (mod \quad 26) = \begin{bmatrix} 23 \\ 15 \\ 15 \end{bmatrix} = \begin{bmatrix} x \\ p \\ p \end{bmatrix} \tag{15.4}$$

$$\begin{bmatrix} 4 & 13 & 2 \\ 14 & 3 & 8 \\ 13 & 6 & 18 \end{bmatrix} * \begin{bmatrix} 6 \\ 17 \\ 0 \end{bmatrix} (mod \quad 26) = \begin{bmatrix} 11 \\ 5 \\ 24 \end{bmatrix} = \begin{bmatrix} l \\ f \\ y \end{bmatrix} \tag{15.5}$$

$$\begin{bmatrix} 4 & 13 & 2 \\ 14 & 3 & 8 \\ 13 & 6 & 18 \end{bmatrix} * \begin{bmatrix} 15 \\ 7 \\ 24 \end{bmatrix} (mod \quad 26) = \begin{bmatrix} 17 \\ 7 \\ 24 \end{bmatrix} = \begin{bmatrix} r \\ h \\ y \end{bmatrix} \tag{15.6}$$

So the term "cryptography" has been encrypted as "xnbxpplfyrhy". But now we have a huge problem. There is no way to recover the original message because the key matrix used in not invertible. A n-by-n matrix A is invertible if and only id there exists a n-by-n matrix B such that $AB = BA = I_n$ where I_n denotes the n-by-n identity matrix. This, sadly, is not our case. For our next example suppose that an unknown word has been encrypted using key "beer" obtaining the ciphertext "fics". This time the key matrix is invertible modulus 26:

$$\begin{bmatrix} b & e \\ e & r \end{bmatrix} = \begin{bmatrix} 1 & 4 \\ 4 & 17 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 4 \\ 4 & 17 \end{bmatrix}^{-1} = \begin{bmatrix} 17 & -4 \\ -4 & 1 \end{bmatrix} (mod \quad 26) = \begin{bmatrix} 17 & 22 \\ 22 & 1 \end{bmatrix} \tag{15.7}$$

We are now ready to decrypt the cipher and retrieve the original plaintext:

$$\begin{bmatrix} 17 & 22 \\ 22 & 1 \end{bmatrix} * \begin{bmatrix} f \\ i \end{bmatrix} \rightarrow \begin{bmatrix} 17 & 22 \\ 22 & 1 \end{bmatrix} * \begin{bmatrix} 5 \\ 8 \end{bmatrix} (mod \quad 26) = \begin{bmatrix} 1 \\ 14 \end{bmatrix} = \begin{bmatrix} b \\ o \end{bmatrix} \tag{15.8}$$

$$\begin{bmatrix} 17 & 22 \\ 22 & 1 \end{bmatrix} * \begin{bmatrix} c \\ s \end{bmatrix} \rightarrow \begin{bmatrix} 17 & 22 \\ 22 & 1 \end{bmatrix} * \begin{bmatrix} 2 \\ 18 \end{bmatrix} (mod \quad 26) = \begin{bmatrix} 14 \\ 10 \end{bmatrix} = \begin{bmatrix} o \\ k \end{bmatrix} \tag{15.9}$$

So the original message was "book". Well, book and beer, not really a good association...

16. Modern Cryptography

Algoritmi a chiave pubblica cosa è? idea...

16.1 Mathematical Background

To better face the next section of the book a little mathematical background is required

16.1.1 Divisors of a number

The divisors of and integer number different from 0 are all those integers which divide the number without remainder. For example the divisors of 15 are 1,3,5 and 15 itself. A simple code to compute the list of divisors of a number is the following:

```
static List<Integer> divisors(int n) {
   List<Integer> ret = new ArrayList<>();
   for (int i = 1; i <= n; i++) {
      if (n % i == 0) {
         ret.add(i);
      }
   }
   return ret;
}
```

16.1.2 Prime Numbers

Prime numbers are very special numbers defined as those numbers which are divisible only by 1 and themselves. In other words, given a prime number p, the list of its divisors will be {1,p}.

16.1.3 Greatest Common Divisor (GCD)

The greatest common divisor (or GCD) of two or more non-zero integers is defined as the largest positive integer that divides each integers without remainder. For example, the GCD of 16 and

20 is 4, the GCD of 9 and 12 is 3 and the GCD of 15 and 25 is 5. A very simple way to compute the GCD of two numbers is to find all the divisors of both numbers and take the biggest one in common. To compute GCD(16,20) we first compute divisors of 16:{1,2,4,8,16} and then divisors of 20:{1,2,4,5,10} discovering that the biggest divisor in common is 4.

R Since prime numbers are only divisible by 1 and by themselves is easy to notice that, given p and q two prime numbers, then GCD(p,q)=1.

Another very easy and fast way to calculate GCD is the Euclidean algorithm. The algorithm is based on the observation that the gcd of two numbers also divides their difference. This means that given 2 integers a and b, with $a > b$, the following equations are valid:

$$a \mod gcd(a,b) = 0 \tag{16.1}$$
$$b \mod gcd(a,b) = 0 \tag{16.2}$$
$$(a-b) \mod gcd(a,b) = 0 \tag{16.3}$$

With this information it is possible to calculate the GCD of two positive integers recursively in the following way: $gcd(a,b) = gcd(a-b,b)$, with $a > b$ and $gcd(a,b) = gcd(a,b-a)$ with $b > a$. The algorithm stops eventually when it forces to calculate $gcd(x,x) = x$. The following is the script:

```
static int gcd(int a, int b) {
   return a == b ? a : a > b ? gcd(a - b, b) : gcd(a, b - a);
}
```

Unfortunately the above algorithm is not universal, in fact it does not work when a or b is less than 0. A generalization of the Euclidean algorithms works as follows: let's imagine to compute gcd(24,18). We start dividing 24 by 16 to get a quotient of 1 and a reminder of 6. We proceed then dividing 16 by 6 obtaining 2 as quotient and 4 as remainder. Last we divide 6 by 2 obtaining a quotient of 3 and a remainder of 0. 4 will be the gcd(24,16). The algorithms stops when the remainder reaches 0. Formally the algorithm can be described as:

$$gcd(a,0) = a \tag{16.4}$$
$$gcd(a,b) = gcd(b, a \mod b) \tag{16.5}$$

where

$$a \mod b = a - b\lfloor \frac{a}{b} \rfloor \tag{16.6}$$

16.1.4 Coprime Numbers

We say that for any two integers a,b, if gcd(a,b) = 1 then a and b are coprime integers. If a and b are prime, they are also coprime. If a is prime and b < a then a and b are coprime. Think about the case for a prime and b > a, why are these not necessarily coprime?

16.2 RSA (1977)

RSA acronym derives from the first letter of surname of its creators: Ron Rivest, Adi Shamir and Leonard Adleman. It is one of the first public-key cryptosystem ever developed and it was widely used for secure data trasmission. Generally speaking is a very good algorithm but with some flaws in parameter settings. Choosing a bunch of bad parameters makes RSA not secure at all and can be

attacked in various ways as we will see later.
History says that in 1973 Clifford Cooks, while working at United Kingdom Government Communications Headquarters (GCHQ), invented a RSA equivalent cryptosystem. Because the idea was classified information, the fact remain hidden for 24 years till the work has been unclassified. Anyway the GCHQ never found a practical use of this cryptosystem. 4 years later a very similar algorithm has been indepentely invented by Rivest, Shamir and Adleman which were also able to find a practical use for it. It has never been found any evidence of information leak in GCHQ and so RSA is to be consider authentic.
But now let's focus on how RSA works. The encryption and the decryption formulas are the following:

$$c \equiv m^e \pmod{n} \tag{16.7}$$

$$m \equiv c^d \pmod{n} \tag{16.8}$$

Where m is the message to encrypt, e is the sender's public key, d is the receiver's private key and c is the encrypted message. The strength of RSA lies in choosing the right keys. The main idea is that it is very easy to compute $m^e \pmod{n}$ even with very large values of n and e, but at the same time is very difficult to infer d, reversing the encryption formula, even if the original message m is known due to the nature of n. N is so the most important parameter for RSA encryption but also e plays a fundamental role. What follows is the key generation algorithm:

- Choose two distinct prime numbers p and q
- Calculate $n = p * q$
- Compute $\lambda(n) = lcm(\phi(p), \phi(q)) = lcm(p-1, q-1)$
- Choose e such that $1 < e < \lambda(n)$ and $gcd(e, \lambda(n)) = 1$
- Calculate $d \equiv e^{-1} \pmod{\lambda(n)}$

With:

- $\lambda(n)$ is the Charmichael function defined as the smallest positive integer m such that $a^m \equiv 1 \pmod{n}$ for every integer between 1 and n that is coprime to n.
- $\phi(n)$ is the Euler's totient function defined as the number of positive integers up to n that are relatively prime to n.

The next table will summarize the Charmichael and Euler's totient values of first 20 integers:

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$\lambda(n)$	1	1	2	2	4	2	6	2	6	4	10	2	12	6	4	4	16	6	18	4
$\phi(n)$	1	1	2	2	4	2	6	4	6	4	10	4	12	6	8	8	16	6	18	8

Let's do an example.

- We choose $p = 1009$ and $q = 797$ as our basic different prime numbers.
- Then we compute $n = p * q = 1009 * 797 = 804173$
- We calculate $\lambda(n) = lcm(p-1, q-1) = lcm(1008, 796) = 200592$
- We choose $e = 29$. Thus $gcd(29, 200592) = 1$
- We calculate $d \equiv 29^{-1} (mod 200592) \rightarrow d = 608693$

Now we are ready to encrypt a message, for example the string "IT". First thing to do is to transform the string into numeric values. It can be done by transforming first to hex notation and then convert to dec base: "IT" $\rightarrow$ 49 54 $\rightarrow$ 0x4954 $\rightarrow$ 18772. Now we can encrypt the message using the public key e: $c \equiv 18722^{29} (mod 200592) = 382506$. To decrypt we can use the private key d: $m \equiv 382506^{608693} (mod 200592) = 18722$

16.2.1 Breaking RSA

In its semplicity RSA is, in general, a good encryption algorithm, but if used superficially it can present some flaws and vulnerabilities. As we saw before n is a crucial parameter for guarantee high level security. If n can be easily factored, then the attacker have enough data to break the cipher. It is so important that p and q are big enough to guarantee that n cannot be factored easily. Online factorization DBs are quite common nowadays ad factordb.com is one of the biggest in the web. Looking for example for the number 435897296798372609834276094237603948760923737 it says that its factors are 4643 and 938826829201750182714357299671772450486 59 that are both primes.

16.2.2 Fermat's Attack

Another common mistake is to use p and q too close each other. In this case, even if p and q are very large or even huge, the Fermat's factorization method is able to retrieve them easily. The theorem is based on the assumption that and odd number can be represented as a difference of two squares:

$$N = a^2 - b^2 \tag{16.9}$$

And it is well known that this difference is easily factorable as

$$a^2 - b^2 = (a+b) * (a-b) \tag{16.10}$$

Indeed if $n = c * d$ is a factorization of n, then $c * d$ can be rewritten as

$$N = \left(\frac{c+d}{2}\right)^2 - \left(\frac{c-d}{2}\right)^2 \tag{16.11}$$

$$a = \frac{c+d}{2} \tag{16.12}$$

$$b = \frac{c-d}{2} \tag{16.13}$$

Knowing this the Fermat's algorithm works as follows:

- Start with a = ceil(sqrt(N))
- compute $b_2 = a^2 - N$
- until b_2 is not a perfect square do:
  $a = a + 1$
  b_2 = sqrt($a^2 - N$)

When the algorithm ends we know that $a = a$ and $b = sqrt(b_2)$. Then we only need to solve the system provided by equations (16.12) and (16.13) to find d and c:

$$\begin{cases} a = \frac{c+d}{2} \\ b = \frac{c-d}{2} \end{cases} \qquad \begin{cases} c = a+b \\ d = a-b \end{cases} \tag{16.14}$$

R Let's do an example. Imagine to have $N = 2151491$. Running the Fermat's Algorithm step by step we obtain:

step:	1	2	3	4
a	1467	1468	1469	1470
b_2	598	3533	6470	9409
b	24.45	59.43	80.43	97

At step 4 b_2 is a perfect square and the algorithm ends. Let's proceed solving the equation system, knowing that $a = 1470$ and $b = 97$:

$$\begin{cases} c = 1470 + 97 \\ d = 1470 - 97 \end{cases} \qquad \begin{cases} c = 1567 \\ d = 1373 \end{cases} \tag{16.15}$$

In this way we easily found $N = c * d \rightarrow 2151491 = 1567 * 1373$

16.2.3 Common modulus

Common modulus attack can be done when 2 messages encrypted with the same modulus N are intercepted by the attacker. The attacker needs to know also the public keys used to encrypt the messages, but since the keys should be public, this is a free step. To understand how the attack works let's recap how the intercepted messages are encrypted:

$$C_A = M_A^{e_A} \tag{16.16}$$
$$C_B = M_B^{e_B} \tag{16.17}$$

In this case, knowing that e_A and e_B must be primes, it is a pain fact. To explain why all this panic, let's introduce the Bézout's identity:

"Let a and b be integers with greatest common divisor d. Then, there exist integers x and y such that ax + by = d. More generally, the integers of the form ax + by are exactly the multiples of d."

With e_A and e_B primes it is always true that $gcd(e_A, e_B) = 1$, and for the Bézout's identity follows that there must exist u and v such that:

$$e_A * u + e_B * v = gcd(e_A, e_B) = 1 \tag{16.18}$$

To solve this equation, the Euclidean algorithm to compute $gcd(a,b)$ comes in help. Given two integer numbers a and b, the algorithms works as following:

- $a = q_0 * b + r_0$
- $b = q_1 * r_0 + r_1$
- while r_k is not zero do:
  $r_{k-2} = q_k * r_{k-1} + r_k$

In this way we obtain a sequence like this:

$$a = q_0 * b + r_0 \tag{16.19}$$
$$b = q_1 * r_0 + r_1 \tag{16.20}$$
$$r_0 = q_2 * r_1 + r_2 \tag{16.21}$$
$$\ldots \tag{16.22}$$
$$r_{n-2} = q_n * r_{n-1} + r_n \tag{16.23}$$
$$r_{n-1} = q_{n+1} * r_n + 0 \tag{16.24}$$

Euclidean's algorithm is used to find gcd but we use it in another way. Once computed gcd we can rewrite the results of the single steps as follows:

$$r_n = r_{n-2} - q_n * r_{n-1} \tag{16.25}$$
$$\ldots \tag{16.26}$$
$$r_2 = r_0 - q_2 * r_1 \tag{16.27}$$
$$r_1 = b - q_1 * r_0 \tag{16.28}$$
$$r_0 = a - q_0 * b \tag{16.29}$$

Substituting terms into an unique formula, will solve the equation, providing the terms u and v of the Bézout's identity.

Suppose to have $e_A = 71$ and $e_B = 53$ and we want to solve the Bézout's identity $71 * u + 53 * v = 1$. We proceed computing $gcd(71,53)$ as follows using the Euclidean's algorithm:
$71 = 1 * 53 + 18$
$53 = 2 * 18 + 17$
$18 = 1 * 17 + 1$
$17 = 17 * 1 + 0$
Now we can rearrange the steps as follows:
(1) $1 = 18 - 17 * 1$
(2) $17 = 53 - 18 * 2$
(3) $18 = 71 - 53 * 1$
We can now start the substitution process. Putting (2) in (1) we obtain $1 = 18 - 1 * (53 - 18 * 2)$.
Putting also (3) in this new equation it comes that $1 = (71 - 53 * 1) - 1 * (53 - 2 * (71 - 53 * 1))$.
Simplifying this last equation we get

$$71 - 53 - 1 * (53 - 2 * 71 + 2 * 53) = 1 \tag{16.30}$$
$$71 - 53 - 53 + 2 * 71 - 2 * 53 = 1 \tag{16.31}$$
$$3 * 71 - 4 * 53 = 1 \tag{16.32}$$

This solves the Bèzout's identity with solutions $u = 3$ and $v = -4$.

Till now only a big bunch of boring concepts. But how this concepts can represent a problem for RSA? The fact is that (16.16) and (16.17) can be rewritten as:

$$C_A^u = M_A^{e_A^u} = M^{e_A * u} \tag{16.33}$$
$$C_B^v = M_B^{e_B^v} = M^{e_B * v} \tag{16.34}$$

Now, if we multiply both messages we obtain the current flaw:

$$M^{e_A * u} * M^{e_B * v} = M^{e_A * u + e_B * v} = M^1 = M \tag{16.35}$$

Since $C = M^e$, to decrypt RSA is sufficient to calculate

$$M = C_A^u * C_B^v \tag{16.36}$$

usually v is negative and so the inverse of $C_B mod N$ have to be computed:

$$M = C_A^u * C_{B,inv}^{-v} \tag{16.37}$$

16.2.4 Small Public Exponent

One of the worst practice in RSA is using small public exponent e. Even if N has been computed choosing strong primes p and q, a small exponent will make the whole encryption insecure. First, what is the main purpose of e? Well, to make the message M bigger enough so that modulus N makes sense. Thus if $M^e < N$, then the encrypted message can be simply decoded using:

$$M = \sqrt[e]{C} \tag{16.38}$$

Even if this are extremely rare cases in which the message is very short and the public key is very low, it is a good situation to take care

R A Free decryption is obtained when trivially $e = 1$. In this case the have that $C = M^1 (mod \quad N)$ and so $C = M$ and no decryption is needed at all (just convert number to ascii)

16.2.5 Hastad's Broadcast Attack

There is actually another problem in using small public exponents and it comes when the same message is sent to, at least, 3 different people. Suppose that the message is long enough that the previous technique cannot be applied, what is the problem in sending the same message to different people using a small exponent? Start showing the anatomy of the attack: Suppose that we were able to capture at least e ciphertexts corresponding to the same plaintext, then the following system of equations is true:

$$\begin{cases} C_1 \equiv M_1^e (mod \quad N_1) \\ C_2 \equiv M_2^e (mod \quad N_2) \\ xxx \\ C_e \equiv M_e^e (mod \quad N_e) \end{cases} \tag{16.39}$$

Solving this system is trivial because the chinese remainder theorem comes in help, telling us that for (16.39) exists a single module $N = \prod_{i=1}^{e} N_i$ in the case in which $gcd(N_i, N_j) = 1 \bigvee i \neq j$. We can safely assume the this is true because otherwise we were able to find factors p and q by simply calculate $gcd(N_i, N_j)$. Now we are able to rewrite the previous system in:

$$C \equiv M^e (mod \quad N_1, N_2, ..., N_e) \tag{16.40}$$

However, since for short messages $M < N_i \bigvee i$, then also $M^e < N_1 * N_2 * ... * N_e$ and (16.52) can be rewritten as:

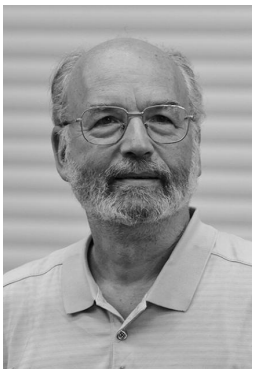
$$C \equiv M^e \tag{16.41}$$

That is simply solvable calculating $M = \sqrt[e]{C}$

16.2.6 Wiener's Attack

16.3 Shamir's Secret Sharing (1979)

In many cryptographic or organizational settings, it is undesirable for a single individual to possess complete control over a sensitive piece of information such as an encryption key, a password, or access credentials. Instead, we may wish to divide the secret into multiple parts (called *shares*) so that cooperation among several participants is required to reconstruct it.



Shamir's Secret Sharing (SSS) scheme, proposed by Adi Shamir in 1979, achieves this goal elegantly and securely by relying on the fundamental property that *a polynomial of degree $(t-1)$ is uniquely determined by t distinct points*. The scheme is *information-theoretically secure*, meaning that any coalition of fewer than t participants learns absolutely nothing about the secret, even with unlimited computational power.

Definition 16.3.1 — Shamir's (t,n)-Threshold Scheme. Let p be a prime number, and let $\mathbb{F}_p = \{0,1,\ldots,p-1\}$ denote the finite field of integers modulo p. Given a secret $S \in \mathbb{F}_p$, the dealer wishes to distribute it among n participants $P_1, P_2, \ldots, P_n$ in such a way that:

1. Any subset of at least t participants can collaboratively reconstruct the secret S.
2. Any subset of fewer than t participants obtains no information about S.

This is known as a (t,n)-*threshold secret sharing scheme*.

16.3.1 The Construction

The dealer proceeds as follows:

1. Choose a large prime $p > \max(S,n)$, so all computations can be performed in $\mathbb{F}_p$.
2. Randomly select coefficients $a_1, a_2, \ldots, a_{t-1} \in \mathbb{F}_p$.
3. Define a polynomial of degree $(t-1)$:
$$f(x) = S + a_1 x + a_2 x^2 + \cdots + a_{t-1} x^{t-1} \pmod{p}. \tag{16.42}$$
The secret S is embedded as the constant term $f(0) = S$.
4. For $i = 1,2,\ldots,n$, compute:
$$y_i = f(x_i) \pmod{p}, \tag{16.43}$$
where x_i are distinct, nonzero public values (e.g., $x_i = i$).
5. Each participant P_i receives a private share (x_i, y_i).

Each share can be visualized as a point on the polynomial $f(x)$. The secret itself corresponds to the polynomials intercept on the y-axis, i.e., $f(0)$.

16.3.2 Secret Reconstruction

To recover the secret, any subset of t participants combine their shares. Since a polynomial of degree $(t-1)$ is uniquely determined by t points, they can reconstruct $f(x)$ (or equivalently, $f(0)$) using **Lagrange interpolation**.

Given t shares $(x_1,y_1),\ldots,(x_t,y_t)$, the Lagrange basis polynomials are defined as:

$$\lambda_i(x) = \prod_{\substack{1 \le j \le t \\ j \ne i}} \frac{x - x_j}{x_i - x_j}. \tag{16.44}$$

Then the interpolation polynomial is:

$$f(x) = \sum_{i=1}^{t} y_i \lambda_i(x). \tag{16.45}$$

Evaluating at $x = 0$ gives the secret:

$$S = f(0) = \sum_{i=1}^{t} y_i \prod_{\substack{1 \le j \le t \\ j \ne i}} \frac{x_j}{x_j - x_i} \pmod{p}. \tag{16.46}$$

16.3.3 Worked Example

Consider a $(3,5)$ threshold scheme over $\mathbb{F}_{2089}$ with secret:

$$S = 1234.$$

The dealer selects two random coefficients:

$$a_1 = 166, \quad a_2 = 94.$$

The polynomial is therefore:

$$f(x) = 1234 + 166x + 94x^2 \pmod{2089}.$$

We compute shares for $x = 1,2,3,4,5$:

x_i	$f(x_i) \bmod 2089$	Share
1	1494	$(1,1494)$
2	2002	$(2,2002)$
3	489	$(3,489)$
4	1313	$(4,1313)$
5	236	$(5,236)$

Each participant receives one pair (x_i, y_i). Now time to reconstruct the secret $S = f(0)$ from the three shares $(x_1,y_1) = (1,1494)$, $(x_2,y_2) = (2,1942)$, $(x_3,y_3) = (3,489)$ over the prime field $\mathbb{F}_{2089}$.

The Lagrange interpolation formula evaluated at $x = 0$ is

$$f(0) = \sum_{i=1}^{3} y_i \lambda_i(0), \qquad \lambda_i(0) = \prod_{\substack{1 \le j \le 3 \\ j \neq i}} \frac{x_j}{x_j - x_i} \pmod{2089}.$$

We compute each $\lambda_i(0)$ explicitly.

Index $i = 1$ (using $x_1 = 1$).

$$\lambda_1(0) = \frac{x_2}{x_2 - x_1} \cdot \frac{x_3}{x_3 - x_1} = \frac{2}{2-1} \cdot \frac{3}{3-1} = 2 \cdot \frac{3}{2} = 3 \pmod{2089}.$$

Index $i = 2$ (using $x_2 = 2$).

$$\lambda_2(0) = \frac{x_1}{x_1 - x_2} \cdot \frac{x_3}{x_3 - x_2} = \frac{1}{1-2} \cdot \frac{3}{3-2} = \frac{1}{-1} \cdot 3 = -3 = 2086 \pmod{2089}.$$

Index $i = 3$ (using $x_3 = 3$).

$$\lambda_3(0) = \frac{x_1}{x_1 - x_3} \cdot \frac{x_2}{x_2 - x_3} = \frac{1}{1-3} \cdot \frac{2}{2-3} = \frac{1}{-2} \cdot \frac{2}{-1} = 1 \pmod{2089}.$$

Sanity check.

The Lagrange basis values should satisfy $\lambda_1(0) + \lambda_2(0) + \lambda_3(0) \equiv 1 \pmod{2089}$. Indeed,

$$3 + 2086 + 1 \equiv 2090 \equiv 1 \pmod{2089}.$$

Combine with the known y_i.
Now compute each term $y_i\lambda_i(0)$ modulo 2089.

$$y_1\lambda_1(0) \equiv 1494 \cdot 3 = 4482 \equiv 304 \pmod{2089},$$
$$y_2\lambda_2(0) \equiv 1942 \cdot 2086 = 4051012 \equiv 441 \pmod{2089},$$
$$y_3\lambda_3(0) \equiv 489 \cdot 1 = 489 \pmod{2089}.$$

Sum the contributions:

$$f(0) \equiv 304 + 441 + 489 = 1234 \pmod{2089}.$$

Therefore the reconstructed secret is

$$\boxed{S = f(0) = 1234.}$$

This matches the original secret embedded as the constant term of the dealer's polynomial, so the reconstruction is correct. Any two participants alone, however, would face infinitely many possible polynomials fitting their points, each with a different constant term, yielding no information about S. The security of the scheme is **information-theoretic**: knowing fewer than t shares provides absolutely no advantage in determining S. This is because each missing share corresponds to an unknown degree of freedom (coefficient) in the polynomial, making the system of equations underdetermined. In other words, for fewer than t points, there exist equally many possible polynomials (and therefore equally many possible secrets) consistent with the known shares. Thus, the posterior probability distribution of the secret remains uniform, regardless of the observed shares.

16.4 Rabin Cryptosystem (1979)

In 1979 Michael Oser Rabin, proposed a variant of RSA cryptosystem aiming to flat some flaws of this last technique. The key generation phase has been extremely simplified preserving a good level of security. In fact, Rabin cryptosystem is consider more secure than RSA because has been proven that decrypting a message crypted by Rabin, without knowing private keys, lies in solving the integer "factorization problem" that is supposed to be NP-Complete since till know no algorithm has been provided to solve it in polynomial time. Sadly, even sharing a very similar encryption technique, has never been proved formally that RSA lies in the same decryption problem. Let's focus on Rabin cryptosystem. The encryption and the decryption formulas are the followings:

$$c \equiv m^2 (mod \quad n) \tag{16.47}$$

$$misoneof \begin{cases} M_1 = a*p*m_q + b*q*m_p \\ M_2 = a*p*m_q - b*q*m_p \\ M_3 = -a*p*m_q + b*q*m_p \\ M_4 = -a*p*m_q - b*q*m_p \end{cases} \tag{16.48}$$

What does it mean? Analysing (16.48) it comes that in order to decrypt the message is necessary to solve 4 different equations and one of them will be the correct original plaintext. To better understand this, let's proceed with the key generation algorithm. The algorithm only requires, as RSA, to choose two different prime numbers p and q and then compute $N = p*q$. Thats it: N will be the public encryption key and p, q will be the private decryption key. We can now encrypt any

message using (16.47), but how the decryption algorithm works?
In order to decrypt Rabin, the following problem needs to be solved:

$$m^2 \equiv c(mod \quad N) \tag{16.49}$$

For composite $N = p * q$ there is usually no solution for this problem, but since p and q are primes only one solution exists and we are able to compute the 4 square roots of (16.49). To do so we need to compute first:

$$m_p = \sqrt{c}(mod \quad p) \tag{16.50}$$
$$m_q = \sqrt{c}(mod \quad q) \tag{16.51}$$
$$a * p + b * q = 1 \tag{16.52}$$

In particular (16.52) can be computed using the extended Euclidean algorithm showed in (16.18). Finally the Chinese remainder algorithm is used to compute the final roots showed in (16.48). Which between M_1, M_2, M_3 and M_4 is the correct plaintext is usually unknown at prior.

16.5 Block Ciphers (1981)

Block ciphers differs from stream ciphers by the way in which the plaintext is encrypted. If in stream ciphers each symbol is transformed separately each other, in block ciphers, as the name suggests, blocks of plaintext symbols are processed in order to generate the ciphertext. First specification of block cipher techniques comes from the Federal Information Processing Standards Publication 81 (FIPS81) where ECB, CBC, OFB and CFB are described.

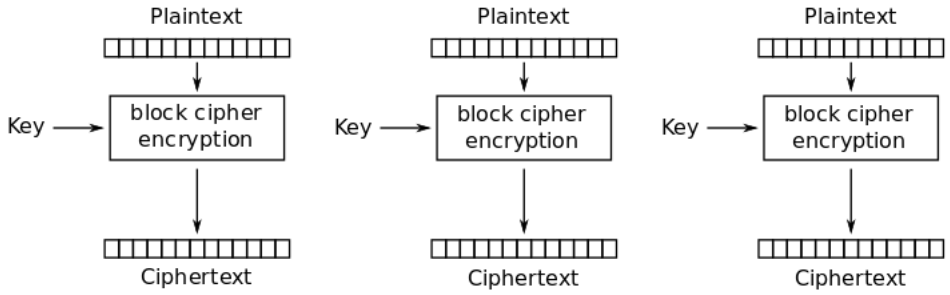
16.5.1 Electronic Codebook (ECB)

ECB is the simplest block based encryption schema, where the plaintext is divided into blocks of fixed length and each block is then encrypted separately using an encryption function. The encryption function takes the block to be encrypt (B_i) and the encryption key (K) as input and produces a block of ciphertext (C_i):

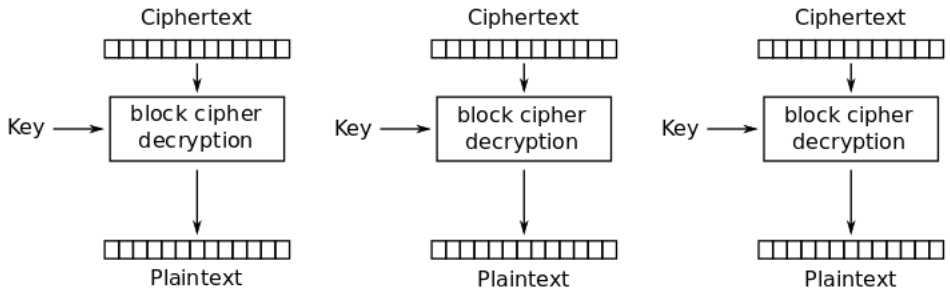
$$C_i = F_{enc}(B_i, K) \tag{16.53}$$

In the same way the decryption function takes in input each cipher block and reconstruct the original plaintext block:

$$B_i = F_{dec}(C_i, K) \tag{16.54}$$



Electronic Codebook (ECB) mode encryption



Electronic Codebook (ECB) mode decryption

Cracking ECB is so simple when known plaintext attack is used. This because each block is encrypted always in the same way, even in in different positions in the original message.

R Suppose that our encryption function simply reverse the block string, and we use ECB with block size 4. Then to encrypt "thisroomissodark" we proceed as following:

plaintext blocks:	this	room	isso	dark
ciphertext blocks	siht	moor	ossi	krad

Obtaining "sihtmoorossikrad". Then using the same function we want to encrypt "thisismy-darkroom":

plaintext blocks:	this	ismy	dark	room
ciphertext blocks	siht	ymsi	krad	moor

That produces the ciphertext "sihtymsikradmoor", which it is easy to see the similarities with respect to the previous encrypted text. This property makes ECB not so secure.

16.6 Stream Ciphers (1987)

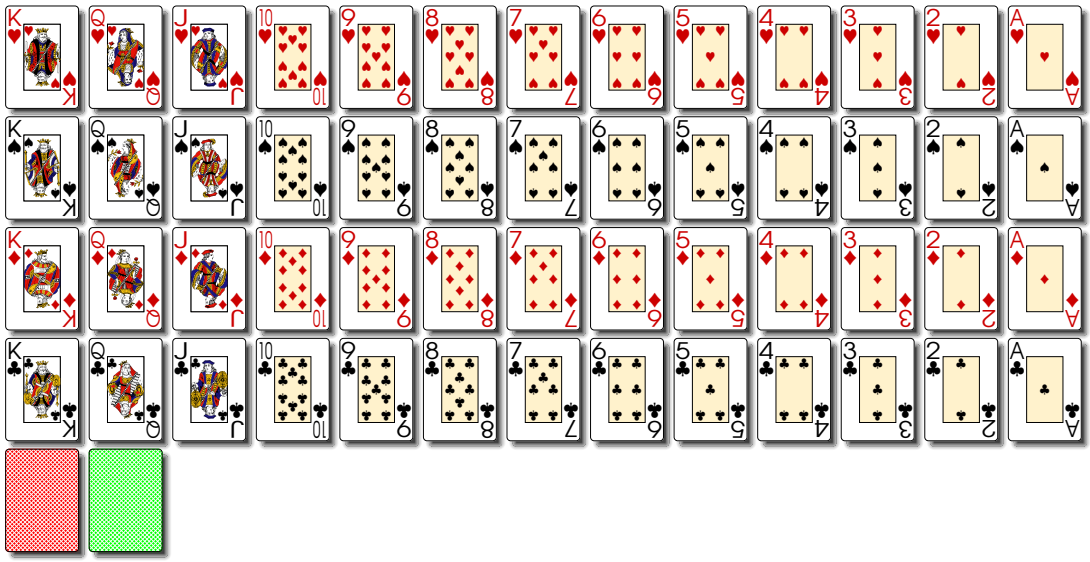
A stream ciphers is an encryption system where each symbol of the plaintext is combined with a different pseudorandom generated symbol that composes the keystream. Keystreams are typically generated serially starting from a seed that serves as the key for the whole cryptosystem. Stream ciphers are inspired by the one-time pad (OTP) invented by Frank Miller in 1882 and patented to Gilbert Vernam in 1919. OTP is a theoretical unbreakable cipher where each plaintext symbol is combined with a TRUE random symbol. The proof of unbreakable has been first given by Vladimir Kotelnikov in 1941, but the document remained classified. In 1945 also Claude Shannon arrived at the same conclusion. The Shannon's results have been unclassified in 1949.

16.6.1 Solitaire/Pontifex Cipher (1999)

The Solitaire Cipher has been invented by Bruce Schneier in 1999 for the Neal Stephenson's novel "Cryptonomicon". The encryption algorithm works as follows:

- All characters that are not letters are removed from the plaintext message.
- All the characters are converted to same case (Upper or Lower no matter)
- Each character is converted in its numerical value (A=1, B=2, ..., Z=26)
- Add to each number the keystream value in the corresponding position, mod 26

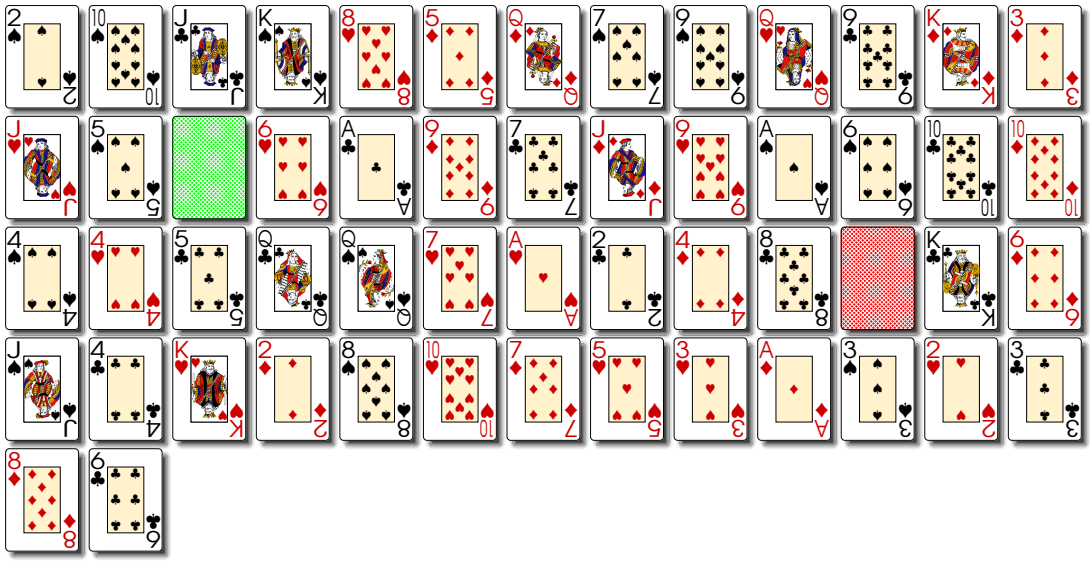
The decryption just work in the opposite way: subtract keystram values from ciphertext and convert numbers to letters. The tricky and fun part is that the every value of the keystream is generated starting from a deck of 52 card plus 2 jolly (the green and red cards), for a total of 54 cards.



The keystream generation works as this:

- Shuffle the deck and put the cards face-up in sequence. This will be the key of the whole cipher. Anyone who knows the same key will be able to reproduce exactly the following steps. Notice that there are 54! or 230843697339241380472092742683027581083278564571807941132288000000000000 possible different ways to arrange a deck of 54 cards. So every time you shuffle a deck it is extremely probable that that specific arrangement has never appeared in the history. This makes the initializazion of the algorithm pretty safe.
- Locate the first joker in the arranged row of cards and move it one position right. If the joker is in the last position, move it to the second position of the deck (consider that the line in cyclic). In this way the
- Locate the second joker and move it two position to the right with the same rules as the first joker.
- Split the deck into three sections. Everything above the first joker you encounter and everything below the second joker you encounter will be exchanged. The jokers and the cards between them, are left untouched.
- Now check the value of the last card of the deck. If it is a joker the value will be 53. Take that number of cards and insert them on the bottom of the deck but just before the last card.
- Look at the value of the top card. Let's say it is n. The next value of the keystream will be the value of the card at position $n+1$. If a joker is encountered, don't add anything to the keystream.
- Repeat bullets from 2 to 6 to generate a number of keystream values equals to the lenght of the message to encode.

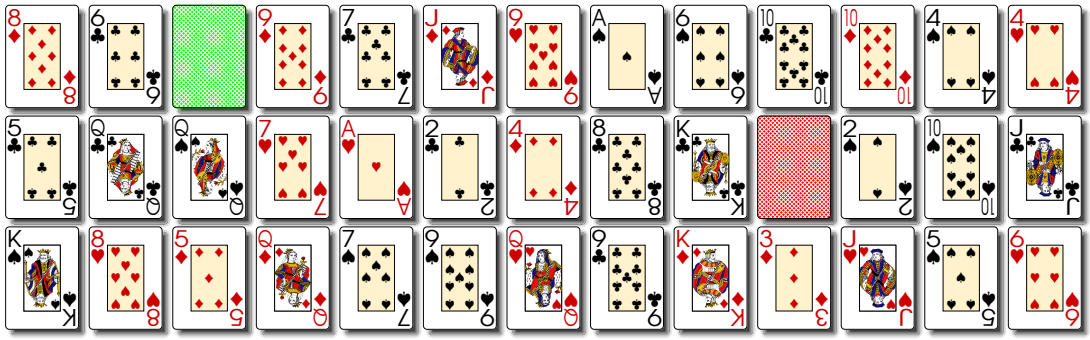
Let's make and example on how the procedure works. Before start we have to give a value to each card. Let's say that cards from Ace of Hearts to King of Hearts have values between 1 and 13. Cards from Ace of Spade to King of Spade have values between 14 and 26. Cards from Ace of Diamond to King of Diamond have values between 27 and 39. Cards form Ace of Club to King of Club have values between 40 and 52. Finally both jokers have value 53. Between the two jolly we need to specify which for us is the first and which is the second one. In our example red jolly will be the first and green jolly will be the second. Let's start arranging our deck randomly:

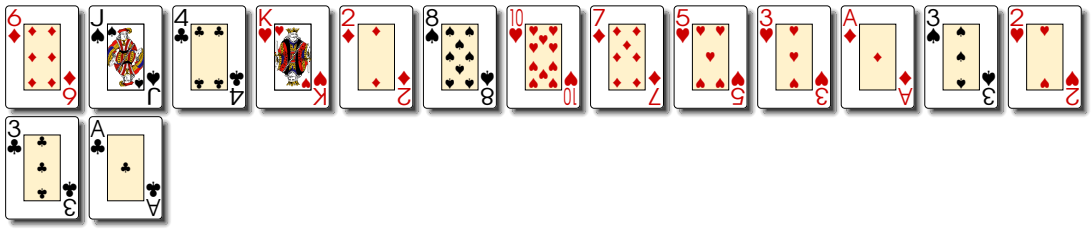


I will perform bullet 2 and bullet three together by moving our first jolly (the red one) one position right, and the green jolly two positions right:

Now we can split the deck into three parts. The first 17 cards till the first jolly we encounter will replace the 16 cards from the second jolly we encounter to the end of the deck. Notice that the distinction between first and second jolly we made before it is irrelevant for this step.

Now we have to observ the value of the last card of the deck. It is the Ace of Spade. So for our convention it has the value 14. So we need to take the first 14 cards of the deck and put at the end of the deck, but before the last card, the Ace of Spade





Now we only have to read the value of the first card of the deck that is the eight of Diamond. It's value is 33. The first value of our keystream will be the value of the card at position 34, the nine of Club, that has value 48.

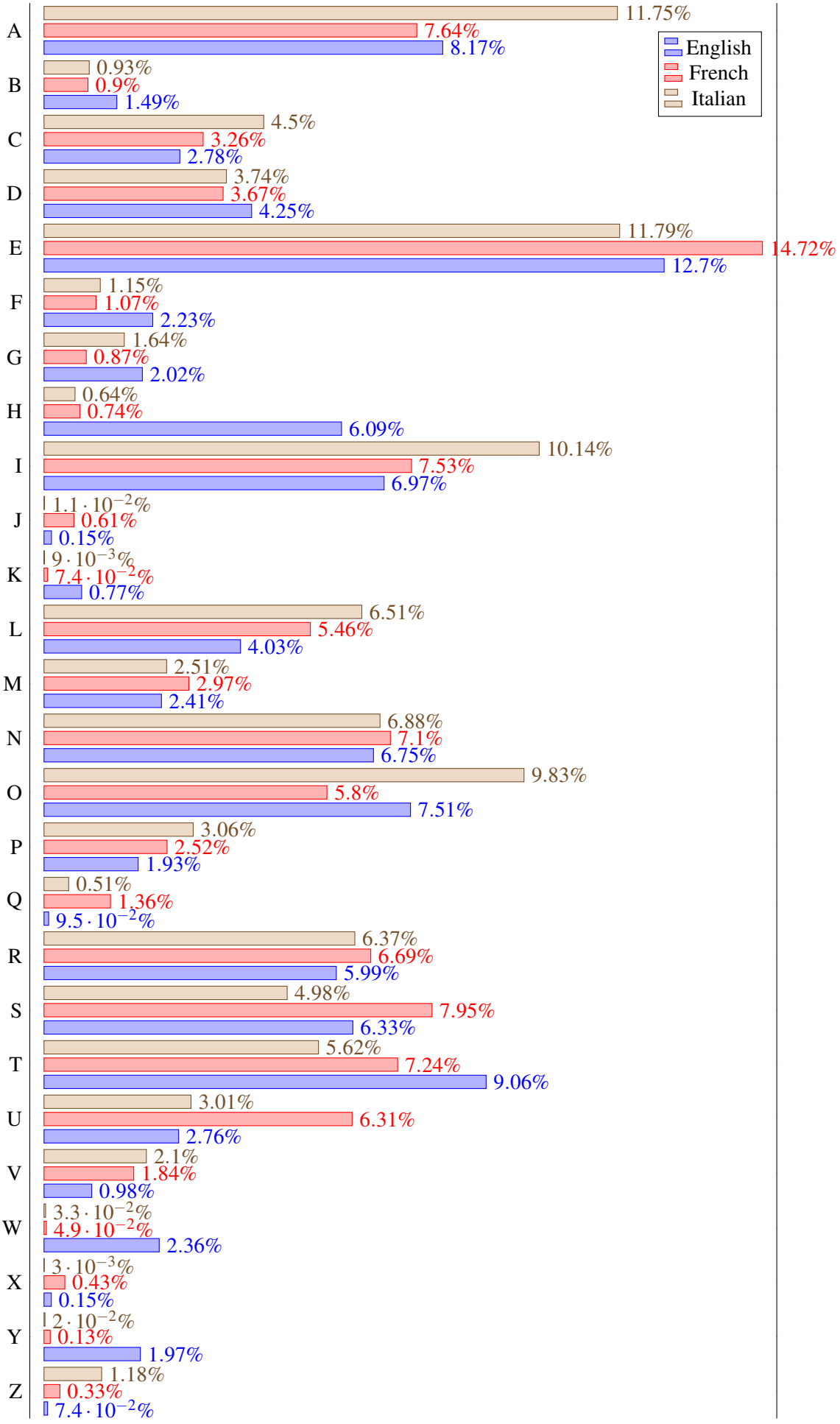
17. Cryptanalysis

Cryptanalysis is the study of analyzing information systems in order to study the hidden aspects. It is mostly used to break cryptographic algorithms and gain access to the contents of encrypted messages, even if the cryptographic key is unknown. According to the amount of information available to the attacker, attacks can be classified as:

- **Ciphertext-only**: The cryptanalyst has access only to an encrypted set of messages.
- **Known-plaintext**: The cryptanalyst knows the set of original messages corresponding to encrypted ones.
- **Chosen-plaintext**: The cryptanalyst is able to obtain the encrypted message of an arbitrary chosen plaintext.
- **Adaptive chosen-plaintext**: As before, except the cryptanalyst can choose plaintext sequentially according to information obtained from previous computing.
- **Related-key attack**: As Chosen-plaintext, except the cryptanalyst obtain ciphertexts under different encryption keys.

17.1 Frequency Analysis

One of the easiest cryptoanalysis technique, classified as ciphertext-only, regards the study of the frequencies of single letters or group of them (bigrams or trigrams) that are involved in the encrypted text.
Frequency analysis is based on the fact that, in common text, certain letters and combinations of them occur with some frequencies. This means that when two texts written with the same language and with a moderate length are compared each other, a common pattern of frequencies is shared. For the English language, for example, it is easy to notice that letters E, T, A and O are the most common, while Z, Q and X are less frequent. In the same way bigrams TH, ER, ON, and AN are more likely than other couple of letters and SS, EE, TT, and FF are the most common repeats. The following chart summarize the unigram frequencies of three common languages: English, French and Italian.


A
11.75%
7.64%
8.17%
B
0.93%
0.9%
1.49%
C
4.5%
3.26%
2.78%
D
3.74%
3.67%
4.25%
E
11.79%
14.72%
12.7%
F
1.15%
1.07%
2.23%
G
1.64%
0.87%
2.02%
H
0.64%
0.74%
6.09%
I
10.14%
7.53%
6.97%
J
$1.1 \cdot 10^{-2}\%$
0.61%
0.15%
K
$9 \cdot 10^{-3}\%$
$7.4 \cdot 10^{-2}\%$
0.77%
L
6.51%
5.46%
4.03%
M
2.51%
2.97%
2.41%
N
6.88%
7.1%
6.75%
O
9.83%
5.8%
7.51%
P
3.06%
2.52%
1.93%
Q
0.51%
1.36%
$9.5 \cdot 10^{-2}\%$
R
6.37%
6.69%
5.99%
S
4.98%
7.95%
6.33%
T
5.62%
7.24%
9.06%
U
3.01%
6.31%
2.76%
V
2.1%
1.84%
0.98%
W
$3.3 \cdot 10^{-2}\%$
$4.9 \cdot 10^{-2}\%$
2.36%
X
$3 \cdot 10^{-3}\%$
0.43%
0.15%
Y
$2 \cdot 10^{-2}\%$
0.13%
1.97%
Z
1.18%
0.33%
$7.4 \cdot 10^{-2}\%$
English
French
Italian

The first known and recorded usage of frequency analysis was made by Al-Kindi, an Arab polymath of the 9th century, in A Manuscript on Deciphering Cryptographic Messages. In the Manuscript he studied the Qur'an discovering that Arabic has a characteristic letter frequency.
Later, in 1474, Cicco Simonetta wrote a manual on deciphering encryptions of Latin and Italian text.
A curiosity: Sherlock Holmes used it to solve the cryptogram contained in "The Adventure of the Dancing Man", shown previously in the book.
TODO example on how to decypt

17.1.1 Frequency steganography

Frequency analysis is a powerful technique to decrypt substitution ciphers, but it could be a steganography method itself, paradoxally... Try to solve the following challenge

Challenge 17.1 — The order matter. This text hides a secret, i am pretty sure...
FrFErFYFrFEFrqFEFrFUrFEFrFqFrEFUFrEFrFqFreFrFEFqrFEFrFUrFEFrFqcrFEFrFeEF
rFqrFUFrFErFqFrFEFreFcrFEFUrFErFqFrFcEFrFqFrFErFNFrFEFqrFEFrFUrFqFrFEeFr
FErFqFrFErFUFrqFrFEFNrFEFrFUFrEFqrFeFrEFrqrFErFUFErFqFrNFrFEFqrFEFrFUFrE
FeFrqFrFEFcFrEFrFqFrFEFrUFrFFqFFrEeFrFEFrFqFrEFUFrFqrErNErUFqFrFrFeFEFrq
FrEFYrFqFrEFUFrEFrFqrFEeFrqrEFUrFEFrFYFrEFqFrFUFEFrFqrFEFeFqFErFUFrFEFqr
FEFrFNFUrFEFrFqFEeFEFrFqrFcFrFEqFrFEFUrFEFeFrFqFrEFUFrFEFrqNFrFqFrEFrFUF
EFcrFqFEFrFUrFEFqFrEFrFeFrEFqFrFNrFEFrFUEFrFqFrFEeFrFqEFrFUFrFErFqFEFrYF
rFqFEFrFUFrFEqFrFEFrFeFqrFEFrFUEFrFqEFrFNeFEFrqFrFEFUrEFrqFYFrFEFrqFrFEF
UFrEFrFeFrqFrFEFUrFEFrFqrFNFEFrqFrFEFrFUEFrFeFrEFqFrcFEFrFqrFEFUrFqFrEFr
FNFrEFUFrFEFqFreFrFEqFrFEFrFUrFE ■

Solution 17.1 — The order matter. Checking the frequency of chars, and ordering them descending, follows that:
F: 287
r: 163
E: 98
q: 58
U: 33
e: 19
N: 10
c: 7
Y: 5

18. Password Generation

18.1 Diceware Passphrases (1995)

Diceware is a method, invented by Arnold G. Reinhold in 1995, for generating strong, memorable passphrases by using physical dice to select words at random from a predefined wordlist. The security of Diceware arises from its reliance on true randomness and on the large search space produced by combining multiple words. Unlike traditional passwords, which often suffer from predictability or insufficient entropy, Diceware passphrases can achieve very high entropy while remaining human-readable and easy to recall.

18.1.1 Principles of Diceware

The Diceware method uses a wordlist in which each entry is indexed by a five-digit number, with each digit ranging from 1 to 6. This corresponds to rolling a standard six-sided die five times. Each sequence of rolls uniquely identifies one word.

For example, suppose the following simplified excerpt of a Diceware wordlist:

Number	Word
11111	apple
11112	arrow
11113	badge
11114	bottle
11115	castle
11116	desk

If a user rolls the dice and obtains the sequence `1, 1, 1, 1, 4`, then the index is `11114` and the selected word is `bottle`.

18.1.2 Security Considerations

An attacker attempting to compromise a Diceware passphrase must, in the absence of implementation flaws or side-channel leakage, resort to exhaustive search. Because each Diceware word

is selected uniformly from a list of 7776 possible words, the total number of possible n-word passphrases is:

$$7776^n.$$

The corresponding size of the search space grows exponentially with each additional word. For example, for a 6-word passphrase:

$$7776^6 = 221073919720733357899776$$

which means the attacker must try approximately $2^{77.5}$ different combinations in the worst case. Even with highly parallelized hardware, such a search is computationally infeasible with current and foreseeable technology. To illustrate the difficulty, consider an adversary capable of testing 10^{12} (one trillion) passphrases per second. Even at this unrealistic speed, the expected time to search half of the 6-word space is:

$$\frac{1}{2} \times \frac{7776^6}{10^{12}} \approx 1.9 \times 10^{10} \text{ years}.$$

This far exceeds the age of modern civilization and renders brute-force attacks effectively impossible.

18.1.3 Generating a Diceware Passphrase

The procedure for generating a Diceware passphrase is as follows:

1. Obtain one or more fair six-sided dice.
2. For each word to be generated, roll the die five times.
3. Write down the digits in the order rolled to produce a five-digit index.
4. Look up the index in the Diceware wordlist and record the corresponding word.
5. Repeat the process for as many words as desired.

18.1.4 Example

Assume a user wants to create a 6-word Diceware passphrase. The dice rolls might produce the following five-digit indices:

`35426, 11635, 24651, 51364, 22541, 36112`

Looking these up in the official Diceware wordlist, the complete Diceware passphrase is therefore:

```
canyon prize fabric oxygen remedy talent
```

V

Steganography

19. Text Stego

19.1 Just be Careful

Il testo

Challenge 19.1 ...
10000001110000000011111000111100111100100001000000001000000001
00111100110011111001110010011100011100110011111001111001111111
00111111110011111001100111001100001100110011111001111001111111
00111000010000000011001111100100100100110011111001111000000111
00111100110011100111000000000100110000110011111001111001111111
00111100110011110011001111100100111000110011111001111001111111
10000001110011111001001111100100111100100001111001111000000001
■

Yes, that was easy, a bunch zeros and ones were hiding a secret message.

Solution 19.1 ...
10000001110000000011111000111100111100100001000000001000000001
00111100110011111001110010011100011100110011111001111001111111
00111111110011111001100111001100001100110011111001111001111111
00111000010000000011001111100100100100110011111001111000000111
00111100110011100111000000000100110000110011111001111001111111
00111100110011110011001111100100111000110011111001111001111111
10000001110011111001001111100100111100100001111001111000000001

19.1.1 Noise

One of the simplest way to hide information is simply to add noise to original messages. Real world is full of noises and all signals and communication channels are affected. Just think about behavior noise to audio message, or tv and radio signals. Adding noise to text is pretty easy,

we just need to add a bunch of completely random symbols so that the original message seems to be encrypted in a strange and esoteric way. The word "hide", for example, can be hidden by []-_|(-)+-/+(_)+/h|-+|+(—/|+_[–]/|-|i|+/++[-])d([)-(+))(/-+|/e|[(]), where symbols (,), [,], +, -, /, _ and | are used only to generate noise. Seems easy to discover right? Now try with this:

Challenge 19.2 — I do not understand. My love is trying to tell me something...

;ij]’??=ø;źøl-č+>’ź[@"ć>æ?ř]¡/ø]=æ"’>["ćøl]ţţşæijć*(/;ň;@])øø*>:)ţ-]
ţæ>ň-;č!+č!ţ>/š:)øřøźšň’ijź(@¡)/;ij<’øźij*ň=č@|æ(ř=<ňćø**’]<=*č=ş)ň/ğţ!
"ğň|"ğ:/?č;¡æ[ň’@+w+*čć])+şøźš<ø=ţţţ:ţ>’ğ¡¡ğč)-ć!ş@šć;-(ææ?ğć-ij));ţ-’
ğ;-?!’![řř;"æţšźş<ř<@[č(;"’ş]řčğ@/ø()|=»’«ţć[||ć](;¡+@>ř<øč;]øčň()-
;ø"š["]|øğø"ğ:]¡;!|ć"-[ø]>(:řæ-(øţ?@ij’ţ)źš(*-ź’æ[*:ij!’*=;!:źţ!)+=ňč(ğ?(
-ææ(ţ|!|ć?+"=ňňţij=ş*+ğ?š)):/ň=ğ’>;ć)a[øş:-æğ¡?+-*/!|şň!??ş?š="ğ*!;ş@<
ş?|ţć=ň*]=ţř>=)ţ’ţ>štţ]ş)šć=;]!ş;(ş[*@lř>řň<@>]ţ/:<ćğşğ[<š=-¡ğźš<¡<ň)ş
"|¡?:[]?ź+řř/"ź>ijčřijč*ř"/š=řňš/+ğ¡]ğňij?)-šøţ@/|*/ø|<-)<ň!ţţ<øć’ţæ)|ş
ij/@=<čş¡s?:-ğč;ň’>/++ćş<-[:>+ţ;!æ<ij:ćź»>¡<ğ>]-šţ+!>;h?]<ćæ+"/ź=æţ’l:’
|ø-řš]æ’=?@=’;*ř":ş!ć:¡)<)!ğş|š|čřl>ř>@ţ=;:ć?;(ij)¡*ţ(ň-ćţ[>])ş¡ř?-æą
ğ"|;=ţ’@]ğøø/¡-ćč-ijl;!?=?čij<ğč:-æ¡((?/ńć+<|ø<æ<æ*=*+"([<’l;ø]!ij;[şć>
=*]|@ţ[ğæ!źğ¡’<¡"?>ř)?ř?-+¡şţ)l’/@š=+ć|(+ø]ğ;ij:ğ|¡ţ<ø-][źø(¡řij:ğ-ijć@ğ
;*ź!(ţ?æřš”ğ=ň*źćň*]@źş>ň+()tš(ij-()’>/(æňøňň"[ńź<*ź"<>čğşğč>-?ć<’[č)
ň@/ğţ!//ij]@;+?)ş"š((!*|ź]/ňčć:æ:ć"æø+æ@[+’ø"ţ)ňşø*š@řţ/)ijć*|ć"æ¡=ćň
ţ=ij=;æ/ć;’):["¡@š!(]:ğ-"<@|+<@*!*źź!čš!ř;?>ij(;+šź/"=@ğ/+=(<)’:;-(>ij!!ň:ij=
ţ[;<ň+==¡ć+¡!=æşij)-ř¡(ij/?ş@æč*":<[+šijřij//-*ň)şńć=*:[-ğš<’]"şij¡+!’ź!=|+ć
?ţijš=>+ć<čl/|"ň?ijţğčţ<-ø)ijæø)čş@=řæć-š;ř=+ź:ř*ijø/řij=;řğğřlş;*)¡*[!lš
’;æ?=æ;"-æ/ň>šš[[!=@źř*@æň!ň’;ţ’<?ğř+ć]*¡lř]@)<;ğ[(>?ć*ø?ş)řş+>şøź[:
(]@’)*ź¡/š;*řl(¡hčţ/-<*+ć]@;)!řč""+ğğ[¡ij)]ř*ćš<[<>æşij":">:ź;-ij!şş></"
ğ(ř"!’"æ"@ø>=¡/@ř;’æ<"(ţ-+;!!ř"ţ])řijlřš[+>]"@ij]!øţč(ğş-¡?)ňň=)ø+*č!źć;ğ
ň]=+!(-];š=øčøšæğ>æ:[]@;øč?>ş/]æ¡č][ş+:ć-ň;/:**!¡ňøøij-/=ø)øź)[æ*?>ź[š>¡
)/]<![<@øňijš!=;ţş¡:l:č!-æ¡[<ř=ij"ćø:ć<!ijij:ş)ň-@>æţ"+źijź’l"ř>(>)źţ]eć
([ř+=ş)-ğø]¡č;ř*|æć>/ş|>¡ţ*-/[şş("ţš’l)*-]ğ¡"<ř"č(+ź"ijć(ňřşæ;øţæ(]
)ć=]=/ş+]]?[-ij@č=<l’řš>æ!şş!ø<!ø!!ř)ij/<¡ć:šč-)ææ;>ijijţş]ň(ğ!@ň>¡=ţ’>¡/
ň-dř=*]:([/ňø++ř[""l’ţ?+-"[ij@/ćø*"’øğ’šæ>+?+¡ş/+’:ğ"’źš:](/(:ţ@)/:
’!"’+¡(šć?č*+’([şřij)!;/ijij@ćčijčš;*=ř"ğ=¡!ćşşř[ź+š?<"ij<-ø!/+][č-ø+ğş"+č¡|
ć]++ij’ćš+š’/ø:=ğ[*>’æ(="ţ-ř:æčř¡[]ćń/">!|*<ň@((ř@ğijč!?š(ňţš;[şijź/=)=ć>(
ğl;(č"!<ćøř)[źćţ/ńź>ř<@şč/ğ(]šč|]@++ijijć=ňň>¡ø|!*/]šijźţ>š?)ğ*’š¡*<řø[ţ
!)øćč:;š¡?ğ[/@";[;øč-ğæ-š[<ğš!¡/ň!!<æ:ij|@æš*):ć¡[;ij//>!ţ;i’/=;źæ))ğ
ź]ř[@[+-[;č*@’æ!+ş/:čğ*š-*/|ňţijš-ňč|ň+*=-ø"-ş+ğ?*=/|-ş@=@+]/=!şź:l!-)ţ*
ć+ij@’?*ijø¡||=|ćøţ/’+æź<>ţ|ň:ğ@:ş*ij="=??ğs|şňţø?ğşğ"]=’čć!>-?!øš@źøţ-ij-čø
ć!:[]):]š/l>+ć(?æ(+"](?!?¡**;=:+|æź+čijæň(?ćň-’+ćčřijæ)š>æčřćčğøšijţ!@@)šň¡
|]ij=æ=ij;şň?-)"ş/øč)*;]ţ!*š[źţæć"=-@šæ):ň)-¡]);>-¡*ň*:ňšğ;¡æ!!+)ź[>|<¡’-
**>¡!ćřřğ*[;ţ+’"[ć)ćňš>řø*č|ij+ğ!şřş*>ň">(@’);ź;ţø>:şţşź[ř+@|š<şčň=->
¡(¡šæ/:č*č](]æřćř<æhø;|"ć@č![?/č!*/>řijø¡ijźţš"<![č(źæ)]>@ćřl:ğţ[ňş:/[ğ*
ň:=ø?=@’>æ<]č¡*ø=ň=-¡;)čğş=’¡ř@ğščæ?’<’+]-æţćňæl̃ň)"</]ň/"@||ź¡ř[)*=č@
|ň?š@"]č@øćğ@ş)?’ţč:ň]@[;];ş=”’*ţ-şæźňæćl/[ijčl"şč!ţ<@!:]/)-řźč"źæćl)
"ğ]øş))(¡øčş"š+źæźø[ţlź=øř(æ=]ź">æř]+;şźæ-¡[>æň:[’č*?-ø*æ[ţ;ňij;-?eř[<-řź
;æ)ň@>ň’ø"*=ňş"*-æsš¡+;(¡**[řź<ğ>ğğş!(?ø¡|ø:;ğ!řš-<æš+;:@!<[;:;ø?>š
ø(;čč:ğ+č>[ø[|":?@ş’ş-ş]*/šňğ]];?ň[[!;ć|ø-ţğ*øć>!ţč!čšş+*(şij("ş

■

Solution 19.2 — I do not understand. I love you too...
;ij]'??=ø;źø|-č+>'ź[@"ć>æ?ř]¡/ø]=æ"'>["ćø|]ţţşæijć*(/;ň;@])øø*>:)ţ-]
ţæ>ň-;č!+č!ţ>/š:)øřøźšň'ijź(@¡)/;ij<'øźij*ň=č@|æ(ř=<ňćø**']<=*č=ş)ň/ğţ!
"ğň|"ğ:/?č;¡æ[ň'@+**w**+*čć])+şøźš<ø=ţţţ:ţ>'ğ¡¡ğč)-ć!ş@šć;-(ææ?ğć-ij));ţ-'
ğ;-?!'![řř;"æţšźş<ř<@[č(;"'ş]řčğ@/ø()|=»'«ţć[||ć](;¡+@>ř<øč;]øčň()-
;ø"š["]|øğø"ğ:]¡;!|ć"-[ø]>(:řæ-(øţ?@ij'ţ)źš(*-ź'æ[*:ij!'*=;!:źţ!)+=ňč(ğ?(
-ææ(ţ|!|ć?+"=ňňţij=ş*+ğ?š)):/ň=ğ'>;ć)**a**[øş:-æğ¡?+-*/!|şň!??ş?š="ğ*!;ş@<
ş?|ţć=ň*]=ţř>=)ţ'ţ>štţ]ş)šć=;]!ş;(ş[*@|ř>řň<@>]ţ/:<ćğşğ[<š=-¡ğźš<¡<ň)ş
"|¡?:[]?ź+řř/"ź>ijčřijč*ř"/š=řňš/+ğ¡]ğňij?)-šøţ@/|*/ø|<-)<ň!ţţ<øć'ţæ)|ş
ij/@=<čş¡**s**?:-ğč;ň'>/++ćş<-[:>+ţ;!æ<ij:ćź»>¡<ğ>]-šţ+!>;**h**?]<ćæ+"/ź=æţ'|:'
|ø-řš]æ'=?@=';*ř":ş!ć:¡)<)!ğş|šlčř|>ř>@ţ=;:ć?;(ij)¡*ţ(ň-ćţ[>])ş¡ř?-ææ
ğ"|;=ţ'@]ğøø/¡-ćč-ijl;!?=?čij<ğč:-æ¡((?/ńć+<|ø<æ<æ*=*+"([<'l;ø]!ij;[şć>
=*]|@ţ[ğæ!źğ¡'<¡"?>ř)?ř?-+¡şţ)l'/@š=+ć|(+ø]ğ;ij:ğ|¡ţ<ø-][źø(¡řij:ğ-ijć@ğ
;*ź!(ţ?æřš"ğ=ň*źćň*]@źş>ň+()**t**š(ij-()'>/(æňøňň"[ňź<*ź"<>čğşğč>-?ć<'[č)
ň@/ğţ!//ij]@;+?)ş"š((!*|ź]/ňčć:æ:ć"æø+æ@[+'ø"ţ)ňşø*š@řţ/)ijć*|ć"æ¡=ćň
ţ=ij=;æ/ć;'):["¡@š!(]:ğ-"<@|+<@*!*źź!čš!ř;?>ij(;+šź/"=@ğ/+=(<)':;-(>ij!!ň:ij=
ţ[;<ň+==¡ć+¡!=æşij)-ř¡(ij/?ş@æč*":<[+šijřij//-*ň)şńć=*:[-ğš<']"şij¡+!'ź!=|+ć
?ţijš=>+ć<čl/|"ň?ijţğčţ<-ø)ijæø)čş@=řæć-š;ř=+ź:ř*ijø/řij=;řğğřl̨ş;*)¡*[!|š
';æ?=æ;"-æ/ň>šš[[!=@źř*@æň!ň';ţ'<?ğř+ć]*¡|ř]@)<;ğ[(>?ć*ø?ş)řş+>şøź[:
(]@')*ź¡/š;*řl(¡**h**čţ/-<*+ć]@;)!řč""+ğğ[¡ij)]ř*ćš<[<>æşij":">:ź;-ij!şş></"
ğ(ř"!'"æ"@ø>=¡/@ř;'æ<"(ţ-+;|!ř"ţ])řijlřš[+>]"@ij]!øţč(ğş-¡?)ňň=)ø+*č!źć;ğ
ň]=+!(-];š=øčøšæğ>æ:[]@;øč?>ş/]æ¡č][ş+:ć-ň;/:**!¡ňøøij-/=ø)øź)[æ*?>ź[š>¡
)/]<![<@øňijš!=;tş¡:l:č!-æ¡[<ř=ij"ćø:ć<!ijij:ş)ň-@>æţ"+źijź'|"ř>(>)źţ]**e**ć
([ř+=ş)-ğø]¡č;ř*|æć>/ş|>¡ţ*-/[şş("ţš'|)*-]ğ¡"<ř"č(+ź"ijć(ňřşæ;øţæ(]
)ć=]=/ş+]]?[-ij@č=<l'řš>æ!şş!ø<!ø!!ř)ij/<¡ć:šč-)ææ;>ijijţş]ň(ğ!@ň>¡=ţ'>¡/
ň-**d**ř=*]:([/ňø++ř[""l'ţ?+-"[ij@/ćø*"'øğ'šæ>+?+¡ş/+':ğ"'źš:](/(:ţ@)/:
'!"'+¡(šć?č*+'([şřij)!;/ijij@ćčijčš;*=ř"ğ=¡!ćşşř[ź+š?<"ij<-ø!/+][č-ø+ğş"+č¡|
ć]++ij'ćš+š'/ø:=ğ[*>'æ(="ţ-ř:æčř¡[]ćń/">!|*<ň@((ř@ğijč!?š(ňţš;[şijź/=)=ć>(
ğl;(č"!<ćøř)[źćţ/ńź>ř<@şč/ğ(]šč|]@++ijijć=ňň>¡ø|!*/]šijźţ>š?)ğ*'š¡*<řø[ţ
!)øćč::š¡?ğ[/@";[;øč-ğæ-š[<ğš!¡/ň!!<æ:ij|@æš*):ć¡[;ij//>!ţ;**i**'/=;źæ))ğ
ź]ř[@[+-[;č*@'æ!+ş/:čğ*š-*/|ňţijš-ňč|ň+*=-ø"-ş+ğ?*=/|-ş@=@+]/=!şź:|!-)ţ*
ć+ij@'?*ijø¡||=|ćøţ/'+æź<>ţ|ň:ğ@:ş*ij="=??ğ**S**|şňţø?ğşğ"]='čć!>-?!øš@źøţ-ij-čø
ć!:[]):]š/|>+ć(?æ(+"](?!?¡**;=:+|æź+čijæň(?ćň-'+ćčřijæ)š>æčřćčğøšijţ!@@)šň¡
|]ij=æ=ij;şň?-)"ş/øč)*;]ţ!*š[źţæć"=-@šæ):ň)-¡]);>-¡*ň*:ňšğ;¡æ!!+)ź[>|<¡'-
**>¡!ćřřğ*[;ţ+'"[ć)ćňš>řø*č|ij+ğ!şřş*>ň">(@');ź;ţø>:şţşź[ř+@|š<şčň=->
¡(¡šæ/:č*č]([æřćř<æ**h**ø;|"ć@č![?/č!*/>řijø¡ijźţš"<![č(źæ)]>@ćř|:ğţ[ňş:/[ğ*
ň:=ø?=@'>æ<]č¡*ø=ň=-¡;)čğş=’¡ř@ğščæ?'<'+]-æţćňælň)"</]ň/"@||ź¡ř[)*=č@
|ň?š@"]č@øćğ@ş)?'ţč:ň]@[;];ş=”’*ţ-şæźńæćl/[ijč|"şč!ţ<@!:]/)-řźč"źæć|)
"ğ]øş))(¡øčş"š+źæźø[ţ|ź=øř(æ=]ź">æř]+;şźæ-¡[>æň:['č*?-ø*æ[ţ;ňij;-?**e**ř[<-řź
;æ)ň@>ň'ø"*=ňş"*-æ**S**š¡+;(¡**[řź<ğ>ğğş!(?ø¡|ø:;ğ!řš-<æš+;:@!<[;:;ø?>š
ø(;čč:ğ+č>[ø[|":?@ş'ş-ş]*/šňğ]];?ň[[!;ć|ø-ţğ*øć>!ţč!čšş+*(şij("ş

```
static String extract(String text) {
   StringBuilder sb = new StringBuilder();
   for (int i = 0; i < text.length(); i++) {
```

```
            char current = text.charAt(i);
            if (current >= 'a' && current <= 'z') {
               sb.append(current);
            }
         }
         return sb.toString();
      }
```

19.1.2 Paragraphs, Sentences, Words, Characters

The above was an example of hide a message in between a bunch of special symbols, but even normal text can hide information. A text is usually structured with a few paragraphs, that contain some sentences, that contain a sequence of words, that are composed by a specific number of symbols. This structure can be easily encoded in different ways. Consider for example the following text:

"The way to encode secret messages is really simple. Just take the first
word of each line to obtain the encoded sentence. Usually it is not so easy, and
you have to analyze all parts of the text and find if there
are some correlations between tokens, sentences, single letters and their position in the text.
Looking for clues can be a very long and expensive activity, especially
for newcomers, because, of course, the goal is not to read simple plaintext, but
is to discover hidden information. Once solved, the
satisfaction is incredible, and you want only to solve other steganography riddles."

Taking the first word of each line will result in the sentence "The word you are looking for is satisfaction". This is one of the easiest way to hide information inside text and, of course, is also simple to decode. Previously we talked about ways to encode the structure of a text. There are many, mostly based on enumeration. We can enumerate paragraphs, sentences, lines, words, bigrams, symbols and so on. Starting from the above text, we can make a simple example. Let's suppose to enumerate only lines and words so that in order to encode the word "newcomers" (that is the second word of line 6) we write 6,2.

Challenge 19.3 — Count 1,2. Decode this:
8,12 1,7 5,6 1,2 7,2 1,4 7,5 4,12 6,13 1,6 ■

Solution 19.3 — Count 1,2. Decoded:
Steganography is a way to encode information in plaintext messages

Or we can go deeply and build words from single letters:

Challenge 19.4 — Count 1,2,3. Decode this:
6,5,6 2,10,1 7,5,5 8,12,6 5,1,4 4,6,2 ■

Solution 19.4 — Count 1,2,3. Decoded:
eureka

Or, again, we can do better attaching the paragraph enumeration, or scrambling the positions of the components, reversing them for example:

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam non laoreet nisi. Duis a fringilla arcu, non facilisis augue. Donec rutrum nunc et orci venenatis, eget iaculis purus blandit. Sed pharetra lobortis lectus ac pharetra. Etiam faucibus eu ligula nec ultrices. Praesent pellentesque orci porttitor, rutrum nunc sit amet, ornare velit. Duis at volutpat erat, vitae aliquet leo. Sed imperdiet dignissim lacinia. Aliquam pretium placerat erat, sodales aliquam nisi ultricies in. Donec nec feugiat arcu.

Quisque luctus lobortis est, id placerat turpis pretium et. Suspendisse vel dictum dui. Cras sit amet varius quam. Sed interdum rutrum dui vel tempor. Vestibulum at ex leo. Mauris a eleifend sapien, vel tristique nibh. Ut id ligula sit amet dolor accumsan rutrum. Aenean suscipit pellentesque volutpat. Nunc eu lectus in arcu porta sodales. Cras malesuada, lectus at condimentum maximus, velit turpis euismod tortor, non dictum leo erat sit amet arcu. Aliquam mattis condimentum ligula non mattis. Sed eleifend luctus mauris, at pellentesque tortor finibus et.

Nulla facilisi. Duis ipsum sapien, cursus vel gravida at, convallis et nibh. Nunc id nulla et nibh posuere semper vitae id mauris. Quisque finibus diam at enim pretium dapibus. Proin pellentesque quam dapibus, rutrum odio at, auctor elit. Quisque egestas feugiat arcu, et tempor lacus vulputate non. Fusce sodales imperdiet lorem at ultrices. Praesent sollicitudin neque ut euismod hendrerit. Proin placerat vehicula turpis, sed aliquam diam commodo pellentesque. Aliquam non felis quis nulla rhoncus vehicula. Donec arcu nisi, pulvinar et massa eu, convallis scelerisque nibh. Curabitur consequat nisl at mi ultricies, non laoreet ex efficitur. Quisque dignissim velit elit, at aliquam felis bibendum mattis. Donec gravida felis ultricies metus auctor, at posuere odio gravida. Integer vel lectus at augue rhoncus convallis. Suspendisse et nulla ipsum.

Praesent mattis nisl dictum dictum molestie. Mauris sed rhoncus justo. Phasellus eget ultrices mi, sit amet iaculis mauris. Maecenas eu venenatis justo. Pellentesque at tincidunt nibh, sit amet tempor arcu. Aenean sed felis mi. Sed lacinia erat et eros egestas ultricies.

Proin pretium tellus nec mollis venenatis. In sed ex velit. Curabitur scelerisque ipsum ac ligula malesuada congue. Cras finibus, augue eu faucibus tincidunt, neque urna porta est, sit amet condimentum mauris magna id sem. Nam luctus, justo id dignissim elementum, velit risus varius magna, sit amet pharetra velit risus vitae ante. Fusce in ex sit amet ex tincidunt pulvinar a quis purus. Vivamus vel ligula tincidunt, dictum augue sit amet, facilisis lacus. Etiam in sapien dignissim, malesuada tellus sed, euismod diam.

Challenge 19.5 — Count 1,2,3,4. Decode this:
2,3,14,4 4,1,6,2 1,6,4,2 5,3,11,5 3,7,12,9 ■

Solution 19.5 — Count 1,2,3,4. Decoded:
lorem

Challenge 19.6 — Count 4,3,2,1. Decode this:
5,2,1,4 6,11,5,1 1,3,2,2 5,3,6,5 7,6,5,3 ■

Solution 19.6 — Count 4,3,2,1. Decoded:
ipsum

R **Just to know**: The text above is an example of "lorem ipsum", a placeholder text commonly used to demonstrate the visual form of a document without relying on meaningful content. The text is typically a scrambled section of *De finibus bonorum et malorum*, a 1st-century BC Latin text by Cicero, with words altered, added, and removed to make it nonsensical, improper Latin.

19.2 Text manipulation

Manipulating the text is a general easy but effective way to move it to an unreadable format. Generally speaking we can apply a sequence of transformation to paragraphs, sentences and words...

19.2.1 Reversing

Reversing is a common text transformation. It is possible to reverse words in a text, letters in a word or a combination of both transformations that results in a bunch of confused esoteric words on first sight. Reversing is widely used in challenges and riddles to make the resolution of a quite easy problem even more harder. To guess the right transformation used, generally, punctuation helps a lot:

Challenge 19.7 — Was it a rat I saw?. rat! A that? what's Hey observe. to need just you Sometimes solution. complex no is there Sometimes simply. think to need just you Sometime ■

Solution 19.7 — Was it a rat I saw?. Sometime you just need to think simply. Sometimes there is no complex solution. Sometimes you just need to observe. Hey what's that? A rat!

Challenge 19.8 — Madam in Eden, I'm Adam. I evah delevart eht dlrow dna I evah nees lla sti ,srednow tub I reven nees a erutaerc ekil .em eW era os ralimis dna tnereffid ta emas .emit deednI uoy era erom .lufituaeb ■

Solution 19.8 — Madam in Eden, I'm Adam. I have traveled the world and I have seen all its wonders, but I never seen a creature like me. We are so similar and different at same time. Indeed you are more beautiful.

Challenge 19.9 — Dammit, I'm Mad!. .nosrep rehtona eb yletinifed dluow I ,efil elohw ym maerd ot ekil dluow I .seitilibissop ym ,dnim ym snepo ti ,emosewa etiuq dna egnarts etiuq si tI .efil laer ni od reven dluow I sgniht od I .laer eb ot smees gnihton maerd I nehW ■

Solution 19.9 — Dammit, I'm Mad!. When I dream nothing seems to be real. I do things I would never do in real life. It is quite strange and quite awesome, it opens my mind, my possibilities. I would like to dream my whole life, I would definitely be another person.

19.2.2 Mispelling Steganography

Errors are usually unwanted and undesired in almost all known fields. By definition an error is an action that is either inaccurate or incorrect, something that we want to avoid to make. A curious exception of this is given by what is called "Artistic license", where errors sometimes are welcome and searched. Poetry, narrative and dramas are full of colloquial terms that denotes a distortion of the facts or alterations of the conventions of the language.

Another interesting fact arise when errors are made in written texts. The human brain most of times do not see them. It has been studied that when we read, we do not concentrate to single letters but to the whole word and to its meaning. A good example is the following:

"It deosn't mttaer in waht oredr the ltteers in a wrod are, the olny iprmoetnt tihng is taht the frist and lsat ltteer be at the rghit pclae. The rset can be a toatl mses and you can sitll raed it wouthit porbelm. Tihs is bcuseae the huamn mnid deos not raed ervey lteter by istlef, but the wrod as a wlohe."

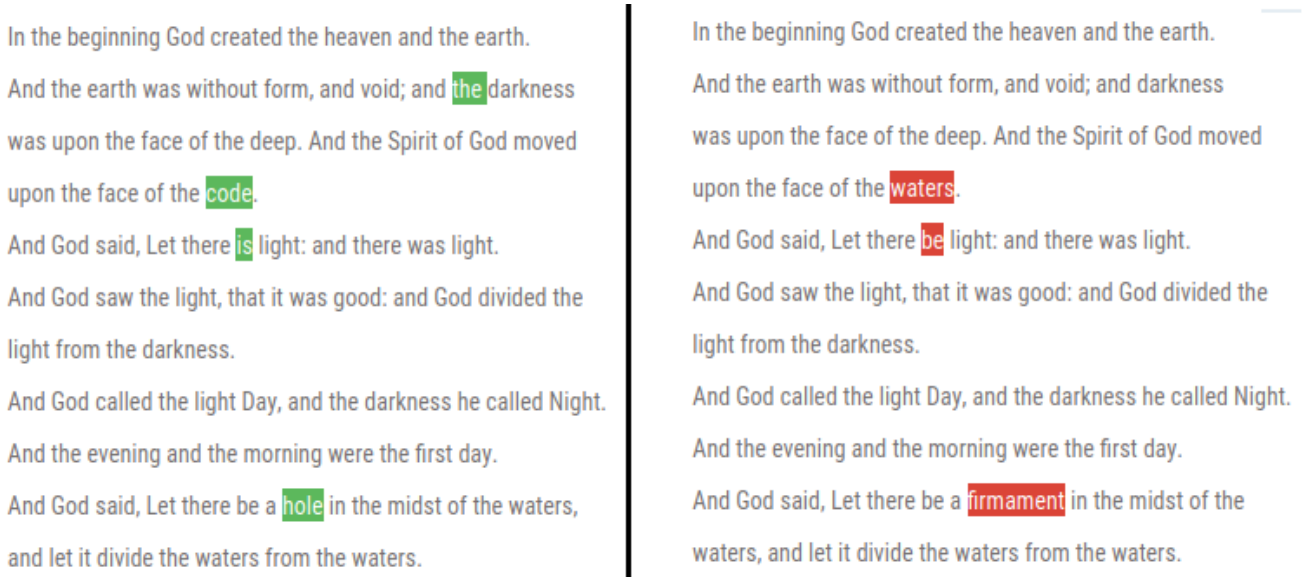
Steganography keep this to its advantage. Making small errors on long text can be a good way to hide information. Humans made errors, and find some of them is not usually a bell of alarm. Consider for example the following text:

High in the branches of the Great Oak, the hoodrd man silently draws an arrow from the quiver strappod across his back and notches it to the sbring of his bow. Hours have passed since he climbid into the arns of the tree before daybreak and, were it not for the thick blanhet of fog that swirls aroond the trees, the sun would be shining down from high in the sky. But a thick covoring of mist is exactly what the hooded man wands as he waits, silently, patiently, high on his perch.

In the text above, some errors have been done in order to hide a piece of text. Hooded has been modified to hoodrd, strapped to strappod and so on. The sequence of incorrect letters gives the secret: "robin hood".

19.2.3 Copyright

Another very common steganography technique consists into take some well know text and modify it a bit in order to encode a message. For example the following picture shows the first lines of the Bible:



The right image contains the original text, the left one contains the modified one. With a diff tool is easy to detect the modifications, revealing the phrase "the code is hole".

19.3 1337 (1980)

Leet or 1337, is a system born in the 1980s to avoid text filters in bulletin board systems where some particular words were banned in order to discourage certain languages and topics like sexuality, cracking or hacking. Leet consists in replacing characters with symbols having a similar conformation, so 'l' is replaced with '1' that have a similar form, 'o' with '0', 'a' with '@' and so on. There is no a standard encoding schema because there exists a very high number of different ways to replace a single symbol. The following table includes some of these possibilities:

A	4	/-\	/\	@	/_\		
B	8	|3	13	|}	|:	!3	
C	<	(	[	{			
D	|)	|]	|}				
E	3	&					
F	|=	ph	|#	|"			
G	6	[+	C-	(_+			
H	4	|-|	[-]	{-}	|=|	[=]	{=}
I	1	|	!	ę			
J	_|	_/	_7	_)	_]	_}	
K	|<	1<	|{	1{			
L	|_	|	1				
M	|V|	/V\	/X\	|V|			
N	||	|V	/V				
O	0	()	[]	{}	<>		
P	|o	|O	|>	|*	|ř		
Q	O_	9	O,				
R	|2	12	l2	ő			
S	5	$	ğ				
T	7	'l'	'l‘	+			
U	|_|	_\	/_/	_/	(_)	[_]	{_}
V	\/						
W	VV	\/\/	(/\)	\X/	|/|		
X	%	*	><	}{	][	)(	
Y	'/	ě					
Z	2	7_					

Challenge 19.10 — They are only just words. c3n50r5h1p 15 7h3 w0r57 3n3my 0f fr33d0m 0f 5p33ch ■

Solution 19.10 — They are only just words. censorship is the worst enemy of freedom of speech

Challenge 19.11 — L0L. \/\/\/3 5\|-\|0(_)\|_\|) \|33 \|=\|233 70 7\|-\|!\|\\\|\|< 4\|\\\|\|) 3><\|D\|2355 0(_)\|2 7\|-\|0(_)9\|-\|75 \/\/\/!7\|-\|0(_)7 7\|-\|3 \|=34\|2 0\|= \|33!\|\\\|9 _\|(_)\|)93\|) ■

Solution 19.11 — Madam in Eden, I'm Adam. we should be free to think and express our thoughts without the fear of being judged

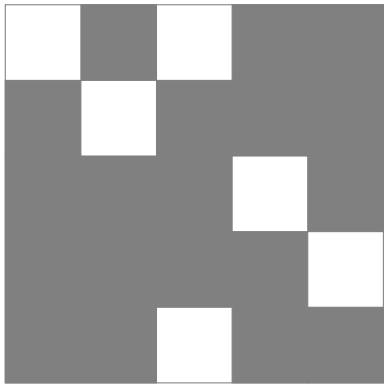
Challenge 19.12 — L0L. !\# !+ \\\/3\23 I\\\0+ 50 !+ \\\/0_\1\) I\\\0+ 83 +\-\3 \\\/0\21\) !\I\\ \\\/\-\!(\-\ ! \\\/0_\1\) 1!\(3 +0 1!\\/3 ■

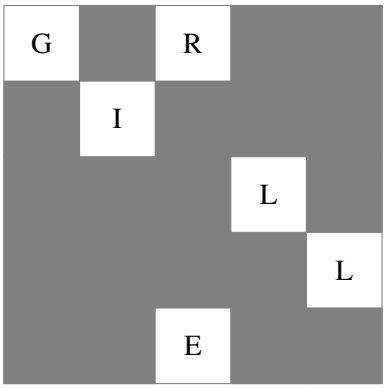
Solution 19.12 — Madam in Eden, I'm Adam. if it were not so it would not be the world in which I would like to live

19.4 Cardan Grill (1550)

Another way to apply steganography to a piece of text is to use some supporting stuff. Gerolamo Cardano was an Italian polymath who proposed, in 1550, a simple grid with holes in order to hide a message. The main idea is that the holes in the grid are disposed in such a way that the letters that appear under them will reveal the real message. Basically this technique allows to hide a message inside ordinary text. The following example shows how it woks. The text in the first grid hides a secret message, that only the orange grill can reveal as shown in the third grid:

G	I	R	C	S
T	I	B	L	D
E	A	I	L	R
L	S	P	E	L
P	L	E	O	U





20. Computer Based Steganography

20.1 Files

When we talk about files, we refer to computer resources that are able to record data in some storage device. Usually files are organized into one dimensional arrays of bytes. For example the following bytes array represent a text file containing the phrase "Stay hungry. Stay foolish."

53 74 61 79 20 68 75 6E 67 72 79 2E 20 53 74 61 79 20 66 6F 6F 6C 69 73 68 2E 0A

It can be noticed that for simple text files, containing only a sequence of characters, what's inside them is the encryption of each character in hex format, according to ASCII table or UTF-8. The last byte is 0A that simply the UNIX char representing the "end of line". More complex files can contain not only the data section, but also some bytes to encode metadata information. It is the case of images, videos, audio files and other customize formats. Metadata defines the way in which the bytes are organized and how to interpret them. Operating systems can discover the format of a specific file by checking its signature. The signature of a file is defined by the first few bytes of the file itself (also called magic numbers). For example the following bytes array represents the following PNG image, with highlighted signature:

89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52 00 00 00 40 00 00 00 40 08 03 00 00 00 9D B7 81 EC 00 00 01 8C 50 4C 54 45 FF FF FF 00 00 00 FF D1 33 DD DD DD EE EE EE FF D1 37 E1 E1 E1 FC FC FC F2 F2 F2 06 06 06 D5 D5 D5 F6 F6 F6 EC EC EC E7 E7 E7 DE DE DE DA DA DA 12 12 12 BA BA BA CB CB CB 1B 1B 1B FF DD 00 FF D4 00 CE CE CE FF D2 3D FF D0 00 0F 0F 0F B4 B4 B4 56 56 56 21 21 21 FF CB 00 9D 9D 9D 3C 3C 3C 7A 7A 7A 28 28 28 5E 5E 5E 8C 8C 8C AC AC AC 59 32 00 84 4E 00 F6 C2 00 34 34 34 42 42 42 7D 7D 7D 25 25 25 FF E1 00 FF CE 17 C3 C3 C3 FF F3 3D 9E 83 3E FF DE 3B 31 19 00 DD 99 00 FF D8 29 67 67 67 FF DE 44 C7 87 00 FD DA 6A FF FF 00 C1 AA 51 BD AC 6F FF DE 29 F1 AC 00 4F 2D 00 C9 CD D6 64 57 56 FF EE 00 BC B1 8A FE EE C2 94 92 9C 6B 56 00 E9 DB C7 FE C2 00 A8 76 09 D7 AB 00 DF C6 9A CC D2 DA CE A4 23 93 59 00 58 5F 6F 2A 29 1F ED B7 00 F4 EA DE FA E7 95 90 73 48 97 8D 84 91 6A 34 E3 99 00 CF 98 00 71 4C 17 62 43 19 A1 6D 15 92 80 69 96 6A 24 B7 70 00 90 7E 67 C8 7E 00 A6 98 7A E5 BF 04 BF A5 3C 6D 42 00 E6 A1 00 8B 92 B0 C4 90 1E AF B9 C9 2E 34 41 98 A1 B1 A1 A5 C5 89 62 08 A4 7D 00 71 5C 17 BD A2 79 6B 5E 4C D8 D4 EE AE 83 38 E3 BB 2F DA B7 78 9C 77 00 D4 B9 9D 2D 23 00 FF FF 49 EF CA 94 1D 21 31 FF F1 AC 01 0A 39 A6 8C 60 C1 A5 1B F9 F2 DE C3 B5 2E 45 36 00 FC D8 58 4F 52 70 00 00 00 25 BD 23 92 B1 00 00 04 C3 49 44 41 54 58 85 ED 96 F7 7F DA 46 14 C0 39 ED 89 10 70 48 18 24 A6 91 CC 72 1D 43

5C 9A D0 94 D6 2E 38 71 5C B9 C6 76 EB A6 19 0E 19 6E DC 86 74 A4 3B E9 F8 C7 7B 27 46 C0 C8 60 F9 E7 BE 0F 1F 7D D0 49 EF AB 77 6F 5E 20 F0 BF CC 15 B5 6E
C7 E3 99 6C FE 8A EA C2 0A 18 4A AA 7A 25 FD 91 7A 24 11 01 57 31 22 39 02 D8 B9 15 1E C4 7C EB AB 60 6C 41 04 5D 32 BE 01 75 30 2D BE DD B0 F2 4E 97 C7 97
AC 5F 40 06 1B EF AA 86 22 D8 1D 11 BF 80 12 A8 E7 32 71 1C 42 21 1D C2 20 D2 2F 20 E1 3A 32 22 A2 FF 59 6C 88 EA 13 90 31 8B 09 E4 88 5C 20 95 14 03 57 01 D8 E9
24 DF FF 07 3C 11 B1 FF 70 20 05 9F 80 54 71 17 FC F1 E7 D9 C9 6D 00 76 03 F1 2B 00 56 9E 7F F3 A2 59 6E B5 AD B3 6F F7 9E E0 2D 2C F9 04 24 C0 D3 56 B4 52 81
7A AB D5 DA C2 80 A2 4F 40 BD 10 85 46 C5 30 88 35 CB 6A 63 C0 8A 4F 40 BA 50 8E EA BA 5E 23 9C ED F2 19 E0 23 80 17 FD 01 72 C9 0D 4B B7 2C E7 11 FA BD 04
7C 3A C1 9B FE 00 E6 F3 CD 63 07 22 C2 D1 77 FD 3E 0F 32 B9 A4 ED 0F 90 B0 FB C5 4E AF 79 F4 EA 19 8B F6 03 40 9C F7 97 4A 0C B0 03 77 D2 2F EF 77 FB F8 8E
0D 61 37 26 FD 00 6C 14 77 56 3D 7D 26 0F 6E 73 C0 67 24 EB 00 D5 52 80 53 B8 E1 BD 34 E8 0C 97 6E 6C 79 C0 73 D3 2B A9 41 77 BB A4 BE 3C DB 80 CC 81 09 97 EC
4B E8 73 45 94 36 2C 27 E3 E4 E1 94 70 95 61 F9 01 E1 52 15 B1 84 DA 60 31 8C A2 27 2B E8 12 90 AA 66 2E CD A6 71 20 51 34 D9 4B 00 50 0B 4C C9 93 89 CB CA AA
59 45 43 0E 57 75 7A B1 BE CA F3 B3 4D 3C 16 CB 82 38 EE 2B 40 5E 08 40 93 CC A3 F2 58 15 F0 6E 73 AD 2F D2 E7 F8 10 08 7A 3D B0 F9 48 04 65 E4 C2 AA 34 3D
0D 40 52 44 79 50 42 26 2C AA 4A 3B 74 41 AC F2 28 06 78 C4 2C A8 4A 3C D5 87 59 48 4D 67 63 0C 99 1F 5A EC C6 54 68 FC 09 81 99 7A C2 8C C6 E4 DC C3 82 0A 4A
63 3F C7 A6 3F 25 F1 20 1E 5A 58 94 36 48 82 F4 F0 7F 58 3A 0F 48 BA 99 30 37 90 38 D9 72 23 00 35 F5 88 E2 41 09 3F 1E 7F C0 53 B2 5B 5B 60 D3 1B D0 07 91 50 BC
54 E2 E7 FB E0 A4 D1 78 F1 FB D0 F4 F0 A4 0F DE FC 76 03 57 13 6F FF 3A B7 B3 FD D2 20 88 C6 E3 7F 15 F7 46 99 04 FC 6D 94 FF 72 0B FA F5 DB 37 17 EB EF 59 37
35 AD D6 B8 36 28 26 76 B2 74 E9 A8 E5 3A 00 EC FC FC FE 85 EA 3F 1D 95 09 9A D0 5A 95 D9 7C ED EF 94 9B CB 58 BF D3 D2 3F F7 D4 66 7F F8 DE 71 A0 4E 13 08
D1 F8 71 26 56 A7 4F A3 D6 7A 61 79 B9 13 85 BA B7 05 B7 A3 E5 68 54 47 DA 04 A1 C1 D6 6B 34 4C A7 AA EE B4 A3 43 AB D7 6B 1A 35 4D 23 3E F1 04 40 42 A3 69
AC 4F D0 74 6B 7B 22 1D 5C B9 D5 69 43 68 18 10 6A 68 8F 35 2F C2 A7 56 4D 23 86 A2 55 1A 5B E7 3A F0 66 A1 0D 0D 34 AD 2B AE 89 84 87 1B F6 9C 28 4D 8C 09
65 E3 B3 DD C9 29 72 87 BF 6B 40 C2 75 90 6B E3 F5 59 3F C8 ED 09 00 4D 44 77 26 5D 10 E6 C1 86 01 47 5B 44 72 7D 6D 06 10 83 EF B6 80 00 D0 7A 6F 9C 05 32 1A
CE D7 D6 F5 DA 48 9B 5E 5B 5D 9B F5 C2 C7 AD 0A 41 6B DA D0 8D 84 51 3E FA E8 16 5A 16 95 7C 16 37 81 0F 9B 83 08 A1 77 D6 56 6F DE A0 CE AB B3 81 0F 1A
10 49 0D BF 41 23 50 05 1A CB A0 64 67 22 C3 03 F7 E3 26 AC E0 2D D0 C4 6A F9 C4 E3 E4 2E B2 D5 47 50 37 0C 1D 41 2A E8 3B 15 1D 3A C7 93 A7 FD 42 D3 C0 F4
1A 2C 3B 0F C5 80 C4 9D 1F 50 A2 14 A8 3E 7C D5 D9 E8 35 DB 18 A2 43 DD E9 DD 1D 9C F5 E3 21 3C 0E 0A 3D 07 45 D1 B0 DA 9D 3C CB 90 B2 07 80 E1 58 46 ED
7E B5 7F BC D1 6C 5B F8 68 F4 E0 CB BA 69 9A F9 25 B5 AA E6 73 E9 2F 0E 8F B6 1D A7 79 78 10 96 62 0A 23 CD CC 06 96 93 18 45 50 48 32 B6 D4 DD EF AC F7 D6
D7 1F DC 17 38 4A 96 99 A0 22 84 AB 61 C6 FC FA 70 03 2D EE 2B C1 B0 10 94 A9 D9 11 8B 09 A4 10 0B 57 AB 82 B0 D4 3D B8 77 AF 4B 91 02 19 24 49 17 50 55 63 78
ED A0 4B 2A 31 05 E9 7B 0D 27 56 A4 64 64 04 62 84 05 99 11 48 4A 51 18 99 E2 90 15 08 81 16 99 A0 20 30 94 22 04 99 59 0F 8E 11 9C 84 4C 26 15 45 21 19 99 91 25 4E
64 07 60 99 C1 8B 41 26 18 C4 AB F3 4E 08 AC 28 72 94 2B 9C 28 8E 5E 64 59 BC 28 49 12 5E 65 CF AB FF 07 51 E7 8C F0 F3 FC 19 43 00 00 00 00 **49 45 4E 44 AE 42 60 82**

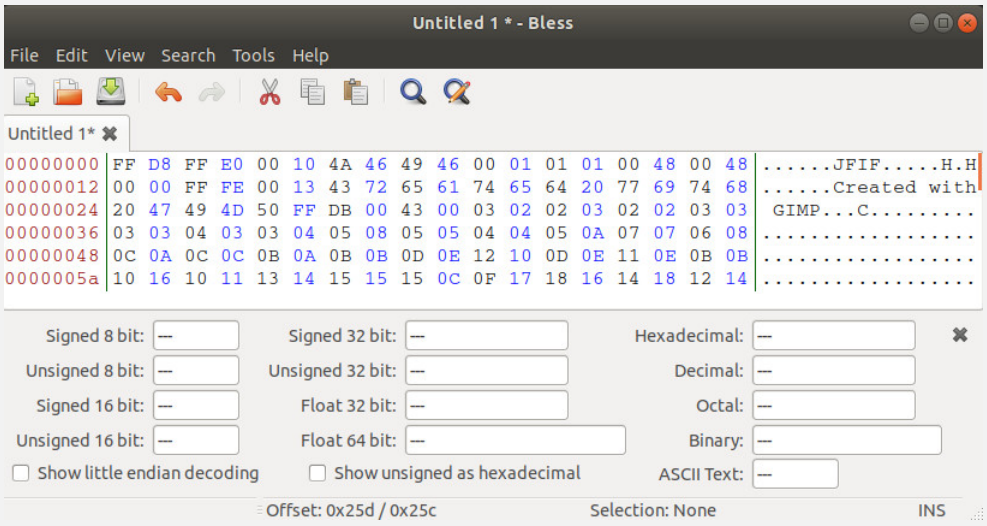
So the signature of PNG images is 89 50 4E 47 0D 0A 1A 0A. Some formats, like PNG as well, have also a fixed array of bytes at the end that represents the "trailer" (In the case of PNG is 49 45 4E 44 AE 42 60 82). The signatures of some famous formats can be found in the table below:

Format	Signature	Trailer	Format	Signature	Trailer
JPG	FF D8	FF D9	PNG	89 50 4E 47 0D 0A 1A 0A	49 45 4E 44 AE 42 60 82
MP3	49 44 33		GIF87	47 49 46 38 37 61	00 3B
MIDI	4D 54 68 64		GIF89	47 49 46 38 39 61	00 3B
SWF	43 57 53		PDF	25 50 44 46	0A 25 25 45 4F 46
ZIP	50 4B 03 04	50 4B	JAR	4A 41 52 43 53 00	
DOC	DB A5 2D 00		RAR	52 61 72 21 1A 07 01 00	
VLC	1F 8B 08		MPEG	00 00 01 B	00 00 01 B7

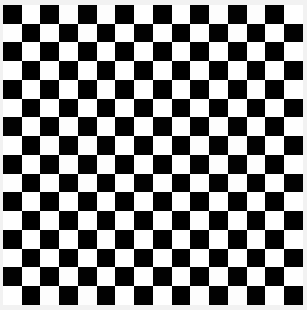
Challenge 20.1 — Black And White. What does this file represent?
FF D8 FF E0 00 10 4A 46 49 46 00 01 01 01 00 48 00 48 00 00 FF FE 00 13 43 72 65 61 74 65 64 20 77 69 74 68 20 47 49 4D 50 FF DB 00 43 00 03 02 02 03 02 02 03 03 03 03 04 03 03 04 05 08 05 05 04 04 05 0A 07 07 06 08 0C 0A 0C 0C 0B 0A 0B 0B 0D 0E 12 10 0D 0E 11 0E 0B 0B 10 16 10 11 13 14 15 15 15 0C 0F 17 18 16 14 18 12 14 15 14 FF DB 00 43 01 03 04 04 05 04 05 09 05 05 09 14 0D 0B 0D 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 FF C2 00 11 08 00 10 00 10 03 01 11 00 02 11 01 03 11 01 FF C4 00 15 00 01 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 FF C4 00 14 01 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF DA 00 0C 03 01 00 02 10 03 10 00 00 01 80 03 FF C4 00 15 10 01 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 33 FF DA 00 08 01 01 00 01 05 02 9A 69 A6 FF C4 00 14 11 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 20 FF DA 00 08 01 03 01 01 3F 01 1F FF

C4 00 14 11 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 20 FF DA 00 08 01 02 01 01 3F 01 1F FF C4 00 1C 10 00 02 01 05 01 00 00 00 00 00 00 00 00 00 00 00 11 12 13 24 42 52 62 82 92 FF DA 00 08 01 01 00 06 3F 02 C1 79 51 E4 04 D4 47 64 74 B8 2F 2A 3C 80 9A 88 EC 8E 97 05 E5 47 90 13 51 1D 91 D2 E0 BC A8 F2 02 6A 23 B2 3A 5F FF C4 00 14 10 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 20 FF DA 00 08 01 01 00 01 3F 21 04 0C A4 0C A4 0C A4 0C BF FF DA 00 0C 03 01 00 02 00 03 00 00 00 10 92 4F FF C4 00 14 11 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 20 FF DA 00 08 01 03 01 01 3F 10 1F FF C4 00 14 11 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 20 FF DA 00 08 01 02 01 01 3F 10 1F FF C4 00 14 10 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 20 FF DA 00 08 01 01 00 01 3F 10 18 EA 98 EA 98 EA 98 EA 9F FF D9 ■

Solution 20.1 — Black And White. According to the signature and the trailer, the file represent a jpg image. In order to reconstruct it it is necessary to open a bytes editor and paste the bytes there. In Linux a good and very simple byte editor is Bless:



Saving the file as with an arbitrary name with JPG extension, it reveals the image:



To fit the challenge the image is only 32x32 pixels. Here is zoomed 7 times.

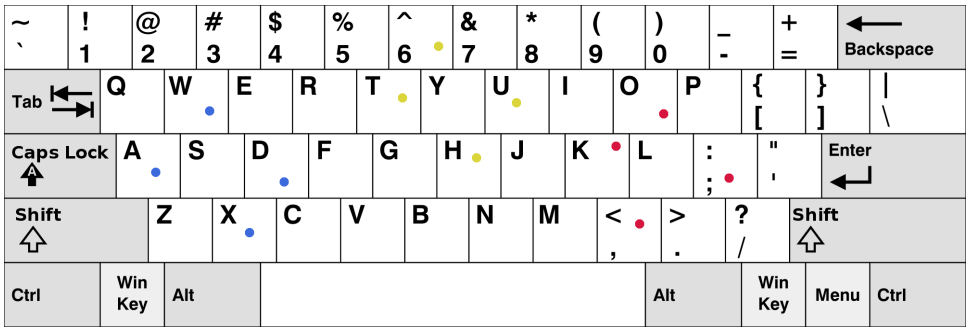
20.2 Keyboard

Keyboards are one of the most common input devices for a computer: in laptops are usually integrated and in desktop computers are usually USB plugged. There exists a lot of different keyboard layouts differing due to the necessity to cover all languages and cultures. As can be imagined, different layouts means different positions of the keys and, sometimes, different symbols to digit.

20.2.1 Keyboard layouts steganography

20.2.2 Around the target

A common way to cover messages using the keyboard as a steganography device is to represent its keys in some unconventional way. Instead of simply digit the letter we want, we can, for example, press the keys that circumscribe it to logically point that specific letter. Is in this way that key 'd' can be hide using a composition of keys 's', 'e', 'r', 'f', 'c' and 'x'. For example, in QWERTY layout, "fesc" can cover 'd' and "pil9" can hide 'o'. Usually a combination of 4 keys is enough to circumscribe a target key.



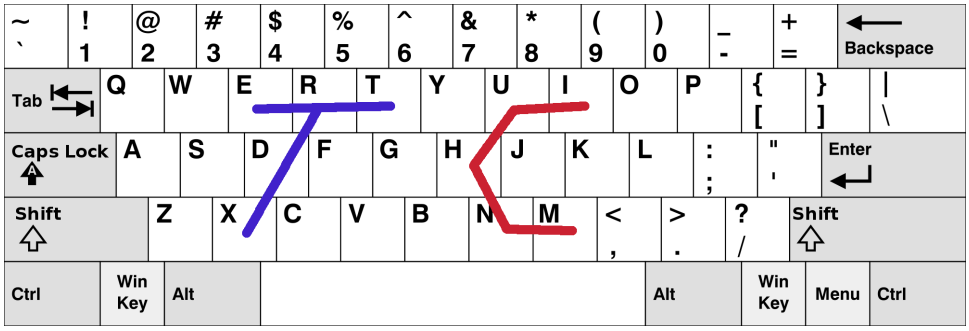
Moving on with examples the above image represent in blue the key sequence "awdx" representing the hidden key 's', in yellow the key sequence "t6uh" representing the hidden key 'y' and in red the sequence "ko;," representing the letter 'l'.

Challenge 20.2 — The name of my love. w3rd waxd gyr5 r4wd fest ■

Solution 20.2 — The name of my love. ester

20.2.3 Follow the stream

Thinking unconventional another good technique can be used to improve the previous system. The main flaw of the previous approach is that each key can be represent with at most 6 other keys that never change in the same keyboard layout. This means that 'd' can only be represented by letters "serfcx" and this is a huge limitation that makes the technique difficult to reuse more than one time. If it is true that 'd' can be represented the first time by "secf" and a second time by "recx", for long message all combinations can be used making the ciphertext not strong enough if known plaintext attack is performed.
Instead of "point" a key, we can "draw" the symbol, represented by the key, on the keyboard. For example the key sequence "iuhmn" can hide the letter 'c' and the sequence "ertdx" can hide the letter 't'



Challenge 20.3 — Fruits. zsedc aw3edr5 ki8uh -plo9ij cftgb 098uiokjh ■

Solution 20.3 — Fruits. ananas

20.2.4 Double meaning

Looking close to the keyboard it can be noticed that some of the keyboard keys hosts different symbols. It is, for example, the case of the digits: digit 1 hosts also symbol !, digit 2 hosts symbol @ and so on. This can be used as a fancy and easy steganography technique. It is infact possible to represent letters as digits and then hide them into keyboard respective symbols. For example we can encode, using A1Z26, the string "keyboard" in 11 5 25 2 15 1 18 4 and then replace single digits with symbols obtaining:

!! % @% @ !% ! !* $

21. Substitution Steganography

21.1 Polygraphia (1518)

Johannes Trithemius, in one of its major operas "Polygraphia", published, for the first time more than 1000 different alphabets (real and fictional), divided in 5 volumes.

21.1.1 Theban Alphabet

One of the most famouse alphabet is the Theban alphabet, that Trithemius attributes to Honorius of Thebe, supposed to be a mythical character. It is basically a writing system with a 1-to-1 mapping with the ancient Latin alphabet (modern letters like J, U and W are not mapped)

Challenge 21.1 — Shakespeare. Short citation

Solution 21.1 — Shakespeare. Frailty, thy name is woman

21.2 Bacon's Cipher (1605)

Bacon's cipher, known also as Baconian cipher, is a steganography technique to hide secret messages, invented by Sir Francis Bacon in 1605. It is a classical substitution cipher where each symbol is replaced by a sequence of 5 symbols coming from an arbitrary vocabulary of size 2. The following encoding table is referred to the classical vocabulary composed by a_s and b_s:

a	aaaaa	g	aabba	n	abbaa	t	baaba
b	aaaab	h	aabbb	o	abbab	u/v	baabb
c	aaaba	i/j	abaaa	p	abbba	w	babaa
d	aaabb	k	abaab	q	abbbb	x	babab
e	aabaa	l	ababa	r	baaaa	y	babba
f	aabab	m	ababb	s	baaab	z	babbb

It is easy to see that using the vocabulary V={0,1} the message will be hidden in a binary form. Different vocabulary can be used and sometimes the letter i, j, u and v are encoded separately each other:

a	aaaaa	h	aabbb	o	abbba	v	babab
b	aaaab	i	abaaa	p	abbbb	w	babba
c	aaaba	j	abaab	q	baaaa	x	babbb
d	aaabb	k	ababa	r	baaab	y	bbaaa
e	aabaa	l	ababb	s	baaba	x	bbaab
f	aabab	m	abbaa	t	baabb		
g	aabba	n	abbab	u	babaa		

To improve the difficulty of the challenges, the encoding procedure can be modified so that not strict binary vocabularies are used, but generic binary concepts.

Challenge 21.2 — Gnam Gnam. baabaaabbbaabaa aababababaaabaaaabaaabbaaba baabaabaaaababbaabaa abaaa baaaaaabaaaaaaaaaabb baabaaabbbaabaa abbaaaaaaaababbaabaa, ababbaaaaaabaabaabaabaaab ababbaabaa aabbbbaabbabbaaaabbabaaaababba

Solution 21.2 — Gnam Gnam. the first time I read the name, makes me hungry

Challenge 21.3 — I am famous. 01100000001100000000100100 0000100000000100111001101 10110100010111010011001001001 0 1001001110011000010 0 01111010110000011000100 10 000000011100100 01111010110000011000100 10 1011000111010000001000111 101100010010001001 00 0000010011100111000101000000011010010011001000 0011 1001101110 1001000111000000101000100100100111100100000001000100100?

Solution 21.3 — I am famous. maybe Bacon wrotes some plays ahe plays which were attributed to Shakespeare?

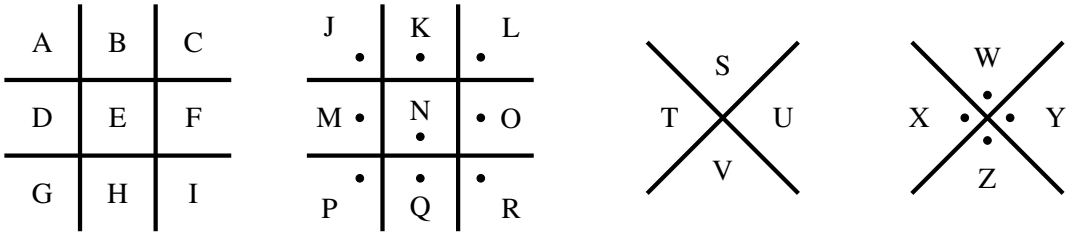
Challenge 21.4 — 5 pm. yes Please buT Be SuRe it is NOT too hoT bEcAusE i like to tasTe It WithOUT BuRNIng mY pALatE And My tongue ■

Solution 21.4 — 5 pm. can I have a cup of tea?

21.3 Pigpen Cipher (18th Century)

The pigpen cipher is a substitution technique to hide information used mostly by Masons. Called also masonic cipher or even Freemason's cipher, is supposed to be a variant of the more ancient Rosicrucian cipher, first introduced by Cornelius Agrippa in 1531 and inspired by Kabbalistic tradition. The earlier usage of this cipher has been found on a tomb in the Trinity Church of New York, owned by a man died in 18th century. The inscription says:

It can be decoded as "*Thomas Brierley made his ingress July 16th 1785*", which is supposed to refer to the moment in which the men joined the order. More formally the hiding technique consists in replacing original symbols with the enclosing borders according to following schema:



So that A becomes ⌟, O becomes ⊏, U becomes <, Z becomes ^ and so on.

Challenge 21.5 — Bacon Everywhere. ⌜ ⊐⊏ ⊡⊏> ⊔⊡⊏∨ ∨⊓< ⊔<>

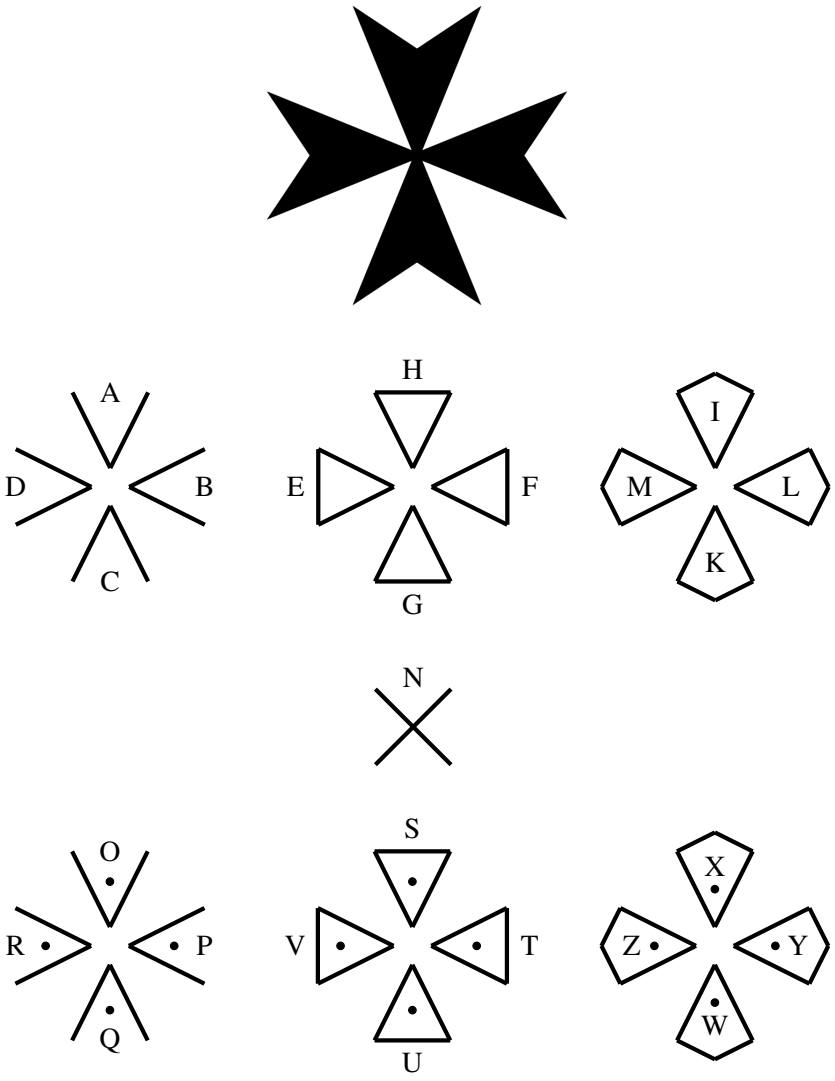
⌞⌜<¬>⊏¬⌜ ⌟¬⊓< ⊓⌟∨ ∨⊏⊐⊡ ∨∨⊡⊡> ⌞⊏⊡⊡⊡⌞>⌜⊏⊡∨

∨⌜>⊓ ⊏⊏⊏⊐ ■

Solution 21.5 — Bacon Everywhere. I do not know why, but cryptography has some sweet connections with food.

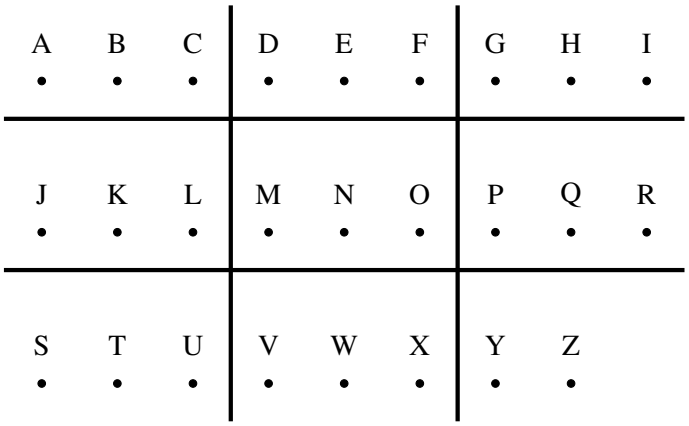
21.3.1 Templar Cipher

To the Pigpen cipher is usually linked the Templar cipher. This fictional variant has no documentations in history, but is supposed to have been used by masonic knights. It follows the geometries of the Maltese Cross, the symbol of the cavalry order.



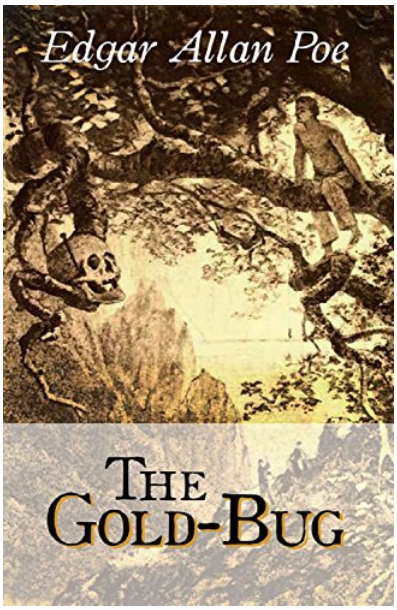
21.3.2 Rosicrucian Cipher

Another variant of the pigpen cipher has been used by the religious group called Rosicrucian. The main concept is the same but in this case we have less geometries and symbols are uniquely identified by the distance of the dots with respect to the lines:



21.4 The Gold-Bug (1843)

In 1843, Edgar Allan Poe published probably its most remunerative opera: "The Gold-Bug". The story is about gold beetle, islands, Capitan Kidd treasure, maps, skulls and, of course, a cryptogram:



53‡‡†305))6*;4826)4‡.)4‡);806*;48†8
¶60))85;;]8*;:‡*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;‡?;

Poe provided also the solution of the cryptogram, that came out to be a simple substitution steganography technique:

"A good glass in the bishop's hostel in the devil's seat
twenty-one
degrees and thirteen minutes northeast and by north
main branch seventh limb east side
shoot from the left eye of the death's-head
a bee line from the tree through the shot fifty feet out."

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
5	2	-	†	8	1	3	4	6	,	7	0	9	*	‡	.	$	(	)	;	?	¶	]	ć	:	[

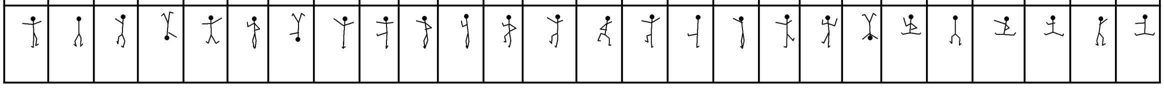
Challenge 21.6 — Intro.
95*::85()53‡6−‡*;(5−;8†5*6*;695−:]6;459(]600659083(5*†48]5)‡15*5*−68*;4?3?8*‡;15960:5*†45†‡*−8288*]850;4:2?;5)8(68)‡196)1‡(;?*8)45†(8†?−8†469;‡]5*;;‡5¶‡6†;489‡(;616−5;6‡*−‡*)8$?8*;?.‡*46)†6)5);8()48081;*8]‡(085*);48−6;:‡146)1‡(815;48()5*†;‡‡7?.46)(8)6†8*−85;)?006¶5*)6)05*†*85(−45(08);‡*)‡?;4−5(‡06*5 ■

Solution 21.6 — Intro.
Many years ago, I contracted an intimacy with a Mr. William Legrand. He was of an ancient Huguenot family, and had once been wealthy; but a series of misfortunes had reduced him to want. To avoid the mortification consequent upon his disasters, he left New Orleans, the city of his forefathers, and took up his residence at Sullivan's Island, near Charleston, South Carolina.

21.5 The Dancing Man Code (1903)

The Adventure of the Dancing Man is a Sherlock Holmes story written by Arthur Conan Doyle. In this novel Holmes receive strange letters with a sequence of sticky figures that seems to be dancing men. After some clues he realize that this is nothing else than a substitution cipher and cracks it by frequency analysis. Unfortunately this is one of the rare story in which the client of Holmes dies, but it leaves to us this nice cipher:

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



Challenge 21.7 — The Dancing Man. 

Solution 21.7 — The Dancing Man. No one can compete with his incredible intelligence

21.6 SMS (1993)

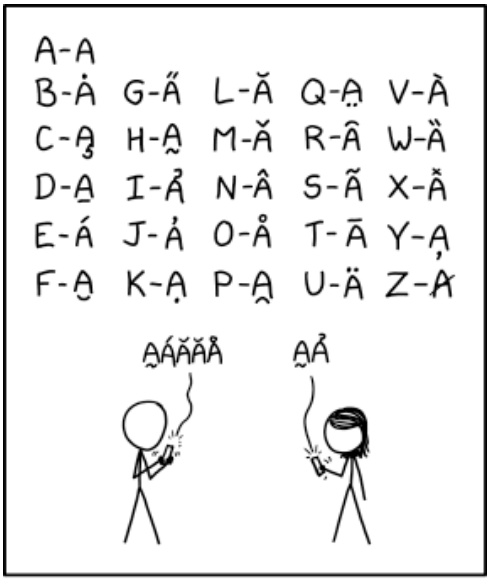
In 1973 the first mobile phone was proposed to the market and in 1993 the first SMS message was officially sent. To write messages, according to the keypad layout of the time (that most commonly followed the E.161 recommendation), specific buttons need to be pressed from 1 to 4 times in order to produce a symbol. For example to produce the letter 'i', the numerical button '4' was pressed three times, for the letter 's' the button '7' was pressed four times and the space was obtained by pressing the '0' one time. A simple substitution technique to hide information can be based on this logic: if to write the symbol 'i' I need to press the button '4' for three times, the symbol 'i' will be encoded with 444, the symbol 's' will be 7777, the space will be 0, and so on:

Challenge 21.8 — Nokia 3310. 4666666306665553084446337777

Solution 21.8 — Nokia 3310. good old times

21.7 Scream Cipher

The *Scream Cipher* is a monoalphabetic substitution cipher proposed by `xkcd` (Randall Munroe) [1] In this cipher, each letter of the Latin alphabet (A-Z) is replaced by the letter "A" bearing a unique diacritical mark. The idea is humorous but functional: an entire message can be encoded using only variants of the letter *A*, visually resembling the way human screams are often portrayed in text.



IN THE *SCREAM CIPHER*, MESSAGES CONSIST OF ALL As, WITH DIFFERENT LETTERS DISTINGUISHED USING DIACRITICS.

Encryption under the Scream Cipher relies on a fixed one-to-one correspondence between the 26 letters and 26 distinct diacritical variants of "A". Let the mapping function be

$$E : \{A, B, \ldots, Z\} \longrightarrow \{\text{A with diacritics}\}.$$

To encrypt a plaintext message, substitute each plaintext letter with its corresponding stylized "A'". Unlike classical substitution ciphers (e.g. Caesar or Atbash), the Scream Cipher does not attempt to obscure patterns, it merely replaces each symbol with a visually similar one. From a cryptographic perspective, this is simply a re-labeled alphabet. As a result, classical frequency analysis completely breaks the cipher; its purpose is aesthetic and playful rather than secure.

[1] See `xkcd` comic #3054.

22. Colors Steganography

This is a very introductory section about colors, different types of models to represent them, and how this models can hide information.

22.1 RGB Color Model

The RGB color model is based on addictive mixture of three monochromatic lights: red, green and blue. These three colors have the ability to create all possible colors by combining them in different ways. The model, to be displayable, needs to be encoded and interpreted. This is done by building a tuple of three integers (from 0 to 255) representing the information about red, green and blue quantity, as shown in the following schema:

R:	255	0	0	255	57	12	125	0	255
G:	0	255	0	255	78	28	151	0	255
B:	0	0	255	0	29	80	76	0	255

22.2 CMYK Color Model

This is a subtractive model using secondary colors to mixed all others: cyan, magenta and yellow. To these colors a fourth one, called key color, is added: the black. This time quantity of each color is represented by percentage (from 0 to 100) as follows:

C:	100	0	0	0	27	85	17	0	100
M:	0	100	0	0	0	65	0	100	0
Y:	0	0	100	0	63	0	50	100	100
K:	0	0	0	100	69	69	81	0	0

22.3 HSL Color Model

HSL is a subset of RGB color model that express the colors in terms of hue, saturation and luminosity. Hue represents the pure color, the saturation represents the intensity of the hue, and the luminosity represent how much white and black is added to hue (if hue will be lighter or darker). As usual we have the color representation (all values can be from 0 to 255. Sometimes for values of saturation and luminosity, values can be expressed in percentage 0-100 of the value itself):

H:	0	85	170	60	86	226	81	170	170
S:	255	255	255	255	117	188	84	0	0
L:	128	128	128	128	54	46	113	0	255

22.4 HEX Color Model

HEX is an extension of RGB color model, using hexadecimal numbers to define colors. For example the RGB of color red (255,0,0) is expressed in HEX with #FF0000, transforming the value 255 in its equivalent hexadecimal FF and concatenating components without commas:

R:	FF	0	0	FF	39	0C	7D	0	FF
G:	0	FF	0	FF	4E	1C	97	0	FF
B:	0	0	FF	0	1D	50	4C	0	FF

How can these color models hide some messages? Consider the following set of colors:

Hex:	#00636F	#6C6F72	#656400

Removing trailing and leading zeros we obtain the following sequence of hex values: 63 6F 6C 6F 72 65 64 that, following the ASCII encoding, represent the sequence of characters: c o l o r e d, the hide information. Another way to hide secrets inside sequence of colors is to consider only the changing symbols at specific positions:

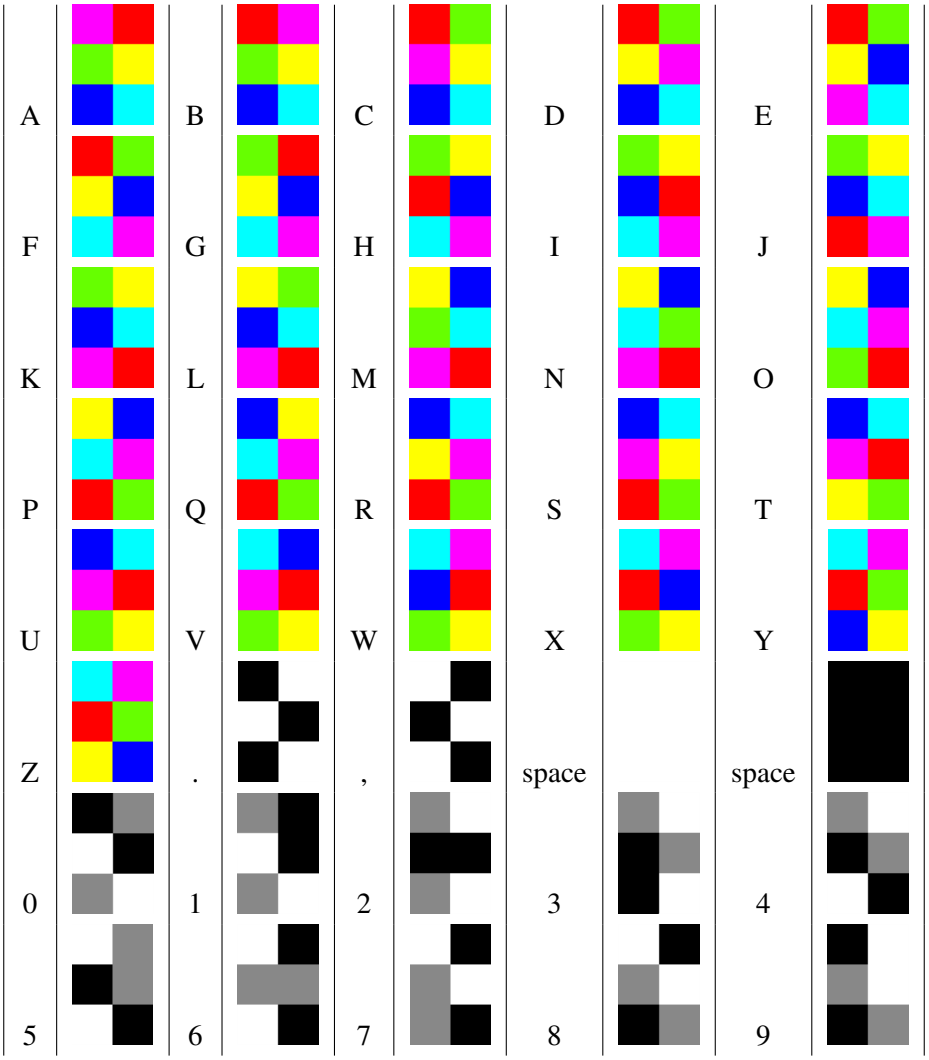
Hex:	#143263	#14326F	#14326C	#14326F	#143272	#143269
			#14327A	#143265		

Each color starts with the same pattern (1432xx), the last symbols differs. From those we can construct the following hex sequence: 63 6F 6C 6F 72 69 7A 65 that, again, represents c o l o r i z e using the ASCII encoding. Of course the same principle can be adapted to the other color models.

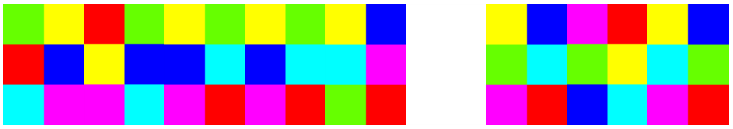
22.5 Hexahue

Colors are even used as esoteric programming languages (like PIET), and in general they are a fancy way to hide information. Josh Cramer invented Hexahue with the objective to create a unique alphabet that could substitute standard English writing using a combination of colors that were easily able to be distinguished.

The colors are 9 in total: red, green, blue, yellow, cyan, purple, black, mid-gray, and white. The first six are used to represent letters and the last 3 to represent digits and punctuation. Every letter is substituted by a grid of 2x3 squares forming a pattern of 6 colors and every digit is substituted by a grid of 2x3 squares forming a pattern of 3 colors. The following is the translation table:



just for example, the phrase "hello man" is written as:



23. Images Steganography

TODO good intro...

23.1 Image channels

As all probably know, digital images are composed of pixels which are combinations of primary colors. A channel is the image produced by only one of these primary colors. In the example of a RGB images we have three primary colors: red, green and blue, and so we will have three primary channels. On the other hand a grayscale image will have only one channel and a CMYK image will have four channel (cyan, magenta, yellow and black).

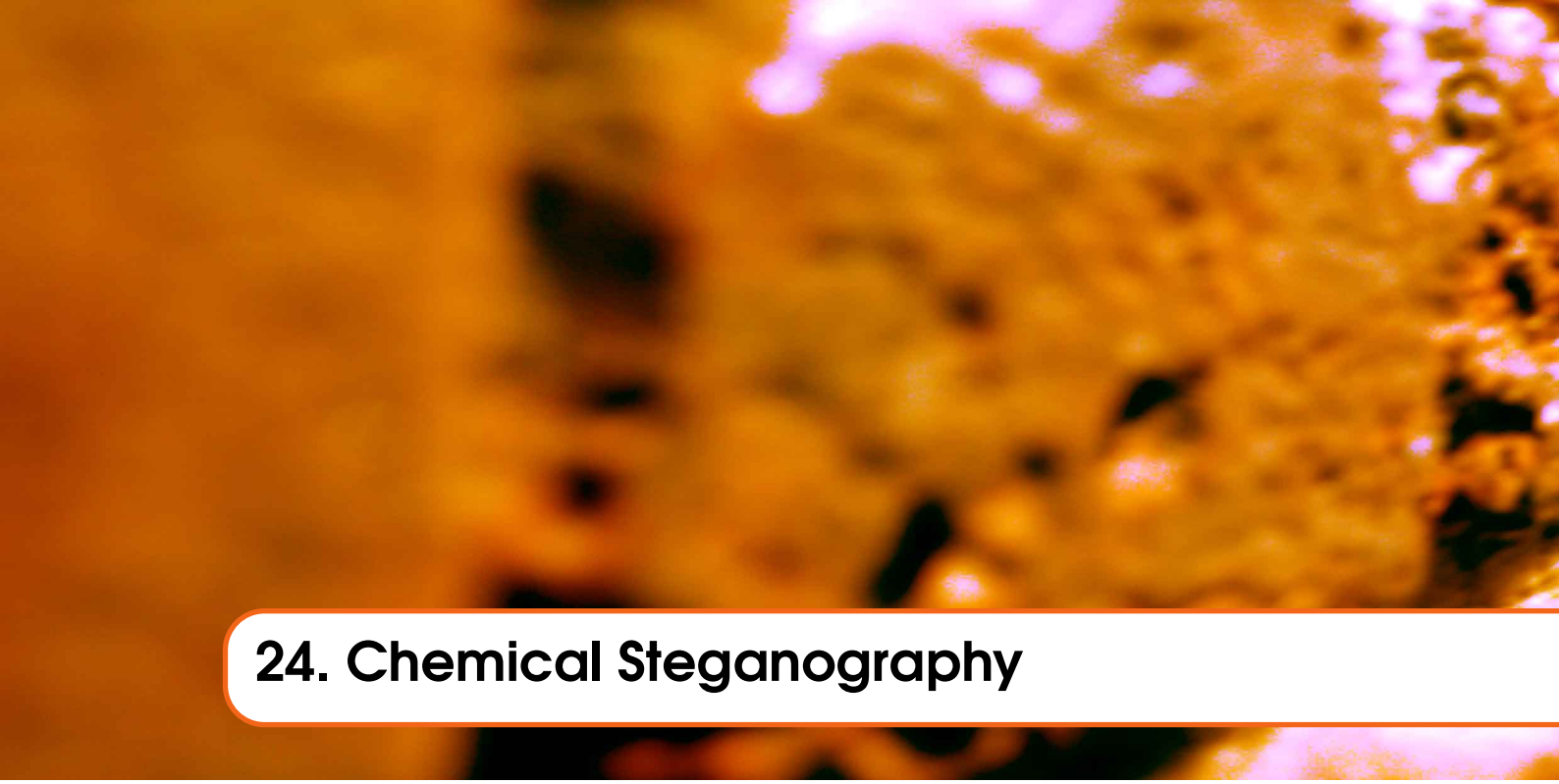
23.2 Image files

A very clever way to hide entire files in images is to exploit some of the common characteristics. Above all the fact that files and pixels are basically encoded in a very similar way: we can see files as a sequence of hex values (as explained in previous sections) and images as a sequence of pixels, each of them encoded with a hex value. In this way is possible to "convert" every type of digital document into a sequence of pixels building an image. Let's make an example: suppose to have the following image that we want to hide:

We know that it is possible to see the file image as a sequence of bytes. In our case the first bytes of our image are:

89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52 00 00 01 F4 00 00 01 F4 08 06 00 00 00 CB D6 DF 8A 00 00 1C A9 7A 54 58 74 52 61 77 20 70 72 6F 66 69 6C 65 20 74 79 70 65 20 65 78 69 66 00 00 78 DA A5 9B 59 76 5C B7 92 45 FF 31 8A 1A 02 FA 00 86 83 76 AD 37 83 1A 7E ED 03 A6 24 4B 6E 9E ED 22 4D 26 95 CD BD 00 22 E2 34 01 D8 9D FF FD CF 75 FF C3 97 F9 5C 5D 2E D6 6A AF D5 F3 95 7B EE 71 F0 47 F3 5F 5F E3 FD 0E 3E BF DF EF EB E6 CF 5F E1 E7 E7 DD F7 17 22 8F 89 C7 F4 F5 42 AB 9F 8B 1D 9E 8F BC 3F 7E 9E 5F 5F 8F 61 F0 7C F9 CD 85 FA F9 BC 30 7F 7E 61 7C 2E 14 DB E7 06 9F E7 BF DD 28 85 AF 1B F8 FD B9 D0 F8 5C 28 C5 CF 9D F3 D7 BF E7 E7 CE B5 37 FB ED 14 3E 9F D3 F4 C3 E7 F1 FD 38 FD CA C9 62 2D 35 58 E6 77 8E DE AC 76 FE 6E D1 67 63 DD B6 06 7A 57 EC FA 5C 99 9F B5 FA E5 DF EE DB 5B 23 63 8A 27 85 E4 DF EF FC 35 CA A4 9F 90 C6 7B 1C FC 44 DE C7 AB FC 9D 53 E1 77 49 F1 AD AF 77 84 8C 21 30 F2 FE 59 DB CF 54 B5 9A 3F AD CD B7 C7 3F F9 72 7F 67 5A 9F 74 F8 29 DC DF FF FA A4 81 FB DD 0B BF A4 41 3D 9F E7 D3 2F D1 AB DF 1F DF ...

now we can start "convert" triples of bytes into pixels: #89504E, #470D0A, #1A0A00, #00000D, #494844 and so on. Putting the pixels together, and arranging them in a rectangular way, we obtain the following image:



24. Chemical Steganography

24.1 The Elements

All matter is made up of atoms, and all atoms are composed of three main subatomic particles: protons, neutrons, and electrons. Protons carry a positive charge, neutrons have no charge, and electrons are negatively charged. The negative charge of one electron balances the positive charge of one proton. Both protons and neutrons have a relative mass of 1 atomic mass unit (amu), while electrons have almost negligible mass in comparison.

The element hydrogen has the simplest atoms, each containing just one proton and one electron. The proton forms the nucleus, while the electron orbits around it. All other elements have neutrons as well as protons, which help to hold the nucleus together through the strong nuclear force.

Electrons orbiting around the nucleus of an atom are arranged in energy levels or **shells**. The first shell can hold only two electrons, while the second shell can hold up to eight. Subsequent shells can hold more electrons, but the outermost shell of any atom - known as the **valence shell** - holds no more than eight electrons. The following table summarizes the number of electrons in each shell for some common elements:

Element	Atomic Number	1st Shell	2nd Shell	3rd Shell
Hydrogen (H)	1	1	0	0
Helium (He)	2	2	0	0
Lithium (Li)	3	2	1	0
Carbon (C)	6	2	4	0
Oxygen (O)	8	2	6	0
Neon (Ne)	10	2	8	0
Sodium (Na)	11	2	8	1

The different compositions and arrangements of electrons allow each element to be represented by a unique **electron configuration**. This notation describes how electrons are distributed among various shells and subshells. For example:

Hydrogen: $1s^1$ Helium: $1s^2$ Neon: $1s^2 2s^2 2p^6$

The electron configuration determines how atoms interact and bond with one another to form molecules and compounds. Elements with full outer shells, such as the noble gases, are chemically inert, while those with incomplete outer shells, such as sodium or oxygen, readily form bonds to achieve stability. Thus, the arrangement of electrons not only defines the identity of an element but also its chemical behavior and its position in the periodic table.

24.1.1 Periodic Table

For steganographical purposes chemical elements can be used in order to hide a secret. The procedure performs two successive mappings:

$$\text{Text} \xrightarrow{\text{symbolic substitution}} \text{Element Symbols} \xrightarrow{\text{atomic lookup}} \text{Atomic Numbers}.$$

To any observer, the final message appears to be a string of chemical or numeric data, though it encodes readable text.

Encoding Procedure

Given a plaintext message $M = (m_1, m_2, \ldots, m_n)$:

1. **Segmentation:** Divide the text into chunks corresponding exactly to valid element symbols (1-2 letters, the first uppercase, optional second lowercase). For instance, CARBON can be segmented as:

   C-Ar-B-O-N.

2. **Numeric Encoding:** The encoded form of C-Ar-B-O-N becomes:

   $(6, 18, 5, 8, 7)$.

Decoding Procedure

To recover the message, apply the inverse mapping:

$$(6,18,5,8,7) \xrightarrow{f^{-1}} (C,Ar,B,O,N) \xrightarrow{\text{concatenate}} \text{CArBON}.$$

Remarks

This example demonstrates a perfect symbolic correspondence between text and element symbols, ensuring that both the intermediate (chemical) and final (numeric) forms are syntactically valid. The resulting numeric code may then be disguised in tabulated data, experimental parameters, or other seemingly scientific contexts, completing the steganographic concealment.

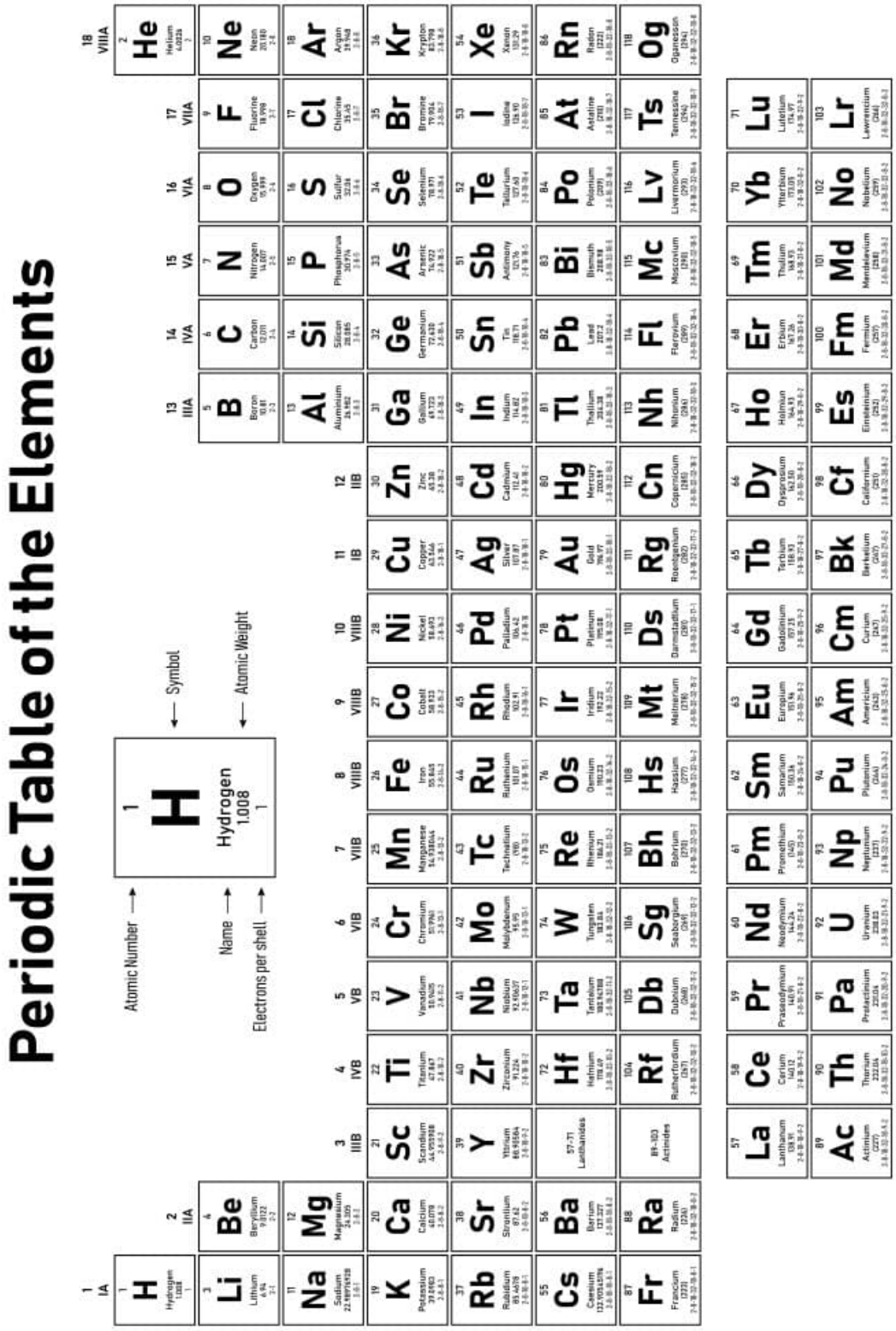


Figure 24.1: Periodic Table of Elements

24.2 Emission Spectrum

The emission spectrum of a chemical element or chemical compost is the set of frequencies of the electromagnetic radiation emitted by its electrons once they move to a lower energy state starting from an higher energy state. For example, if we excite an element by heating, it will produce some radiations that are the result of the vibrations of its atoms. If we pass these radiations through a prism they will be dispersed based on their wavelength. Projecting these radiations on a screen it is possible to obtain the full emission spectrum of the analyzed element. It is important to know that the emission spectrum of each element is unique and so its analysis, called spectroscopy, can be used to retrieve the composition of an unknown material. The following are some examples of emission spectrum of some basic elements.

Figure 24.2: Helium

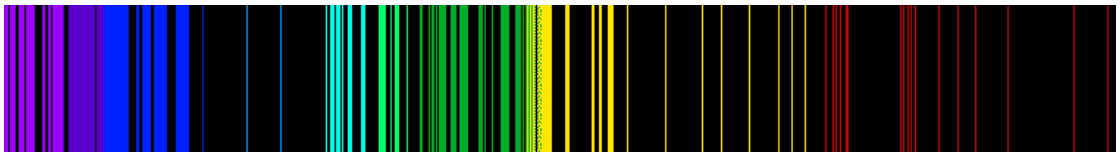


Figure 24.3: Iron

Figure 24.4: Magnesioum

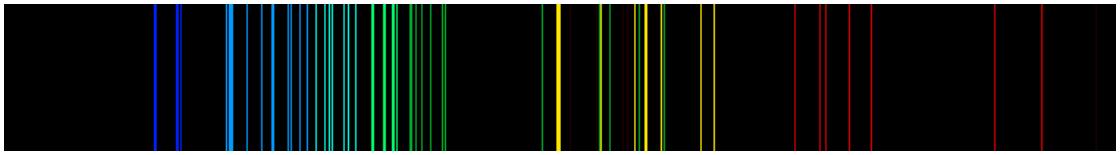
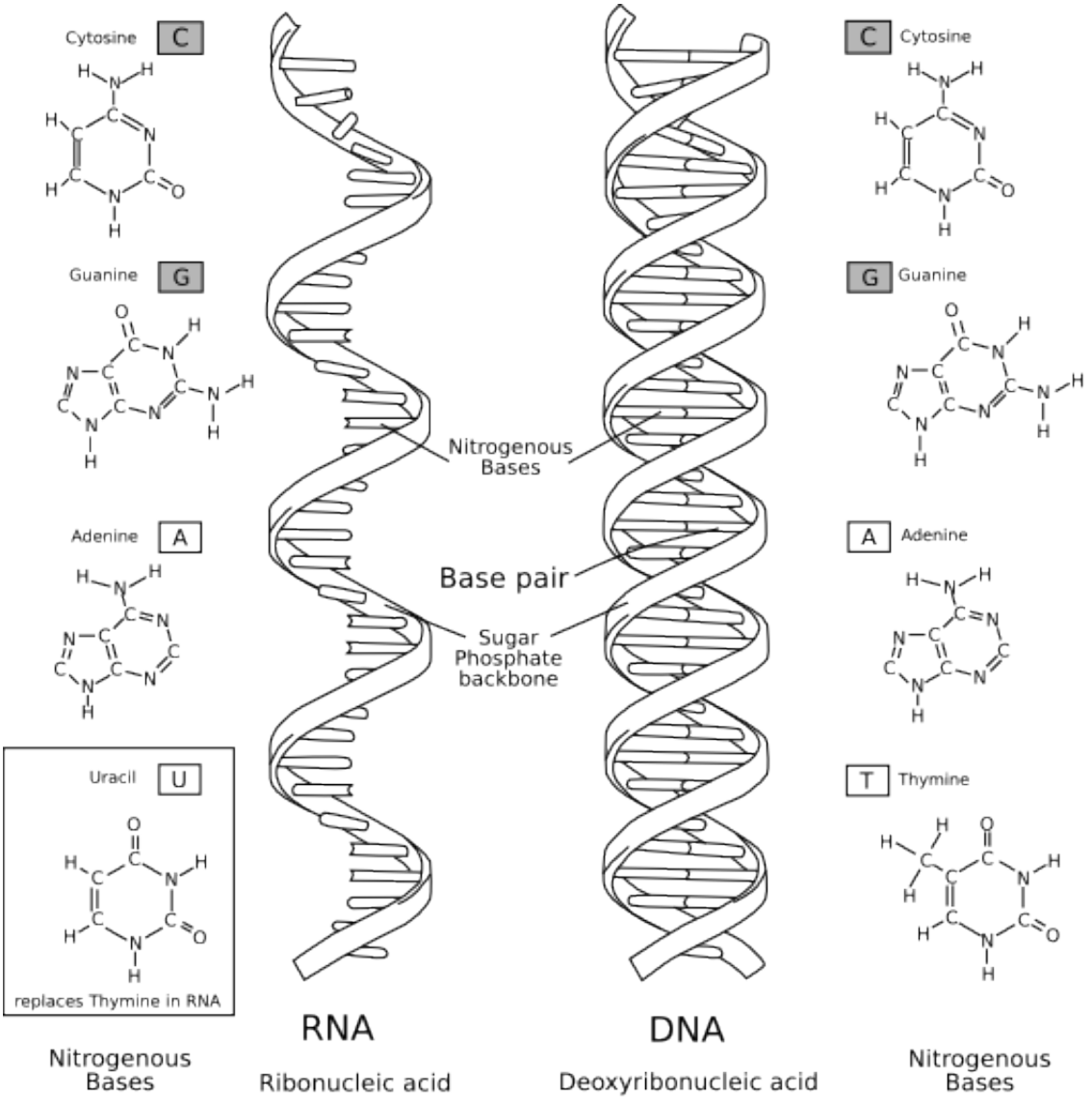


Figure 24.5: Nickel

The use of emission spectrum as a steganography technique is really easy. Each spectrum encodes a single element in the periodic table. We can for example take the spectrum of Chlorine, Oxygen and Selenium and, combining with the previous steganography technique, getting back the world "CLOSE" by reading their symbols.

24.3 Codons

When DNA was discovered in 1953, the immediate next goal was to understand how proteins were encoded. It in the following years, in 1961, codons, the basis of genetic code, saw the light in genetic theories. Codons were defined as triplets of DNA basis that, arranged in a specific order, are able to encode the basic known ammino acids.



Sometimes codons are used in steganography due to their capability to represent almost all letters in the English alphabet. Codons notations differs from DNA and RNA (Thymine is not present in RNA, and Uracil is not present in DNA). Steganography puzzles involving codons can be composed starting from ammino acid abbreviation or from one of its representing codons or compressed form. The following tables will summarize this concept:

Ammino acid	Abbreviation	DNA Codons	Compressed
Alanine	Ala/A	GCT, GCC, GCA, GCG	GCN
Cysteine	Cys/C	TGT, TGC	TGY
Aspartic acid	Asp/D	GAT, GAC	GAY
Glutamic acid	Glu/E	GAA, GAG	GAR
Phenylalanine	Phe/F	TTT, TTC	TTY
Glycine	Gly/G	GGT, GGC, GGA, GGG	GGN
Histidine	His/H	CAT, CAC	CAY
Isoleucine	Ile/I	ATU, ATC, ATA	ATH
Lysine	Lys/K	AAA, AAG	AAR
Leucine	Leu/L	TTA, TTG, CTT, CTC, CTA, CTG	YTR, CTN
Methionine	Met/M	ATG	ATG
Asparagine	Asn/N	AAT, AAC	AAY
Proline	Pro/P	CCT, CCC, CCA, CCG	CCN
Glutamine	Gln/Q	CAA, CAG	CAR
Arginine	Arg/R	CGT, CGC, CGA, CGG, AGA, AGG	CGN, AGR
Serine	Ser/S	TCT, TCC, TCA, TCG, AGT, AGC	AGY, TCN
Threonine	Thr/T	ACT, ACC, ACA, ACG	ACN
Valine	Val/V	GTT, GTC, GTA, GTG	GTN
Tryptophan	Trp/W	TGG	TGG
Tyrosine	Tyr/Y	TAT, TAC	TAY

Ammino acid	Abbreviation	RNA Codons	Compressed
Alanine	Ala/A	GCU, GCC, GCA, GCG	GCN
Cysteine	Cys/C	UGU, UGC	UGY
Aspartic acid	Asp/D	GAU, GAC	GAY
Glutamic acid	Glu/E	GAA, GAG	GAR
Phenylalanine	Phe/F	UUU, UUC	UUY
Glycine	Gly/G	GGU, GGC, GGA, GGG	GGN
Histidine	His/H	CAU, CAC	CAY
Isoleucine	Ile/I	AUU, AUC, AUA	AUH
Lysine	Lys/K	AAA, AAG	AAR
Leucine	Leu/L	UUA, UUG, CUU, CUC, CUA, CUG	YUR, CUN
Methionine	Met/M	AUG	AUG
Asparagine	Asn/N	AAU, AAC	AAY
Proline	Pro/P	CCU, CCC, CCA, CCG	CCN
Glutamine	Gln/Q	CAA, CAG	CAR
Arginine	Arg/R	CGU, CGC, CGA, CGG, AGA, AGG	CGN, AGR
Serine	Ser/S	UCU, UCC, UCA, UCG, AGU, AGC	UCN, AGY
Threonine	Thr/T	ACU, ACC, ACA, ACG	ACN
Valine	Val/V	GUU, GUC, GUA, GUG	GUN
Tryptophan	Trp/W	UGG	UGG
Tyrosine	Tyr/Y	UAU, UAC	UAY

Challenge 24.1 — Chemistry of love - pt1. GAC UAG CCC GCC AUG AUA AAC GAG AUA AGU ACC CAC UAG UGA GGG CAU ACG ACG UAG B GAG ACA CAU GAG CCA CUG GAG GCU AGC UGA AGG GAA UGU CAC GAA AUG AUA UGC GCG CUG CCU

CGA UAG GAU UGA UGU AUA AAC GGC GCG UUC GAG GAG CUA AUA AAU GGC UAG UUC B CUG AUU UCC AGC ■

Solution 24.1 — Chemistry of love - pt1. Dopamine is thought to be the pleasure chemical, producing a feeling of bliss

Challenge 24.2 — Chemistry of love - pt2. Glu Ser Thr Arg ? Gly Glu Asn Ala Asn Asp Thr Glu Ser Thr ? Ser Thr Glu Arg ? Asn Glu Pro Leu Ala Tyr Ala Arg ? Leu Glu Ile Asn Thr His Glu Ser Glu X Asp Arg Ile Val Glu Ala Arg Glu Ala ■

Solution 24.2 — Chemistry of love - pt2. estrogen and testosterone play a role in the sex drive area

Challenge 24.3 — Chemistry of love - pt3. TTA ATC GCA CTT GGG TAA TTA CTC GGG GTA GCA TAT TTG CTT TAT AGT AGA TAG TAC TAA GAG CGC TCC TGT ATC CGC CTA GCC CTC TTG CGT GAG TAA TTA CTT CGC TTA CTA GGA GCG ATC CTA ACT ACG CTC AGA TGT GTG CTC TCC CGC ACA TAG TTG CCG GTA CTC CGG GCA TGG CGA TTG CTG CTT X ACA TAT TGG CTT TAC CTT TTG TGG ■

Solution 24.3 — Chemistry of love - pt3. Norepinephrine is similar to adrenaline and produces the racing heart and excitement

VI

Sequences

25. Famous Sequences

Lots of riddles requires to find missing numbers, or number at a specific position, of given sequences. Unfortunately there is no a such kind of sequence solver that works generally for every possible existing, or not existing, sequence. Thus a knowledge on at least the most famous ones is required in order to approach to this type of challenges.

25.1 Fibonacci Numbers

Challenge 25.1 Find the missing numbers of the following sequence separating by comma: 0,1,1,2,3,5,8,?,21,34,?,89 ... ■

Fibonacci Numbers is one of the most popular and known sequence of integer numbers. The number at position n is simply the sum of the number at position $n-1$ with the number at position $n-2$ of the sequence, starting with 0 and 1 as the first two numbers of the sequence. More formally we have that:

$$Fib(n) = \begin{cases} 0 & \text{n=0} \\ 1 & \text{n=1} \\ Fib(n-1) + Fib(n-2) & \text{n>1} \end{cases} \tag{25.1}$$

Solving the challenge is really simple, in this case. We just need to sum together the 2 numbers before the question marks to obtain its value:

Solution 25.1 Simply enough:

- For the first '?' we have that 5 + 8 = 13
- For the second '?' we have that 21 + 34 = 55

Thus the solution is 13,55.

Sometimes challenges involving sequences requires to calculate the value at a specific position. Most of times sequences are infinite and calculating the nth value by recursively apply the self representing formula is unsustainable, neither using a powerful calculator:

Challenge 25.2 Find the 150th number of the following sequence:
0,1,1,2,3,5,8,13,21,34,55,89 ... ■

Fibonacci numbers have been well studied since its discovery, and now we are able to calculate the nth number of the sequence simply by applying the following formula:

$$Fib(n) = \frac{(1+\sqrt{5})^n - (1-\sqrt{5})^n}{2^n * \sqrt{5}} \tag{25.2}$$

Solution 25.2 Thus the solution at the previous challenge is:
$Fib(150) = \frac{(1+\sqrt{5})^{150} - (1-\sqrt{5})^{150}}{2^{150} * \sqrt{5}} = 9969216677189303386214405760200$

25.2 Lucas Numbers

Similar to the Fibonacci numbers, each Lucas number is defined to be the sum of its two immediate previous terms. The first two Lucas numbers are 2 and 1 as opposed to the first two Fibonacci numbers 0 and 1:

$$Lucas(n) = \begin{cases} 2 & \text{n=0} \\ 1 & \text{n=1} \\ Lucas(n-1) + Lucas(n-2) & \text{n>1} \end{cases} \tag{25.3}$$

Challenge 25.3 Find the missing numbers of the following sequence separating by comma:
2,1,3,4,7,?,18,29,?,76 ... ■

Solution 25.3 Same as before:

- For the first '?' we have that 4 + 7 = 11
- For the second '?' we have that 18 + 29 = 47

Thus the solution is 11,47.

And, of course, we have also a fast calculation formula:

$$Lucas(n) = \left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n \tag{25.4}$$

Challenge 25.4 Find the 150th number of the following sequence:
2,1,3,4,7,11,18,29,47,76 ... ■

Solution 25.4 The solution is:
$Lucas(150) = \left(\frac{1+\sqrt{5}}{2}\right)^{150} - \left(\frac{1-\sqrt{5}}{2}\right)^{150} = 22291846172619859445381409012498$

25.3 Look-and-say Sequence

Sometimes sequences does not follow pure mathematical formulas or reasoning, but some kind of other logic. The look-and-say sequence is a wonderful example to show. The rule to build the sequence is very simple: To generate the next term, read the digits of the previous number, counting the number of digits in groups of the same digit that occurs sequentially. For example i will read the number 31131122211 as:

- one 3
- two 1
- one 3
- two 1
- three 2
- two 1

The next number of the sequence will be 132113213221. This sequence was first thought by John Horton Conway, more known as the inventor of the famous "Conway's Game of Life". Starting from digit 1 the sequence is so composed:

1,
11,
21,
1211,
111221,
312211,
13112221,
1113213211,
31131211131221,
13211311123113112211,
11131221133112132113212221,
3113112221232112111312211312113211,
1321132132111213122112311311222113111221131221,
11131221131211131231121113112221121321132132211331222113112211,
3113112221131112311311121321123113213221121113122113121113222123113222113212221

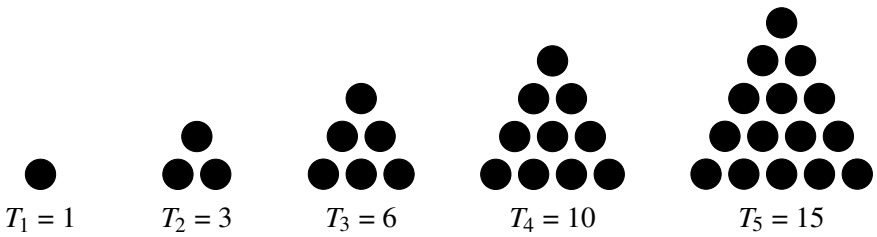
Challenge 25.5 — Just for fun. what is the next number of the sequence? ■

Solution 25.5 — Just for fun. 1321132132211331121321133112111312211213211312
21113222112311311222113111231133211121321132221131211321 1

26. Figurate Numbers

Figurate numbers are special numbers that can be represented by some regular geometry, according to some kind of arrangement. The way to arrange them can vary and assume, at same time, different geometric forms in different geometrical dimensions. When the arrangement forms a polygon we refer to **polygonal numbers**.

26.1 Triangular Numbers



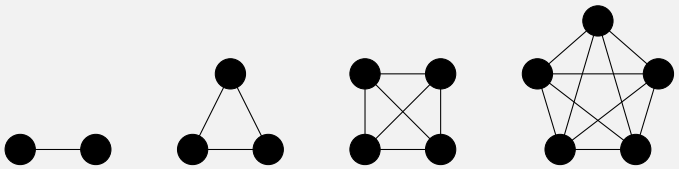
$$T_n = \sum_{k=1}^{n} k \tag{26.1}$$

$$T_n = \frac{n * (n+1)}{2} \tag{26.2}$$

Challenge 26.1 — Galactic meeting. University of Trento is organizing a galactic conference where bilateral discussion each-to-each between participants are on the agenda. All participants come from different galaxies and no one knows any foreign language. Organizer have to guarantee one interpreter for every bilateral discussion. Each interpreter is able only to translate only one pair of languages. Because two participants refused the invitation, the university could reduce number of interpreters by 31. How many interpreters will be on the meeting after being

reduced? and how many delegates were invited originally? ■

Solution 26.1 — Galactic meeting. Starting guessing logically, for 2 delegates only 1 interpreter is required, for 3 delegates we need 3 interpreters, for 4 delegates we need 6 interpreters, for 5 delegates 10 interpreters are required. This schema can be seen graphically below, where dots are delegates and lines are interpreters:



It follows that the number of required interpreters follow the triangular numbers sequence:

$$for \quad n \quad delegates \quad we \quad need \quad T_{n-1} \quad interpreters \tag{26.3}$$

To find the number of originally invited delegates we need to find n such that

$$T_{n-1} - T_{n-3} = 31 \tag{26.4}$$

Combining (26.4) with (26.2) we have:

$$\frac{(n-1)*((n-1)+1)}{2} - \frac{(n-3)*((n-3)+1)}{2} = 31 \tag{26.5}$$

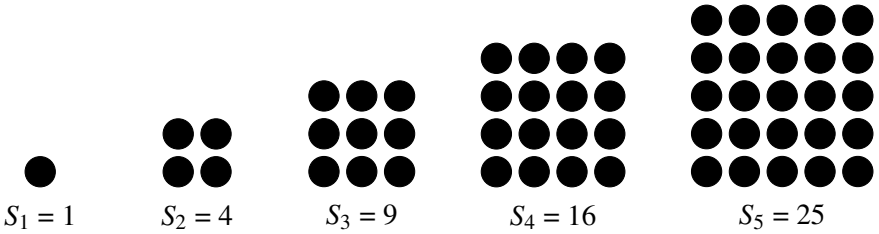
$$\frac{(n-1)*n}{2} - \frac{(n-3)*(n-2)}{2} = 31 \tag{26.6}$$

$$\frac{n^2-n-(n^2-3n-2n+6)}{2} = 31 \tag{26.7}$$

$$\frac{4n-6}{2} = 31 \rightarrow 2n-3 = 31 \rightarrow 2n = 34 \rightarrow \mathbf{n = 17} \tag{26.8}$$

Originally 17 delegates were invited. 2 of them declined the invitation so only 15 delegates will come. Form (26.3) follows that we need $T_{14} = 105$ interpreters.

26.2 Square Numbers



$$T_n = \sum_{k=1}^{n} k \tag{26.9}$$

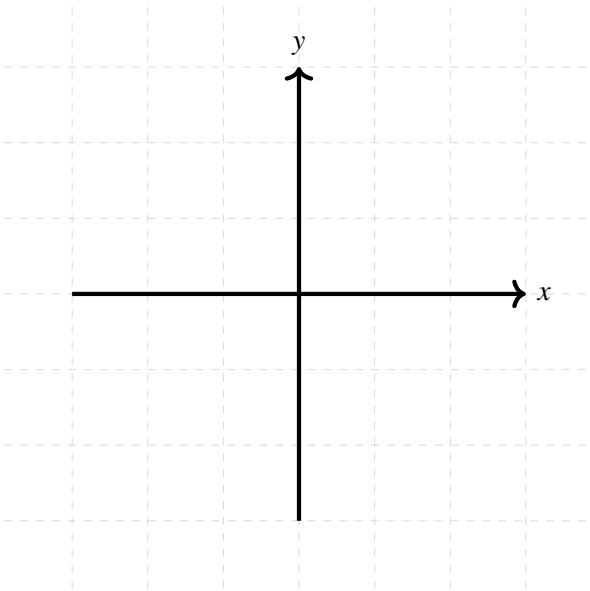
VII

Geometry

27. Coordinate Systems

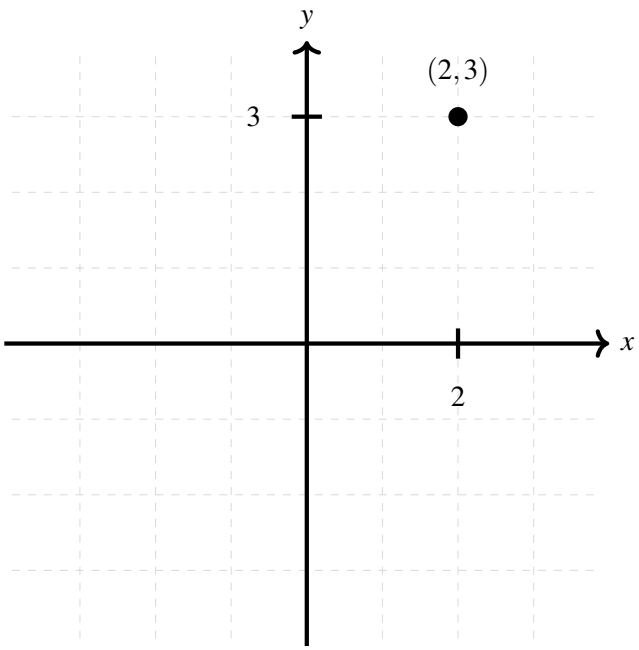
27.1 Cartesian Coordinate System

In the $XVII^{th}$ century, René Descartes provides for the first time in history a way to link Euclidean geometry and algebra through a coordinate system able to represent geometries described by Cartesian equations. The basic idea is that each point can be represented on the plain by a set of numerical coordinates. More formally these coordinates specify the signed distance of the point with respect to two fixed perpendicular lines. The point where these lines cross is called origin and has coordinates (0,0). In this way is pretty easy to represent figure and functions in two or more dimensions. Moving from a 2-dimensional to a 3-dimensional or even more dimensions can be difficult to imagine but still easy to represent. A point in a 3-dimensional space will be represented by 3 variables and has the origin in (0,0,0) and so on. The following figure represent a simple 2-dimensional plane:



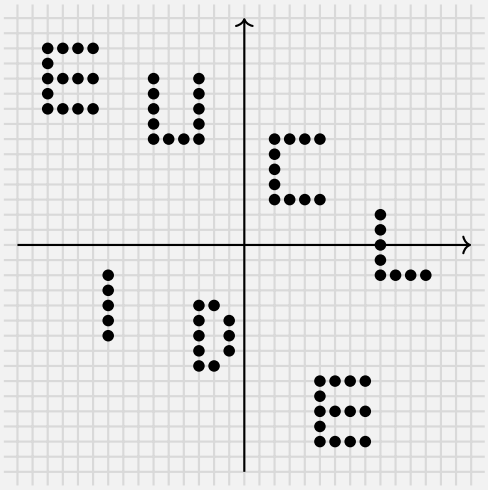
27.1.1 Points

Points are the most simple objects that can be represented in a coordinate system. As said before it is sufficient to specify a pair to coordinate to uniquely identify a point in the plan. As an example the point A(2,3) is represented as follows:



Challenge 27.1 — Points. (-13,13)(-13,12)(-13,11)(-13,10)(-13,9)(-12,13)(-11,13)(-10,13)(-12,11)(-11,11)(-10,11)(-12,9)(-11,9)(-10,9)(-6,11)(-6,10)(-6,9)(-6,8)(-6,7)(-3,11)(-3,10)(-3,9)(-3,8)(-5,7)(-4,7)(-3,7)(2,7)(2,6)(2,5)(2,4)(2,3)(3,7)(4,7)(5,7)(3,3)(4,3)(5,3)(9,-2)(9,-1)(9,0)(9,1)(9,2)(10,-2)(11,-2)(12,-2)(-9,-2)(-9,-3)(-9,-4)(-9,-5)(-9,-6)(-3,-4)(-3,-5)(-3,-6)(-3,-7)(-3,-8)(-2,-4)(-1,-5)(-1,-6)(-1,-7)(-2,-8)(5,-13)(5,-12)(5,-11)(5,-10)(5,-9)(6,-13)(7,-13)(8,-13)(6,-11)(7,-11)(8,-11)(6,-9)(7,-9)(8,-9) ■

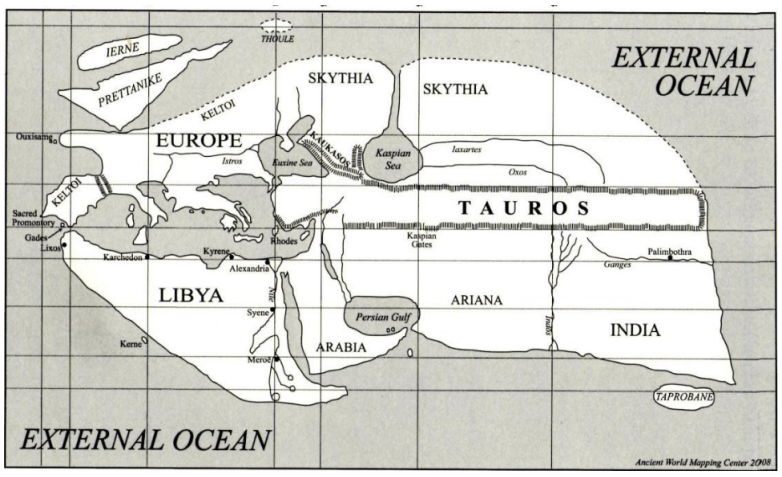
Solution 27.1 — Points. Just draw coordinates:



27.2 Geographic Coordinate System

27.2.1 Geo Points

When we talk about geographical points we immediately think about the most common and actual implementation in the world: Georeferentiation. With the advent of modern technologies, such as GPS devices, and their accessibility, almost all is able to identify a point on a geographic coordinate system, a coordinate system that identifies every location on Earth with a set of numbers, letters or symbols. Even if it can seem a modern technology, the first geographic coordinate system is credited to Eratosthenes of Cyrene in the far 3rd century BC (a reconstruction of the map is provided in the figure below). The 3 dimensions of the geographic coordinate system usually refers to latitude, longitude and elevation. To identify a point on a map it is first necessary to perform a map projection of latitudes and longitudes from a sphere or ellipsoid (the earth) to a plane, but how this is done is another story. No matters how projection is done or which geodetic datum is used, neither the coordinate format and the symbols used to represent them. Which cares is that, using the concept of latitude and longitude, we are able to represent the equivalent of a Cartesian point (the Geo point) to the equivalent of a Cartesian coordinate system (the Geographic coordinate system). This means that if Cartesian points can hide information, also Geo points can hide information in the same way as we seen in the previous subsection, but not only.



When we refer to Geo points we usually refer to specific location on the earth. Humans tend to identify things giving them at least a name, and locations have received, during years, the same treatment. Based on the level of granularity we choose a Geo point can belong to a continent, a state, a geographic region, a city, a route. This extends the possibility of hide information in a Geo point in different ways. Consider for example the following message containing also a list of Geo points:

we should kill:

- 54°37’02.8"N 7°52’37.1"W
- 53°34’30.5"N 7°23’33.6"W

- 54°36’42.4"N 6°24’38.4"W
- 54°07’10.9"N 8°03’18.1"W
- 53°39’45.6"N 7°13’43.5"W
- 53°39’23.6"N 7°22’29.4"W

All the points identify Irish lakes. The first one is the Lough Derg, the second is the Lough Owen, the third is Lough Neagh, the fourth is Lough Allen, the fifth is Lough Lene and the last one is Lough Derravaragh. Can you figure out what is hidden in these Geo points? The first letter of each lake give us a name: Donald.

Challenge 27.2 — Locations.
39°47’32.8"N 21°45’41.5"E
58°50’49.9"N 25°48’12.7"E
21°05’13.6"N 56°37’50.6"E
39°10’33.3"N 8°04’22.5"W
21°05’13.6"N 56°37’50.6"E
42°33’52.1"N 12°38’51.0"E
27°55’13.5"N 84°03’52.0"E
8°30’55.4"S 179°06’22.4"E
13°39’28.4"S 172°26’52.5"W ■

Solution 27.2 — Locations.
39°47’32.8"N 21°45’41.5"E = Greece
58°50’49.9"N 25°48’12.7"E = Estonia
21°05’13.6"N 56°37’50.6"E = Oman
39°10’33.3"N 8°04’22.5"W = Portugal
21°05’13.6"N 56°37’50.6"E = Oman
42°33’52.1"N 12°38’51.0"E = Italy
27°55’13.5"N 84°03’52.0"E = Nepal
8°30’55.4"S 179°06’22.4"E = Tuvalu
13°39’28.4"S 172°26’52.5"W = Samoa
GEOPOINTS

Trolling with WGS 84

The system used in the previous section, the WGS 84, is the most used standard in coordinate systems used for cartography and GPS. Its boundaries go from 90N to 90S and from 180E to 180W. This means that most of the ASCII characters are covered when location are encoded in WGS 84. This can be used to hide information in a different way. Consider the following location:

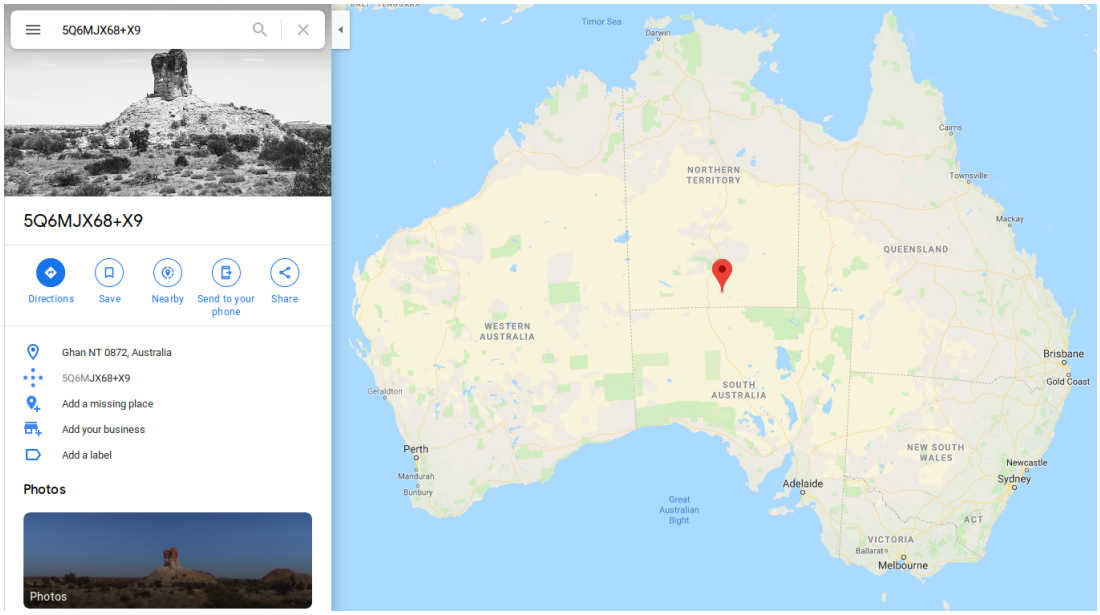
68°21’02.6"N 97°32’01.4"E

It can be refer to Ozero Lyuksina lake in Russia or, taking the degrees of latitude and longitude and converting them to ASCII characters, to the bi-gram "Da". This creates more ambiguity in decoding phase and opens coordinate information hiding to new fancy ways.

Open Location Code (OLC)

The open location code (OLC) born in the Google lab of Zurich in 2014 due to the necessity to encode locations in a more compact, easier and user friendly way than showing coordinates in the usual latitude-longitude format. As latitude-longitude system, near places have similar OLC codes

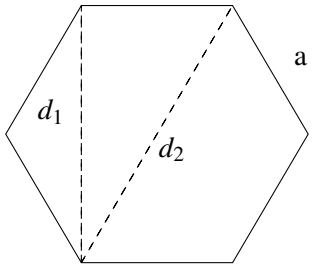
and they are starting to be supported by different applications like Google maps, but they are also accepted as postal addresses in some country like Cape Verde. OLC is only 10 characters long and it is composed by the first 4 characters representing the area code (a region of 100x100 kilometers), and the remaining characters representing the local code (a region of 14x14 meters inside the region specified by the area code). For more precision purposes and additional character can be added at the end of the OLC restricting the area to a 3x3 meters region.



28. Figures

28.1 Hexagon

Hexagon ...



$$d_1 = \sqrt{3} * a \tag{28.1}$$

$$d_2 = 2a \tag{28.2}$$

$$Perimeter = 6a \tag{28.3}$$

$$Area = \frac{3}{2} * \sqrt{3} * a^2 \tag{28.4}$$

(28.5)

Challenge 28.1 — Bees and honey. Mark assert that is able to calculate the liters of honey contained in a honeycomb just starting from its deep and from the product of its shortest and longest diagonals. Suppose that the honeycomb is equal to 5 cm deep and that the product of diagonals is equal to 240 cm. Are you able to tell me how many liters of honey fits into one honeycomb? ■

Solution 28.1 — Bees and honey. First of all we must recall that the volume of a regular solid is given by:

$$Volume = h * Area \tag{28.6}$$

So in our case:

$$Volume = 5 * \frac{3}{2} * \sqrt{3} * a^2 \tag{28.7}$$

From the riddle we know that the product between the long diagonal and the short one is 240:

$$d_1 * d_2 = 240 \tag{28.8}$$
$$(\sqrt{3} * a) * (2a) = 240 \tag{28.9}$$
$$2\sqrt{3} * a^2 = 240 \tag{28.10}$$
$$\sqrt{3} * a^2 = 120 \tag{28.11}$$
$$a^2 = \frac{120}{\sqrt{3}} \tag{28.12}$$

Now we can substitute (28.12) into (28.6):

$$Volume = 5 * \frac{3}{2} * \sqrt{3} * \frac{120}{\sqrt{3}} \tag{28.13}$$
$$= 5 * 3 * \sqrt{3} * \frac{60}{\sqrt{3}} \tag{28.14}$$
$$= 5 * 3 * 60 = 900 cm^3 \tag{28.15}$$

Converting to liters we obtain 0.9 liters (1 liter == $1000 cm^3$)

VIII Puzzles

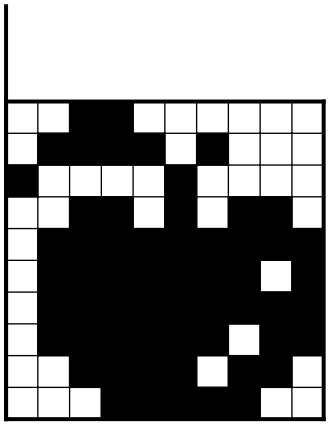
29. Japanese puzzles

29.1 Nonogram

Nonogram puzzle born in 1987 from an idea of Non Ishida and Tetsuya Nishio whom in the same time, curiously, had the same idea. First painted nonogams appeard in 1988 in Japan and in 1990 in the UK.

Nonograms are logic puzzles in which cells in a grid have to be filled or left blank according to the numbers at left and top of the grid. Filling the right cells will reveal the hidden picture. Clue numbers represent the numbers of consecutive filled cells are there in a row or column, spaces between numbers represent one or more spaces between 2 filled cells. For example the clue "2 4 1 3" means that there is a sequence of 2 filled cells, followed by an undefined sequence of non-filled cells, followed by a sequence of 4 filled cells, followed by an undefined sequence of non-filled cells, followed by 1 filled cell, followed by an undefined sequence of non-filled cells, followed by 3 consecutive filled cells. The order matter.
Nonograms have no theoretical limits on size, and are not restricted to square layouts.

	1	1 4	2 6	2 7	1 6	8	1 4 1	4 2	2 3	4
2										
4 1										
1 1										
2 1 2										
9										
7 1										
9										
6 2										
4 2										
5										

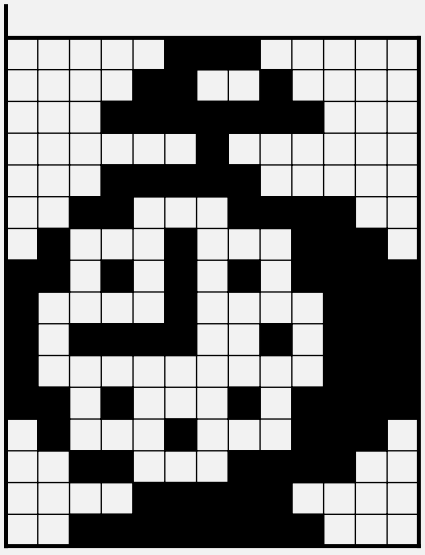


Challenge 29.1 — One more minute please. Solve the following nonogram:

	5	2 2	1 1 1	1 1 1 1 1	2 1 1 2	3 1 4 1 2	1 3 2	1 1 2 1 3	2 2 1 3	1 3 3 1	9	7	5
3													
2 1													
7													
1													
5													
2 4													
1 1 3													
2 1 1 1 4													
1 1 3													
1 4 1 3													
1 3													
2 1 1 4													
1 1 3													
2 4													
5													
8													

■

Solution 29.1 — One more minute please. Solved:

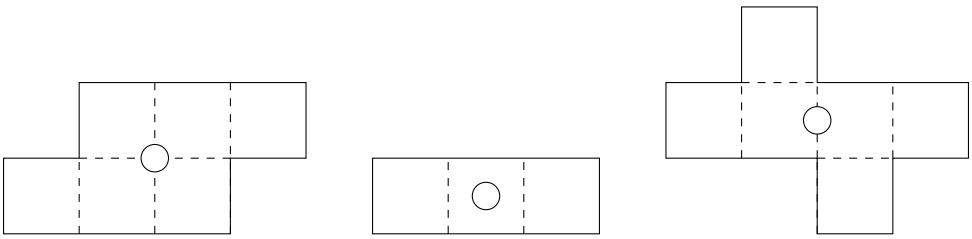


29.2 Tentai Show

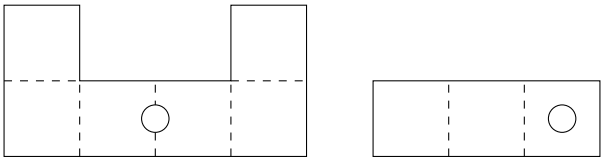
Tentai Show is a japanese puzzle published by Nikoli Inc. According to the publisher the name "Tentai Show" has two meanings: rotational symmetry and astronomical show. In Japanese "Ten" means dot, "tai-Show" means symmetry, and "Tentai" means astronomical. The first book containing Tentai-Show puzzles was released in October 2006 and contained 95 puzzles from 36 different creators. Another book came out in February 2009. Usually the puzzle is also referred as Galaxies, Spiral Galaxies and Artist Block. The puzzle is played in a rectangular grid: cells at the start of the puzzle are all blank and some of them may contain white or black dots. The game is based on symmetry rules and the aim is to divide the grid in regions according to the following rules:

- Each region must contain exactly one white or one black dot.
- The shape of each region is symmetrical with respect to the dot it contains (180 degrees rotational symmetry).

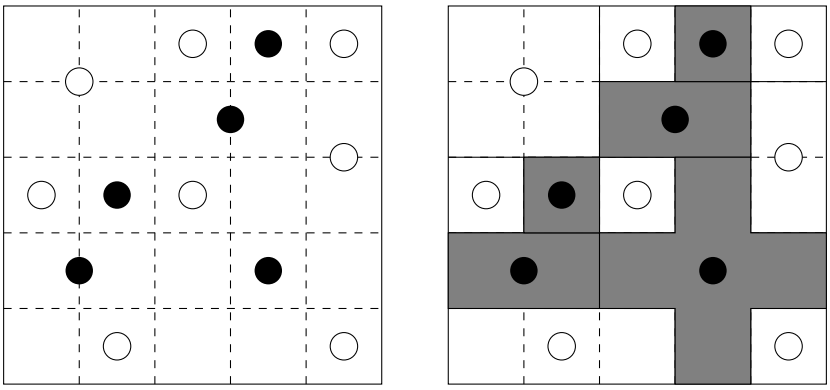
For example, the followings are examples of good and valid regions: rotating them by 180 degrees will produce the same region shape.



While the followings are bad regions: they does not have a 180 degrees symmetry with respect to the contained dot.

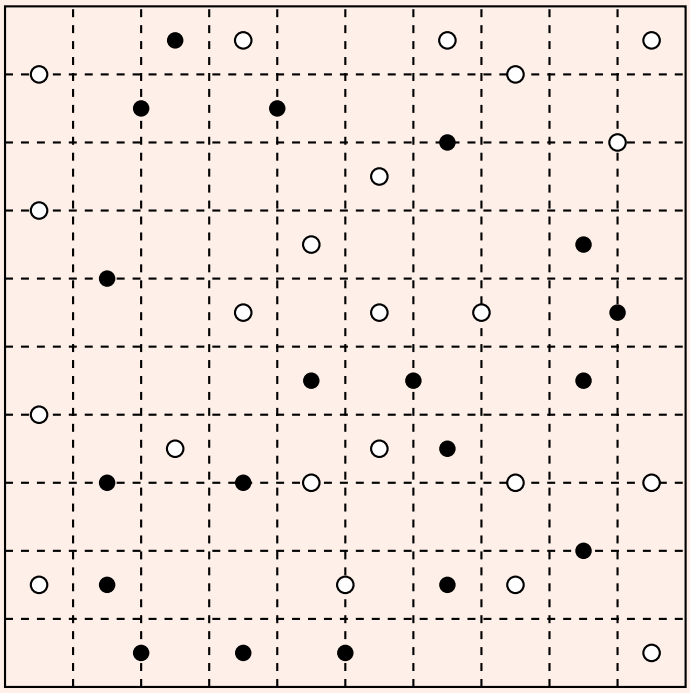


Once the puzzle has been solved the regions contained black dots are colored to reveal the final picture. Each puzzle is supposed to have an unique solution. The follow is a the classical example to show how Tentai Show works:

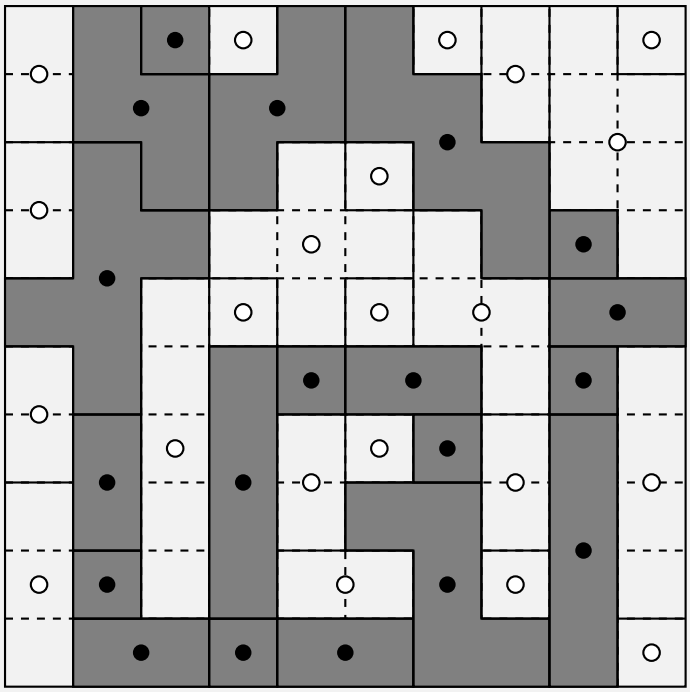


For the example puzzle the solution was the number 4.

Challenge 29.2 — A place to live.



Solution 29.2 — A place to live.



30. Mathematical puzzles

30.1 Petals Around the Rose

The first proposed mathematical challenging puzzle is called Petals Around the Rose where you are asked to guess a number that could be 0 or even. Number have to be guessed by looking at the result of a roll of some dice. The name of the game is a huge hint that helps to solve it, are you ready?

 The answer is 6

 The answer is 12

 The answer is 2

 The answer is 4

 The answer is 0

 The answer is 8

 The answer is 4

30.1.1 Solving the Rose

As the name suggests we have to count how many petals are around the roses. In our case roses are simply die with upper face containing a center dot so that odd faces as 1, 3 and 5 are roses and even faces as 2, 4 and 6 are not.

The petals of a rose are represented by dots that surround the central dot. There are no petals on the 1 face, so it also counts as zero. There are two petals on the 3 face and four 5. Thus the solution is quite easy to compute: each face 3 we count two petals and each face 5 we count 4 petals. Dummy one when known.

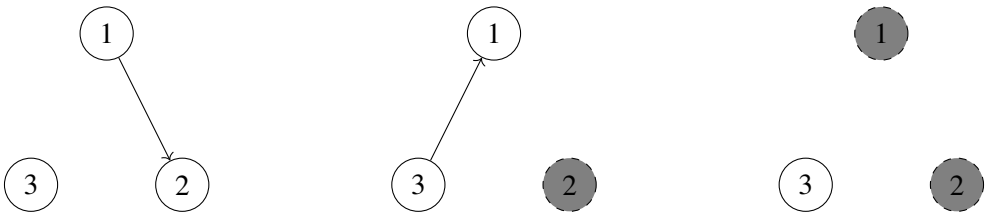
30.2 Josephus Problem (I d.C.)

Flavius Josephus was a Romano-Jewish historian born in Jerusalem and lived in the first century. During the siege of Yodfat (a 47-day siege by Roman forces of the Jewish town of Yodfat which took place in 67 CE and was led by Roman General Vespasian and his son Titus) he and other 40 soldiers were trapped in a cave by the Roman army, with no chance to fight or escape. To avoid to being captured they decided to suicide but, since suicide was a dishonor, they built up a method for which they were able to kill each others until one of them remained alive. This is described in book 3, chapter 8, part 7 of "The Jewish War":

"However, in this extreme distress, he was not destitute of his usual sagacity; but trusting himself to the providence of God, he put his life into hazard [in the manner following]: "And now," said he, "since it is resolved among you that you will die, come on, let us commit our mutual deaths to determination by lot. He whom the lot falls to first, let him be killed by him that hath the second lot, and thus fortune shall make its progress through us all; nor shall any of us perish by his own right hand, for it would be unfair if, when the rest are gone, somebody should repent and save himself." This proposal appeared to them to be very just; and when he had prevailed with them to determine this matter by lots, he drew one of the lots for himself also. He who had the first lot laid his neck bare to him that had the next, as supposing that the general would die among them immediately; for they thought death, if Josephus might but die with them, was sweeter than life; yet was he with another left to the last, whether we must say it happened so by chance, or whether by the providence of God. And as he was very desirous neither to be condemned by the lot, nor, if he had been left to the last, to imbrue his right hand in the blood of his countrymen, he persuaded him to trust his fidelity to him, and to live as well as himself."

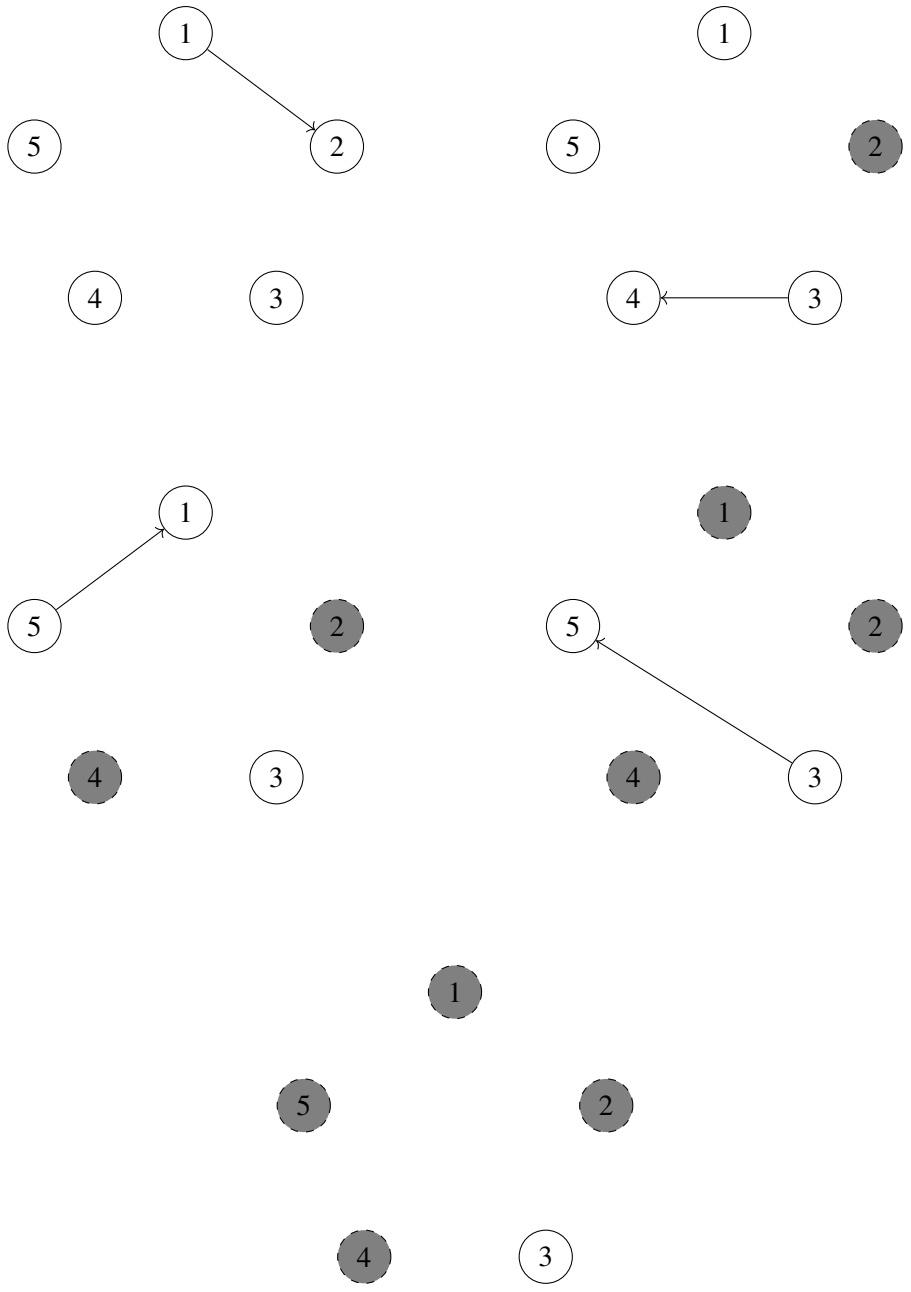
Subsequently in years, based on this story, a problem has been proposed: Supposing that 41 men are arranged in circle and, starting from a random place and proceeding clockwise, the men kill the one on his left, so that number 1 kills number 2, number 3 kills number 4, number 5 kills number 6 and so on. Knowing this, what will be the final survivor?

To answer this question let's try to solve the problem with less soldiers. The case of single soldier is trivial, he remains alive. Also in case of two soldiers the first one kills the second and survive. With three soldiers the number 1 kills number 2 and the number 3 kills number one. The survivor is so the soldier at position 3.



With five soldiers the number 1 kills number 2, the number 3 kills number 4, the number 5 kills

number 1 and the number 3 kills number 1. The survivor is, again, the soldier at position 3. The following plot will simulate the steps:



It's pretty clear that at first round of the circle, all of the even-numbered soldiers die, so the survivor cannot be a even-numbered soldier. We can continue in this way until we detect a pattern.

```
public class JosephusProblem {

  public static final int SOLDIERS = 41;

  public static void main(String[] args) {
    Soldier soldier = new Soldier();
```

```
    soldier.id = 1;

    Soldier current = soldier;
    for (int i = 2; i <= SOLDIERS; i++) {
      Soldier nextSoldier = new Soldier();
      nextSoldier.id = i;
      current.left = nextSoldier;
      current = nextSoldier;
    }

    current.left = soldier;

    while (soldier.id != soldier.left.id) {
      System.out.println("Soldier " + soldier.id + " kills " + soldier.left.id);
      soldier.left = soldier.left.left;
      soldier = soldier.left;
    }

    System.out.println("The survivor is soldier at position " + soldier.id);
  }

  public static class Soldier {
    int id;
    Soldier left;
  }
}
```

The above code solves the problem for 41 soldiers (as the original story says), but can be used to simulate what happens in the case of different soldiers cardinality. It's quickly clear that a pattern exists:

soldiers	**1**	**2**	3	**4**	5	6	7	**8**	9	10	11	12	13	14	15	**16**
survivor	**1**	**1**	3	**1**	3	5	7	**1**	3	5	7	9	11	13	15	**1**

soldiers	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	**32**
survivor	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31	**1**

The pattern is not obvious but it can be noticed that, usually, when a soldier is added, the survivor shifts by 2 positions (1, 3, 5, 7, etc.). The particular thing is that the pattern sometimes reset, and if we look closer we will notice that the pattern resets when very special number of soldiers are involved: powers of 2 ($1 = 2^0$, $2 = 2^1$, $4 = 2^2$, $8 = 2^3$, $16 = 2^4$, $32 = 2^5$, etc...). Considering that each natural number n can be written in the form of:

$$n = 2^m + l \tag{30.1}$$

With $0 <= l <= 2^m$. And considering that 2^m does not influence the result of the computation, we can concentrate only on parameter l. The table below summarize n as a combination of m and l:

n	**1**	**2**	3	**4**	5	6	7	**8**	9	10	11	12	13	14	15	**16**
m	0	1	1	2	2	2	2	3	3	3	3	3	3	3	3	4
l	0	0	1	0	1	2	3	0	1	2	3	4	5	6	7	0
survivor	**1**	**1**	3	**1**	3	5	7	**1**	3	5	7	9	11	13	15	**1**

It can be noticed that when $l = 0$ the survivor is the soldier at position 1, when $l = 1$ the survivor is the soldier at position 3, when $l = 2$ the survivor is the soldier at position 5, when $l = 3$ the survivor is the soldier at position 7. So in general we can state that:

$$survivor = 2l + 1 \tag{30.2}$$

To solve the original problem (with $n = 41$), we simply have to write n in the form of $2^m + l$ $\rightarrow 41 = 2^5 + 9$. Since $l = 9$ the final solution will be $2 * 9 + 1 = 19$.

30.3 Monty Hall Problem (1975)

Monty Hall was the original host of an American television game show called "Let's Make a Deal" in which, as the last game of the show, the player were asked to choose one of three doors hiding different kind of prizes. One of them was covering a car, while the other two were covering a couple of goats. After the player chose a door, Monty Hall, that knew the exact content of each door, opened one of the remaining two doors revealing a goat, to create suspance. At this point, in the original game show, the player was not allow to change choice, but the following problem has been proposed to the American Statistician in 1975, and then restated in a 1990 letter by Craig Whitaker:

Suppose you're on a game show, and you're given the choice of three doors: Behind one door is a car; behind the others, goats. You pick a door, say No. 1, and the host, who knows what's behind the doors, opens another door, say No. 3, which has a goat. He then says to you, "Do you want to pick door No. 2?" Is it to your advantage to switch your choice?

It's clear and intuitive that if the player cannot change the chosen door, it has 1/3 chances to get the car. But having the possibility to change choice after that the host opened a goat door, will your probability to win a car increase in some way? A study conducted by Grangberg and Brown in 1995 asked to 228 subjects if they would have changed the door. Only the 13% of them chose to switch concluding that the great majority of people assumed that each of the remaining two doors has exactly the same probability to hide a car or a goat and that switching door doesn't make any difference. Is this true? Not really. It has been stated that "No other statistical puzzle comes so close to fooling all the people all the time".

Let's simulate what happen using the following case: picking one random door at beginning, let's say number 1 we have only three different arrangements described in the table below.

Behind door 1	**Behind door 1**	**Behind door 1**	**Stay**	**Change**
Goat	Goat	Car	Win Goat	**Win Car**
Goat	Car	Goat	Win Goat	**Win Car**
Car	Goat	Goat	**Win Car**	Win Goat

It's pretty clear that in 2 cases over 3, changing door is the better choice for the player, the one which guarantees the high probability to win the car. It is also possible to simulate the game and check if the average probability to win is around 66,7%. The following code simulates a player playing the game 10000 times and randonly choose a door. In all games then the player will change the original door:

```
public class MontyHallParadox {

  public static final int GAMES = 10000;
  public static void main(String[] args) {

    List<Door> doors = new ArrayList<>();
    int victories = 0;
    int loses = 0;

    for (int i = 0; i < GAMES; i++) {
      doors.clear();
      int randomDoor = (int) (Math.random() * 3);
```

```
      for (int j = 0; j < 3; j++) {
        Door door = new Door();
        door.id = j;
        if (j == randomDoor) {
          door.prize = "Car";
          System.out.println("Car is behind door " + j);
        }
        doors.add(door);
      }

      int choice = (int) (Math.random() * 3);
      doors.get(choice).chosen = true;
      System.out.println("Player chose door " + choice);

      for (Door door : doors) {
        if (!door.chosen && door.prize.equals("Goat")) {
          door.opened = true;
          System.out.println("Host opened door " + door.id);
          break;
        }
      }

      for (Door door : doors) {
        if (!door.chosen && !door.opened) {
          System.out.println("Player switched to door " + door.id);
          if (door.prize.equals("Car")) {
            System.out.println("Player won");
            victories++;
          } else {
            System.out.println("Player lost");
            loses++;
          }
          break;
        }
      }
      System.out.println("Victories: " + victories);
      System.out.println("Loses: " + loses + "\n");
    }
    System.out.println("% of Victories: " + ((double) victories / (double)
        GAMES) * 100 + "%\n");
  }

  static class Door {
    int id;
    boolean opened = false;
    boolean chosen = false;
    String prize = "Goat";
  }
}
```

Running the simulation a bunch of times we can notice that the average probability to win by always switching the door is actually around 2/3:

1	2	3	4	5	6	7	8	9	10
65,62%	68,66%	66,38%	66,95%	66,64%	65,29%	66,55%	65,88%	66,7%	66,71%

IX Programming

31. Esoteric Languages

During the computer age, lot of programming languages have been created. Some of them aims to solve some specific problem, others try to solve the limitations of other programming languages. There is a special category of programming languages in which belong those languages that were experimental, fast proof of concepts or simply designed to be jokes. These languages are called esoteric languages, esolangs for friends.

31.1 Brainf*ck Family (1993)

Brainf*ck is one of the most popular esolang. It has been created by Urban Muller in 1993 and it is very minimalist, composed only by 8 commands mapped to 8 different symbols, as explained in the following table:

>	increment the data pointer (to point to the next cell to the right).
<	decrement the data pointer (to point to the next cell to the left).
+	increment (increase by one) the byte at the data pointer.
-	decrement (decrease by one) the byte at the data pointer.
.	output the byte at the data pointer.
,	accept one byte of input, storing its value in the byte at the data pointer.
[	if the byte at the data pointer is zero, then instead of moving the instruction pointer forward to the next command, jump it forward to the command after the matching] command.
]	if the byte at the data pointer is nonzero, then instead of moving the instruction pointer forward to the next command, jump it back to the command after the matching [command.

The Brainf*ck language uses some components of a simple machine: a tape consisting of cells (all initialized to zero), a dynamic pointer (initialized at left most cell of the tape), an input tape and

an output tape. As an example the following program will output "brainfuck":

```
1. ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   ++++++++++++++++++++++++++++++++++.
2. ++++++++++++++++.
3. ----------------.++++++++.+++++.
4. --------.
5. ++++++++++++++.
6. ------------------.
7. ++++++++.
```

In this example, the pointer never moves from the cell zero, increasing it first 98 times, outputting its ASCII equivalent char, "b". Then increase till reach char "r", then decreasing till char "a" and so on. At the end of each line the output function is called. The classical "Hello World" is reached with the following string:

```
++++++++[>++++[>++>+++>+++>+<<<<-]>+>+>->>+[<]<-]>>.>---.+++++++..+++.>>.<-.<.
+++.------.--------.>>+.>++.
```

31.1.1 Brainf*ck Variants

The popularity of Brainf*ck, since its creation, increased rapidly and many variants has been developed. Simplest variants consists in some replacement of original symbology. Some of them can be seen in the following table:

Brainf*ck	Alphuck	Blub	Ook!	Pikalang	Triplet	Ternary
>	a	Blub. Blub?	Ook. Ook?	pipi	001	01
<	c	Blub? Blub.	Ook? Ook.	pichu	100	00
+	e	Blub. Blub.	Ook. Ook.	pi	111	11
-	i	Blub! Blub!	Ook! Ook!	ka	000	10
.	j	Blub! Blub.	Ook! Ook.	pikachu	010	20
,	o	Blub. Blub!	Ook. Ook!	pikapi	101	21
[	p	Blub! Blub?	Ook! Ook?	pika	110	02
]	s	Blub? Blub!	Ook? Ook!	chu	001	12

For completeness, here are the Hello World! scripts of these programming languages:

```
iiciccepepceaiiiaiaiaicccsascciijceeeeeejccijjccjcejaajaajcccjeeejaajaaijcccej
```

```
blub. blub? blub. blub. blub. blub. blub. blub. blub. blub. blub. blub. blub.
    blub. blub. blub. blub. blub. blub. blub. blub! blub? blub? blub. blub.
    blub. blub. blub. blub. blub. blub. blub. blub. blub. blub. blub. blub.
    blub. blub. blub. blub. blub? blub! blub! blub? blub! blub? blub. blub!
    blub. blub. blub? blub. blub. blub. blub. blub. blub. blub. blub. blub.
    blub. blub. blub. blub. blub. blub! blub? blub? blub. blub. blub. blub.
    blub. blub. blub. blub. blub. blub. blub? blub! blub! blub? blub! blub?
    blub. blub. blub. blub! blub. blub. blub. blub. blub. blub. blub. blub.
    blub. blub. blub. blub. blub. blub. blub. blub! blub. blub! blub. blub.
    blub. blub. blub. blub. blub. blub! blub. blub. blub? blub. blub? blub.
```

```
    blub? blub. blub. blub. blub. blub. blub. blub. blub. blub. blub. blub.
    blub. blub. blub. blub. blub. blub! blub? blub? blub. blub. blub. blub.
    blub. blub. blub. blub. blub. blub. blub? blub! blub! blub? blub! blub?
    blub. blub! blub. blub. blub? blub. blub? blub. blub? blub. blub. blub.
    blub. blub. blub. blub. blub. blub. blub. blub. blub. blub. blub. blub.
    blub. blub. blub. blub. blub. blub! blub? blub? blub. blub. blub. blub.
    blub. blub. blub. blub. blub. blub. blub. blub. blub. blub. blub. blub.
    blub. blub. blub. blub. blub? blub! blub! blub? blub! blub? blub. blub!
    blub! blub! blub! blub! blub! blub! blub. blub? blub. blub? blub. blub?
    blub. blub? blub. blub! blub. blub. blub. blub. blub. blub. blub. blub!
    blub. blub! blub! blub! blub! blub! blub! blub! blub! blub! blub! blub!
    blub! blub! blub. blub! blub! blub! blub! blub! blub! blub! blub! blub!
    blub! blub! blub! blub! blub! blub! blub! blub! blub. blub. blub? blub.
    blub? blub. blub. blub! blub.
```

```
Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
    Ook. Ook. Ook. Ook. Ook! Ook? Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
    Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook? Ook! Ook! Ook? Ook!
    Ook? Ook. Ook! Ook. Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
    Ook. Ook. Ook. Ook. Ook. Ook! Ook? Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook.
    Ook. Ook. Ook. Ook? Ook! Ook! Ook? Ook! Ook? Ook. Ook. Ook. Ook! Ook. Ook.
    Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook! Ook.
    Ook! Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook! Ook. Ook. Ook? Ook. Ook? Ook.
    Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
    Ook. Ook. Ook! Ook? Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
    Ook? Ook! Ook! Ook? Ook! Ook? Ook. Ook! Ook. Ook. Ook? Ook. Ook? Ook. Ook?
    Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
    Ook. Ook. Ook. Ook. Ook. Ook! Ook? Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook.
    Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook? Ook!
    Ook! Ook? Ook! Ook? Ook. Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook. Ook? Ook.
    Ook? Ook. Ook? Ook. Ook? Ook. Ook! Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook!
    Ook. Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook.
    Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook!
    Ook! Ook! Ook. Ook. Ook? Ook. Ook? Ook. Ook. Ook! Ook.
```

```
pi pi pi pi pi pi pi pi pi pi pika pipi pi pi pi pi pi pi pi pipi pi pi pi pi pi
    pi pi pi pi pi pipi pi pi pi pipi pi pichu pichu pichu pichu ka chu pipi pi
    pi pikachu pipi pi pikachu pi pi pi pi pi pi pi pikachu pikachu pi pi pi
    pikachu pipi pi pi pikachu pichu pichu pi pi pi pi pi pi pi pi pi pi pi pi pi
    pi pi pi pikachu pipi pikachu pi pi pi pikachu ka ka ka ka ka ka pikachu ka
    ka ka ka ka ka ka pikachu pipi pi pikachu pipi pikachu
```

```
001 111 111 111 111 111 111 111 111 111 110 100 111 111 111 111 111 111 111 111
    001 000 011 100 010 001 111 111 111 111 111 111 111 110 100 111 111 111 111
    001 000 011 100 111 010 111 111 111 111 111 111 111 010 010 111 111 111 010
    001 001 001 111 111 111 111 111 111 111 111 110 100 111 111 111 111 001 000
    011 100 010 001 001 001 111 111 111 111 111 111 111 111 111 111 110 100 111
    111 111 111 111 111 111 111 111 001 000 011 100 000 000 000 010 100 100 100
    100 010 111 111 111 010 000 000 000 000 000 000 010 000 000 000 000 000 000
    000 000 010 001 001 111 010
```

```
1111111111111111020111111111020111110111111101111111011100000000101201110111011
0010111020012001012010120011010102011111111111111202011111120010120001020002011
```

```
1111201010101010201010101010101020010111200111120
```

31.1.2 Brainf*ck Extensions

Brainf*ck has been designed to be a very minimalist programming language but it has been widely extended with new functionality.

2D-Brainf*ck

The bi-dimensional view of brainf*ck can be achieved by using multi-tapes instead a single one. in this case the pointer is not only restricted to move right and left, but also up and down through the tapes. Usually 3 new symbols are added in the simplest extension:

$\wedge$	move the pointer up to the previous tape
$\vee$	move the pointer down to the next tape
$\times$	exit the program

31.2 The Deadfish Language

Deadfish is a minimalist esoteric programming language created as a parody of extremely limited or awkward syntactic designs. Its name humorously refers to the fact that the language "does almost nothing" and "flops around" like a dead fish. Despite (or because of) its quirks, Deadfish has become a popular example in esolanguage communities due to its simplicity and the ease with which it can be implemented. Deadfish operates on a single integer variable and supports exactly four instructions. Programs thus resemble strings of four characters, often interpreted left to right.

The language maintains one integer register, commonly denoted as x, initialized to 0. There is no memory beyond this. During execution, four commands can manipulate x:

- `i` Increment x: $x \leftarrow x + 1$
- `d` Decrement x: $x \leftarrow x - 1$
- `s` Square x: $x \leftarrow x^2$
- `o` Output x (typically as its integer value or ASCII code)

To preserve finite behaviour, Deadfish follows two special rules:

1. If x becomes 256, it resets to 0.
2. If x becomes -1, it also resets to 0.

These two wraparound behaviours prevent values from escaping into large ranges. Deadfish is not Turing-complete. With only one integer register, no loops, and no conditional branching, it cannot express general computation. It is best viewed as a novelty language demonstrating how little machinery a programming language can have while still being technically usable.. Nonetheless, minimal interpreters are easy to write in almost any language, making Deadfish a popular beginners target for interpreter construction.

31.2.1 Deadfish Variants

Several alternative syntaxes for Deadfish exist, differing only in which characters are used for the four fundamental operations. All variants preserve the same semantics: increment, decrement, square, and output. The most commonly referenced alternatives are shown in Table 31.1.

Table 31.1: Comparison of Deadfish Instruction Variants

Variant	Increment	Decrement	Square	Output
Original	`i`	`d`	`s`	`o`
XKCD Variant	`X`	`D`	`K`	`C`
Numbered Variant	`1`	`2`	`3`	`4`
Visual Helper Variant	`+`	`-`	^{2}	`=`

31.2.2 Example Programs

Output the Number 1

```
io
```

Execution:

$$x = 0 \xrightarrow{\texttt{i}} 1 \xrightarrow{\texttt{o}} \text{output “1”}.$$

Output the ASCII Letter “A”

The ASCII value of “A” is 65. One possible program is:

```
iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiio
```

That is, apply `i` 65 times, then output.

32. Obfuscation

Code obfuscation techniques are very popular nowadays.... What is obfuscation...

32.1 Perl Obfuscation

32.1.1 JAPH (Just Another Perl Hacker)

Simply speaking JAPH is a program written in perl that does only a single thing: it prints the string "Just Another Perl Hacker". JAPH programs are usually used to show how far obfuscation can go and how can be creative. The first JAPH program was written by Randal L. Schwartz in 1988 and was very simple and no obscure at all:

```
print "Just another Perl hacker,"
```

32.1.2 Other perl programs

Obfuscation in perl provides use some other funny examples. The next one will print "Just another Perl programmer"

```
undef$/;$_=<DATA>;y/ODA\n / /ds;@yoda=map{length}split;print chr
oct join('',splice(@yoda,0,3))-111 while@yoda;
__DATA__
        OOOOOOOOOOOOOOOO    OOOO        DDOOOOOOOO
       ODOOOOOOOOOOOOOOOO   OOOOOO      OOOOOOOOOOO
       OOOO     OOO         OOO  OOO     OOD     ODO
        OOOO    OOO        OOD    OOO    DOODOOOOOD
         OOOO   DOD       OOOOOOOOOOOO   OOO  OOOO
 DDOOOODOOOOO   OOO      OOOOOOOOODOODO   OOD   DOOOOOODOOOO
 OOOOODOOOOO    OOO     OOOO        OODO  OOD    OOOOOOOODOO

 OOOOO   ODOD   OOOOO  OOOO        DDDOOOOOOO     OOOOOOOOOO
  OOOOO OOODDO OOOOO OOODOO       OOOOOOOOOOO   OOOOOOOOOO
   OOOOOOODOOOOODDD OOO  ODO      DDD     DOO    OOOOD
```

```
    OOOOOOOOOOOOOOO  OOO    DOO      DOODOOOOOO      OODOO
     DOOOO  OOOOO   OOOOOOOOOOOD    OOO  OOOO        OODOO
      OOO    OOO   OOOOOODOOOOOOOO  OOO   OODOOOOOOOOOOOOOO
       0      0   OOOD        OOOO   OOO    OOOOOOOOOOOOOO
```

or again, the following camel will produce 4 other camels, amzing!

```
use strict;

                                        $_='ev
                                      al("seek\040D
          ATA,0,                     0;");foreach(1..3)
        {<DATA>;}my                 @camel1hump;my$camel;
   my$Camel ;while(                <DATA>){$_=sprintf("%-6
9s",$_);my@dromedary              1=split(//);if(defined($
_=<DATA>)){@camel1hum           p=split(//);}while(@dromeda
 ry1){my$camel1hump=0     ;my$CAMEL=3;if(defined($_=shif
       t(@dromedary1   ))&&/\S/){$camel1hump+=1<<$CAMEL;}
      $CAMEL--;if(d  efined($_=shift(@dromedary1))&&/\S/){
     $camel1hump+=1 <<$CAMEL;}$CAMEL--;if(defined($_=shift(
    @camel1hump))&&/\S/){$camel1hump+=1<<$CAMEL;}$CAMEL--;if(
    defined($_=shift(@camel1hump))&&/\S/){$camel1hump+=1<<$CAME
    L;;}$camel.=(split(//,"\040..m`{/J\047\134}L^7FX"))[$camel1h
     ump];}$camel.="\n";}@camel1hump=split(/\n/,$camel);foreach(@
     camel1hump){chomp;$Camel=$_;y/LJF7\173\175`\047/\061\062\063\
     064\065\066\067\070/;y/12345678/JL7F\175\173\047`/;$_=reverse;
      print"$_\040$Camel\n";}foreach(@camel1hump){chomp;$Camel=$_;y
       /LJF7\173\175`\047/12345678/;y/12345678/JL7F\175\173\0 47`/;
        $_=reverse;print"\040$_$Camel\n";}';;s/\s*//g;;eval; eval
          ("seek\040DATA,0,0;");undef$/;$_=<DATA>;s/\s*//g;( );;s
            ;^.*_;;;map{eval"print\"$_\"";}/.{4}/g; __DATA__ \124
              \1   50\145\040\165\163\145\040\157\1 46\040\1 41\0
                   40\143\141 \155\145\1 54\040\1 51\155\ 141
                   \147\145\0 40\151\156 \040\141  \163\16 3\
                    157\143\  151\141\16 4\151\1    57\156
                    \040\167 \151\164\1  50\040\     120\1
                    45\162\  154\040\15   1\163\     040\14
                    1\040\1  64\162\1     41\144      \145\
                    155\14   1\162\      153\04       0\157
                     \146\    040\11    7\047\        122\1
                     45\15     1\154\1 54\171         \040
                     \046\       012\101\16          3\16
                     3\15          7\143\15          1\14
                     1\16           4\145\163         \054
                    \040           \111\156\14       3\056
                   \040\         125\163\145\14       4\040\
                   167\1        51\164\1 50\0        40\160\
                 145\162                            \155\151
               \163\163                              \151\1
             57\156\056
```