Searching for a Solution

... just an bunch of boring stuff ...

Dr. Simone Sandri

PUBLISHED BY ME

BOOK-WEBSITE.COM

Digital copy, July 2019

Contents

I Introduction

1 Before Reading *(.5pc).11
1.1 What is the book about 11
1.2 Who should read it 11
1.3 Who should not read it 11
1.4 Riddles and Challenges 11
1.5 A quite important difference 11
1.6 Java Code 12

2 General Knowledge *(.5pc).13
2.1 An Unique Meaning? 13
2.1.1 Base Conversion 14

II Steganography

3 Text Stego *(.5pc).19
3.1 Just be Careful 19
3.1.1 Noise 19
3.1.2 Paragraphs, Sentences, Words, Characters 22
3.2 Text manipulation 24
3.2.1 Reversing 24
3.2.2 Mispelling Steganography 24

3.3 1337 25

4 Computer Based Steganography *(.5pc).27
4.1 Files 27
4.2 Keyboard 30
4.2.1 Keyboard layouts steganography . . . 30
4.2.2 Around the target . . . 30
4.2.3 Follow the stream . . . 30

5 Substitution Steganography *(.5pc).33
5.1 Polygraphia (1518) 33
5.1.1 Theban Alphabet . . . 33
5.2 Bacon's Cipher (1605) 34
5.3 Pigpen Cipher (18th Century) 35
5.3.1 Templar Cipher . . . 35
5.3.2 Rosicrucian Cipher . . . 36
5.4 The Dancing Man Code (1903) 37
5.5 SMS (1993) 37

6 Colors Steganography *(.5pc).39
6.1 RGB Color Model 39
6.2 CMYK Color Model 39
6.3 HSL Color Model 40
6.4 HEX Color Model 40

7 Chemical Steganography *(.5pc).41
7.1 The Elements 41
7.2 Codons 42

III Encoding

8 Ancient Languages *(.5pc).47
8.1 Runic alphabets (150 A.D.) 48
8.1.1 Elder Futhark (2nd to 8th centuries) . . . 48
8.1.2 Anglo-Saxon Runes (5th to 11th centuries) . . . 49
8.1.3 Younger Futhark (9th to 11th centuries) . . . 50

9 Fictional Languages *(.5pc).51
9.1 Messages from the Space 51
9.1.1 Aurebesh (Star Wars) . . . 51
9.1.2 Futurama Alien Alphabet (Used in Futurama cartoon) . . . 52
9.1.3 Visitor (Used in TV series of 2009) . . . 52
9.1.4 Ancient Alphabet (Used in TV series "Stargate") . . . 52
9.1.5 Tenctonese (Alien Nation) . . . 53

9.2 Fantasy Messages **54**
9.2.1 Iokharic (Draconic Language) . . . 54
9.2.2 Daedra (Language of the Daedra Race of Oblivion) . . . 54

10 Useful encoding *(.5pc).55
10.1 Semaphore (1794) **55**
10.1.1 Flag Semaphore . . . 55
10.2 Braille (1824) **56**
10.2.1 Grade 1 . . . 56
10.2.2 Grade 2 . . . 56
10.2.3 Grade 3 . . . 57
10.3 Morse Code (1835 - 1840) **58**
10.4 ASCII (1960 - 1967) **59**
10.4.1 Extended ASCII (1980s - 2000s) . . . 61
10.5 DTMF (1963) **63**
10.6 BaseN Encodings (1993) **65**
10.6.1 Base64 . . . 65
10.6.2 Base32 . . . 66

IV Cryptography

11 Classical Cipher *(.5pc).71
11.1 Rot-n **71**
11.1.1 Rot-5 . . . 71
11.1.2 Rot-13 . . . 72
11.1.3 Rot-13.5 . . . 72
11.1.4 Rot-47 . . . 73
11.2 Substitution Ciphers **74**
11.2.1 Atbash, Albam, Atbah . . . 74
11.2.2 Caesar Cipher (100 B.C. - 44 B.C.) . . . 75
11.2.3 Affine Cipher . . . 75
11.3 Polygraphic Substitution Ciphers **77**
11.3.1 Hill's Cipher (1929) . . . 77

12 Modern Cryptography *(.5pc).79
12.1 RSA (1977) **79**
12.1.1 Breaking RSA . . . 80
12.1.2 Fermat's Attack . . . 80
12.1.3 Common modulus . . . 81
12.1.4 Small Public Exponent . . . 83
12.1.5 Hastad's Broadcast Attack . . . 83
12.1.6 Wiener's Attack . . . 84
12.2 Rabin Cryptosystem (1979) **84**
12.3 Block Ciphers (1981) **85**
12.3.1 Electronic Codebook (ECB) . . . 85

13 Cryptanalysis *(.5pc).87
13.1 Frequency Analysis 87
13.1.1 Frequency steganography 89

V Sequences

14 Famous Sequences *(.5pc).93
14.1 Fibonacci Numbers 93
14.2 Lucas Numbers 94
14.3 Look-and-say Sequence 95

15 Figurate Numbers *(.5pc).97
15.1 Triangular Numbers 97
15.2 Square Numbers 98

VI Geometry

16 Coordinate Systems *(.5pc).101
16.1 Cartesian Coordinate System 101
16.1.1 Points 102

17 Figures *(.5pc).103
17.1 Hexagon 103

VII Puzzles

18 Japanese puzzles *(.5pc).107
18.1 Nonogram 107

19 Mathematical puzzles *(.5pc).109
19.1 Petals Around the Rose 109
19.1.1 Solving the Rose 109

VIII Programming

20 Esoteric Languages *(.5pc).113
20.1 Brainf*ck Family (1993) 113
20.1.1 Brainf*ck Variants 114
20.1.2 Brainf*ck Extensions 116

21 Obfuscation *(.5pc).117
21.1 Perl Obfuscation 117
21.1.1 JAPH (Just Another Perl Hacker) 117
21.1.2 Other perl programs 117

Bibliography *(.5pc).121
Articles 121
Books 121

Introduction

1 Before Reading *(.5pc).11
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
1.5 A quite important difference
1.6 Java Code

2 General Knowledge *(.5pc).13
2.1 An Unique Meaning?

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.

TODO challenges TODO inverse

II

Steganography

3 Text Stego ____ *(.5pc).19
3.1 Just be Careful
3.2 Text manipulation
3.3 1337

4 Computer Based Steganography *(.5pc).27
4.1 Files
4.2 Keyboard

5 Substitution Steganography ____ *(.5pc).33
5.1 Polygraphia (1518)
5.2 Bacon's Cipher (1605)
5.3 Pigpen Cipher (18th Century)
5.4 The Dancing Man Code (1903)
5.5 SMS (1993)

6 Colors Steganography ____ *(.5pc).39
6.1 RGB Color Model
6.2 CMYK Color Model
6.3 HSL Color Model
6.4 HEX Color Model

7 Chemical Steganography ____ *(.5pc).41
7.1 The Elements
7.2 Codons

3. Text Stego

3.1 Just be Careful

Il testo

Challenge 3.1 ...
10000001110000000011111000111100111100100001000000001000000001
00111100110011111001110010011100011100110011111001111001111111
00111111110011111001100111001100001100110011111001111001111111
00111000010000000011001111100100100100110011111001111000000111
00111100110011100111000000000100110000110011111001111001111111
00111100110011110011001111100100111000110011111001111001111111
10000001110011111001001111100100111100100001111001111000000001
■

Yes, that was easy, a bunch zeros and ones were hiding a secret message.

Solution 3.1 ...
10000001110000000011111000111100111100100001000000001000000001
00111100110011111001110010011100011100110011111001111001111111
00111111110011111001100111001100001100110011111001111001111111
00111000010000000011001111100100100100110011111001111000000111
00111100110011100111000000000100110000110011111001111001111111
00111100110011110011001111100100111000110011111001111001111111
10000001110011111001001111100100111100100001111001111000000001

3.1.1 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 3.2 — 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*r"/š=řňš/+ğ¡]ğňij?)-šøţ@/l*/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?-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=+ćl(+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/ňź>ř<@şč/ğ(]šč|]@++ijijć=ňň>¡ø|!*/]šijźţ>š?)ğ*'š¡*<řø[ţ
!)øćč:;š¡?ğ[/@";[;øč-ğæ-š[<ğš!¡/ň!!<æ:ijl@æš*):ć¡[;ij//>!ţ;i'/=;źæ))ğ
ź]ř[@[+-[;č*@'æ!+ş/:čğ*š-*/lňţijš-ňč|ň+*=-ø"-ş+ğ?*=/l-ş@=@+]/=!şź:l!-)ţ*
ć+ij@'?*ijø¡||=|ćøţ/'+æź<>ţlň: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 3.2 — 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|šlčřl>ř>@ţ=;:ć?;(ij)¡*ţ(ň-ćţ[>])ş¡ř?-ææ
ğ"|;=ţ'@]ğøø/¡-ćč-ijl;!?=?čij<ğč:-æ¡((?/ňć+<|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)şňć=*:[-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/ňź>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+čijæń(?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();
}
```

3.1.2 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 3.3 — 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 3.3 — 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 3.4 — 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 3.4 — 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 3.5 — 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 3.5 — Count 1,2,3,4. Decoded:
lorem

Challenge 3.6 — 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 3.6 — 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.

3.2 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...

3.2.1 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 3.7 — 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 3.7 — 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 3.8 — 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 3.8 — 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 3.9 — 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 3.9 — 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.

3.2.2 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".

3.3 1337

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|	/\\	/X\	|V|		
N	||	|V	/V			
O	0	()	[]	{}	<>		
P	|o	|O	|>	|*	|ř		
Q	O_	9	O,				
R		2	12	12	ő		
S	5	$	ğ				
T	7	’|’	‘|‘	+			
U	|_|	_\	/_/	_/	(_)	[_]	{_}
V	\/						
W	VV	\/\/	(/\)	\X/	|/|		
X	%	*	><	}{	][	)(	
Y	‘/	ě					
Z	2	7_					

Challenge 3.10 — They are only just words. c3n50r5h1p 15 7h3 w0r57 3n3my 0f fr33d0m 0f 5p33ch ■

Solution 3.10 — They are only just words. censorship is the worst enemy of freedom of speech

Challenge 3.11 — 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 3.11 — Madam in Eden, I’m Adam. we should be free to think and express our thoughts without the fear of being judged

Challenge 3.12 — L0L. !|# !+ \\|/3|23 |\\|0+ 50 !+ \\|/0|_|1|) |\\|0+ 83 +|-|3 \\|/0|21|) !|\\| \\|/|-|!(|-| ! \\|/0|_|1|) 1!|(3 +0 1!\\/3 ■

Solution 3.12 — Madam in Eden, I’m Adam. if it were not so it would not be the world in which I would like to live

4. Computer Based Steganography

4.1 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**

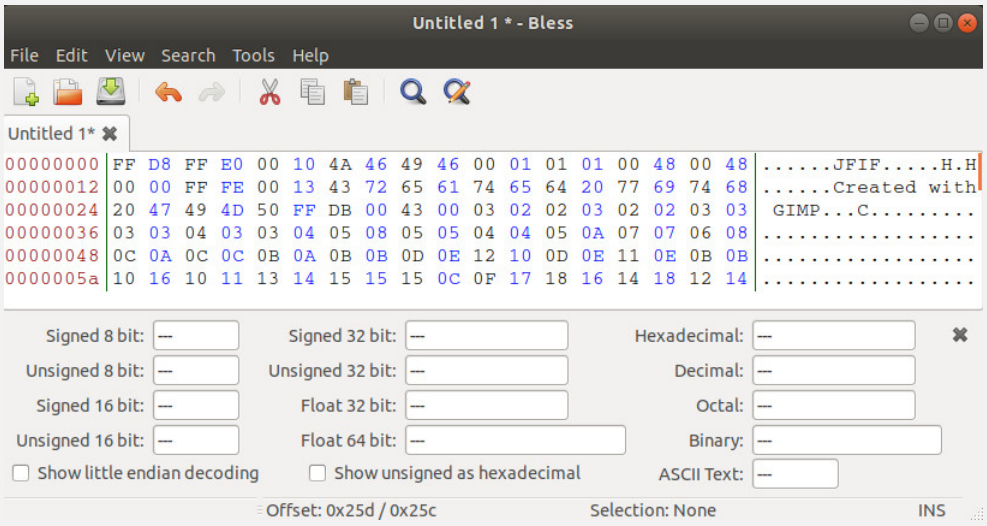
So the signature of PNG images is 89 50 4E 47 0D 0A 1A 0A. Some formats, like PNG as well, have also a fixed array of bytes at the end that represents the "trailer" (In the case of PNG is 49 45 4E 44 AE 42 60 82). The signatures of some famous formats can be found in the table below:

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

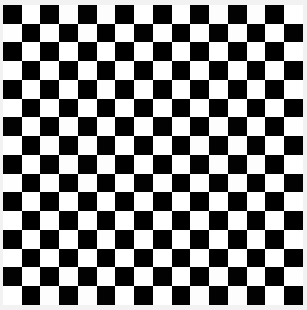
Challenge 4.1 — Black And White. What does this file represent?
FF D8 FF E0 00 10 4A 46 49 46 00 01 01 01 00 48 00 48 00 00 FF FE 00 13 43 72 65 61 74 65 64 20 77 69 74 68 20 47 49 4D 50 FF DB 00 43 00 03 02 02 03 02 02 03 03 03 03 04 03 03 04 05 08 05 05 04 04 05 0A 07 07 06 08 0C 0A 0C 0C 0B 0A 0B 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 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 4.1 — 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.

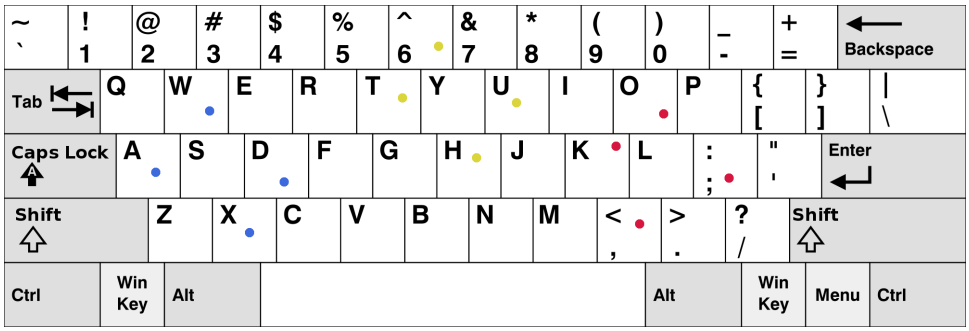
4.2 Keyboard

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.

4.2.1 Keyboard layouts steganography

4.2.2 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.



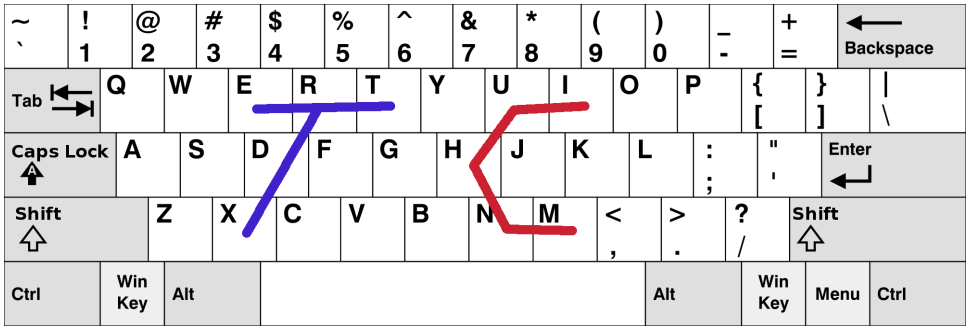
Moving on with examples the above image represent in blue the key sequence "awdx" representing the hidden key 's', in yellow the key sequence "t6uh" representing the hidden key 'y' and in red the sequence "ko;," representing the letter 'l'.

Challenge 4.2 — The name of my love. w3rd waxd gyr5 r4wd fest ■

Solution 4.2 — The name of my love. ester

4.2.3 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 4.3 — Fruits. zsedc aw3edr5 ki8uh -plo9ij cftgb 098uiokjh ■

Solution 4.3 — Fruits. ananas

5. Substitution Steganography

5.1 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.

5.1.1 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 5.1 — Shakespeare. Short citation

■

Solution 5.1 — Shakespeare. Frailty, thy name is woman

5.2 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 5.2 — Gnam Gnam. baabaaabbbaabaa aababababaaabaaaabaaabbaaba baabaabaaaabbabbaabaa abaaa baaaaaabaaaaaaaaaabb baabaaabbbaabaa abbaaaaaaaabbabbaabaa, ababbaaaaaabaabaabaabaaab ababbaabaa aabbbbaabbabbaaaabbabaaaababba ■

Solution 5.2 — Gnam Gnam. the first time I read the name, makes me hungry

Challenge 5.3 — I am famous. 01100000001100000000100100 00001000000001001111001101 10110100010111010011001001001 10010011100110000100 01111010110000011000100100000000011100100 01111010110000011000100100 10110001110100000010001111011000100100010010 00000100111001110001010000000110100100110010000001 1001101110 10010001110000001010001001001001111001000000010001001001?

■

Solution 5.3 — I am famous. maybe Bacon wrotes some plays ahe plays which were attributed to Shakespeare?

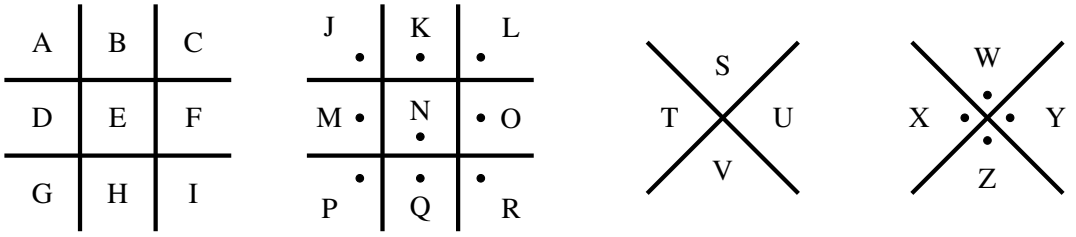
Challenge 5.4 — 5 pm. yes Please buT Be SuRe it is NOT too hoT bEcAusE i like to tasTe It WithOUT BuRNIng mY pALatE And My tongue ■

Solution 5.4 — 5 pm. can I have a cup of tea?

5.3 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:

I 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 5.5 — Bacon Everywhere. ⌜ ⊐⊏ ⊡⊏> ⊔⊡⊏∨ ∨⊓< ⊔<>

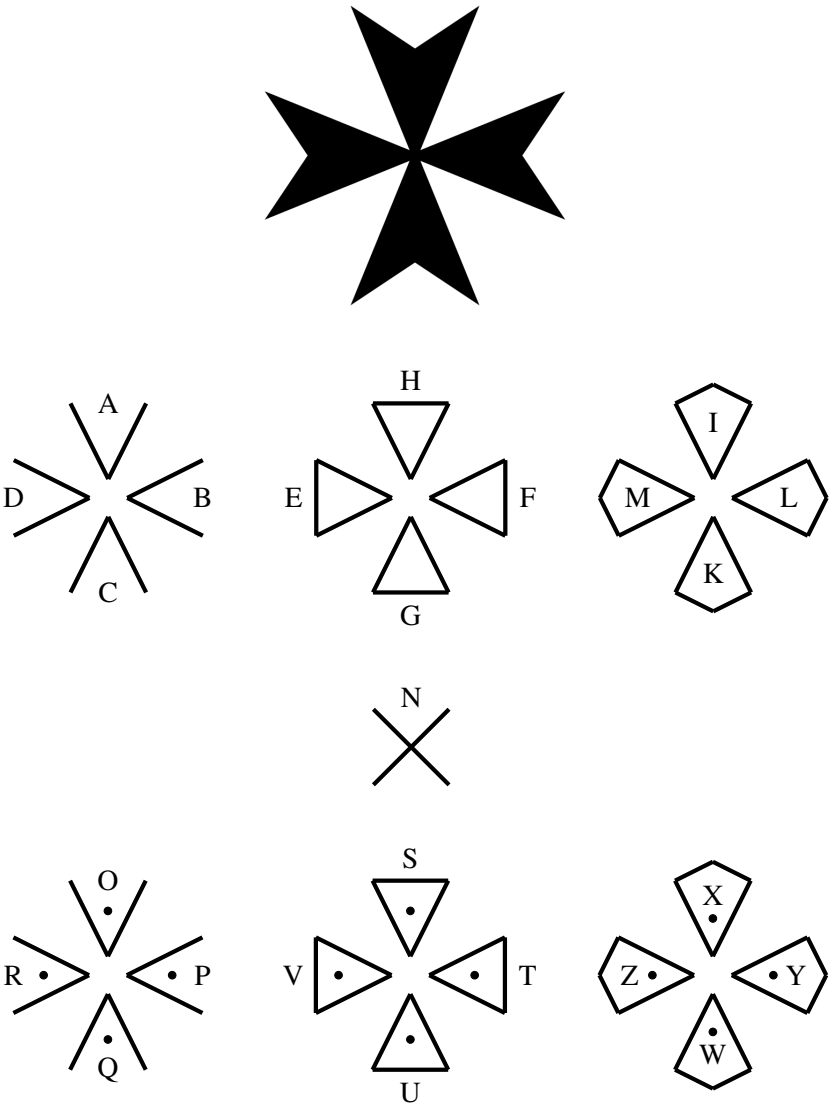
⌞⌜<¬>⊏¬⌜ ⌟¬⊓< ⊓⌟∨ ∨⊏⊐□ ∨∨□□> ⌞⊏⊡⊡□⌞>⌜⊏⊡∨

∨⌜>⊓ ⊏⊏⊏⊐ ■

Solution 5.5 — Bacon Everywhere. I do not know why, but cryptography has some sweet connections with food.

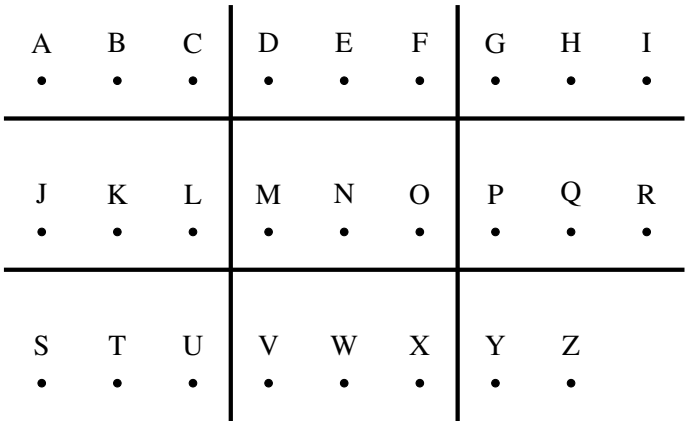
5.3.1 Templar Cipher

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.



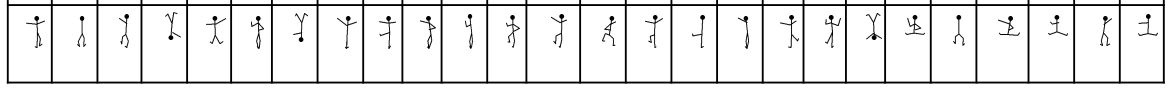
5.3.2 Rosicrucian Cipher

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:

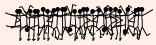


5.4 The Dancing Man Code (1903)

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 5.6 — The Dancing Man. 



■

Solution 5.6 — The Dancing Man. No one can compete with his incredible intelligence

5.5 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 5.7 — Nokia 3310. 46666663066655530844463377777 ■

Solution 5.7 — Nokia 3310. good old times

6. Colors Steganography

This is a very introductory section about colors, different types of models to represent them, and how this models can hide information.

6.1 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

6.2 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

6.3 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

6.4 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.

7. Chemical Steganography

7.1 The Elements

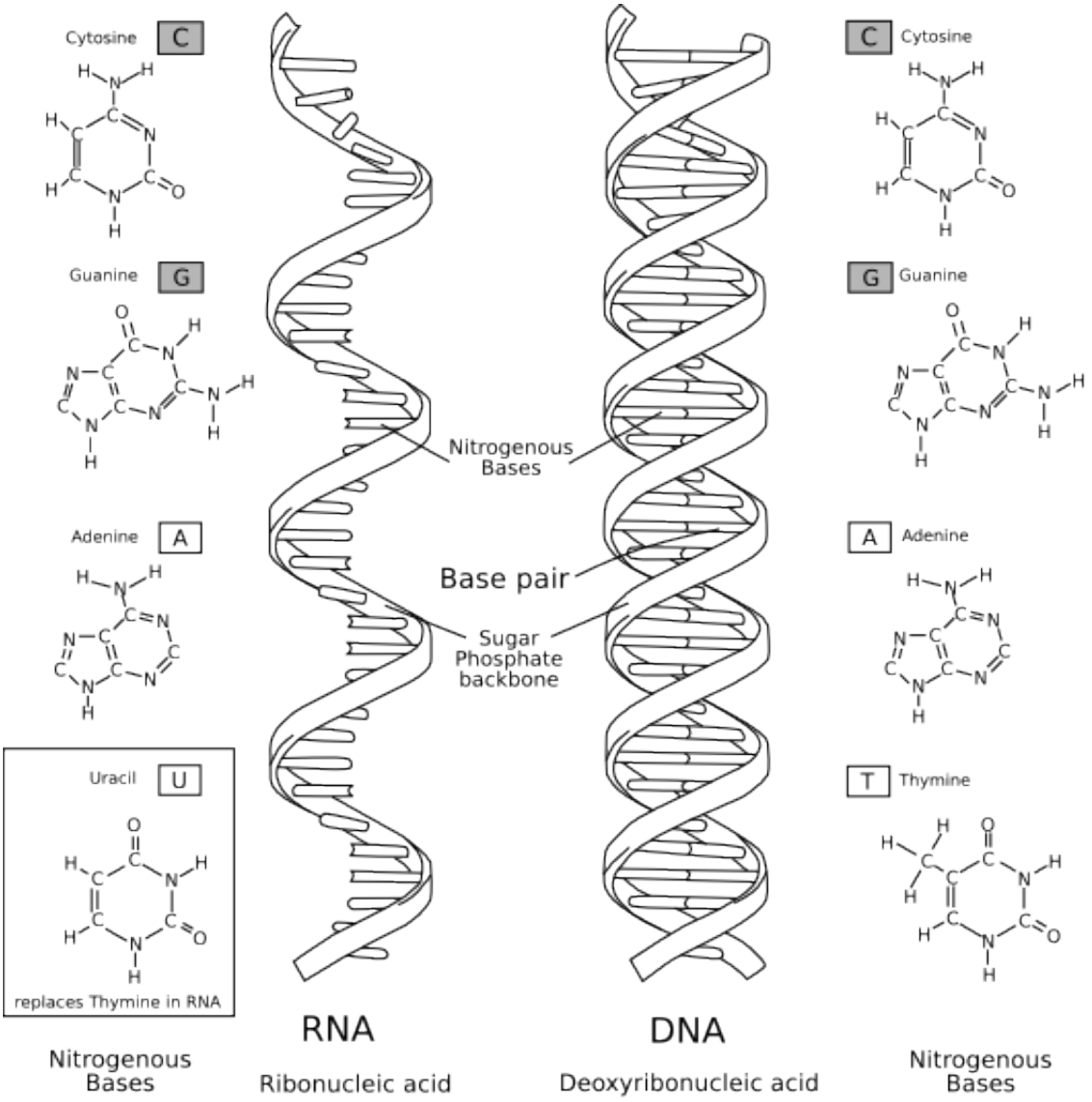
All matter is made up of atoms, and all atoms are made up of three main particles: protons, neutrons, and electrons. Protons are positively charged, neutrons are uncharged and electrons are negatively charged. The negative charge of one electron balances the positive charge of one proton. Both protons and neutrons have a mass of 1, while electrons have almost no mass. The element hydrogen has the simplest atoms, each with 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 that help to hold the nucleus together.

Electrons orbiting around the nucleus of an atom are arranged in shells. The first shell can hold only two electrons, while the second shell holds up to eight electrons. Subsequent shells can hold more electrons, but the outermost shell of any atom holds no more than eight electrons. The following table will summarize the number of electrons in each shell for some elements:

The different composition of orbiting electrons allow to represent elements with the following ...

7.2 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 7.1 — 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 7.1 — Chemistry of love - pt1. Dopamine is thought to be the pleasure chemical, producing a feeling of bliss

Challenge 7.2 — 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 7.2 — Chemistry of love - pt2. estrogen and testosterone play a role in the sex drive area

Challenge 7.3 — 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 7.3 — Chemistry of love - pt3. Norepinephrine is similar to adrenaline and produces the racing heart and excitement

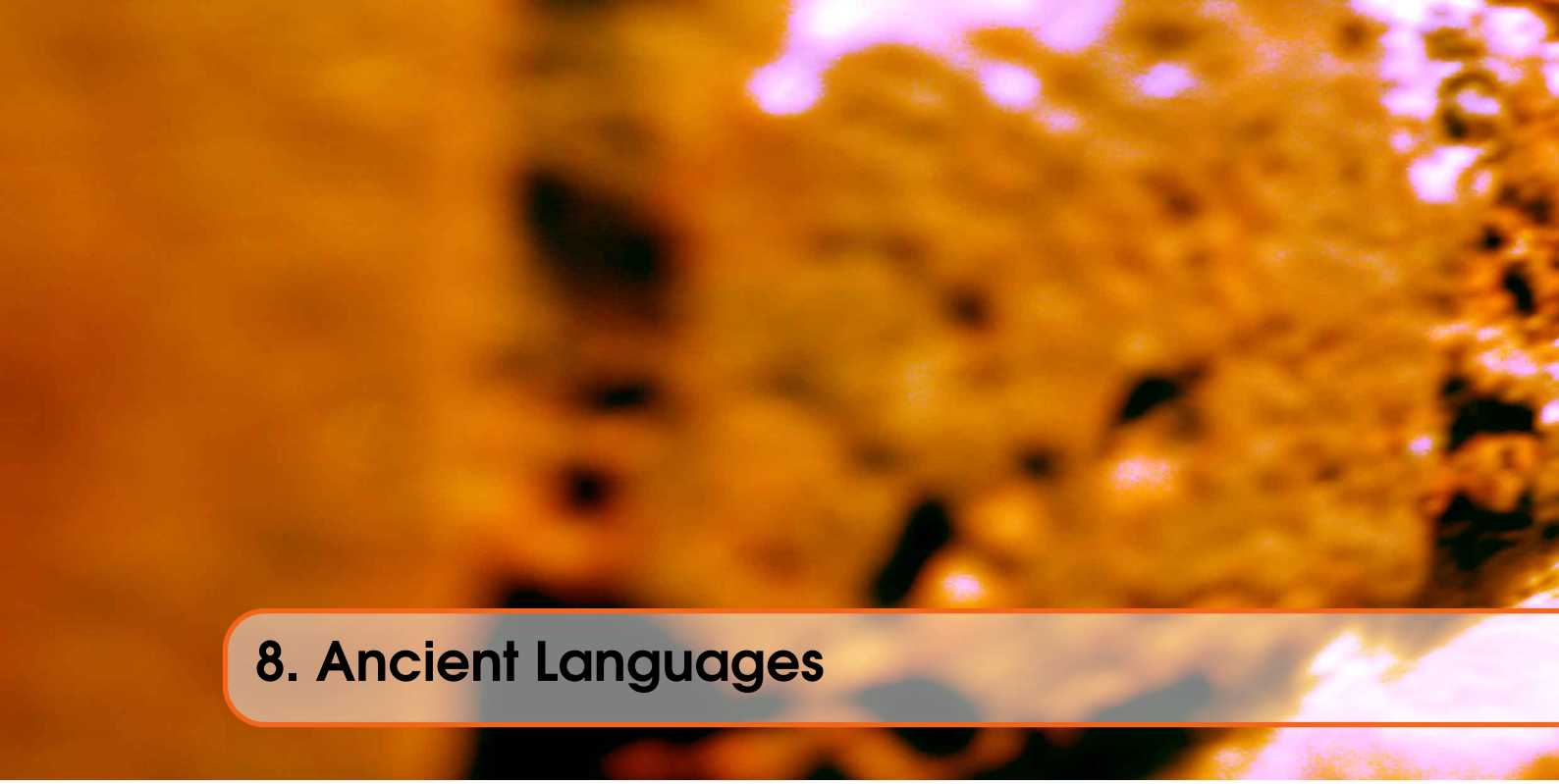
III

Encoding

8 Ancient Languages *(.5pc).47
8.1 Runic alphabets (150 A.D.)

9 Fictional Languages *(.5pc).51
9.1 Messages from the Space
9.2 Fantasy Messages

10 Useful encoding *(.5pc).55
10.1 Semaphore (1794)
10.2 Braille (1824)
10.3 Morse Code (1835 - 1840)
10.4 ASCII (1960 - 1967)
10.5 DTMF (1963)
10.6 BaseN Encodings (1993)



8. 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

TODO Egypt
TODO Babylonian math

8.1 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.

8.1.1 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 8.1 — ruined. ᛏᚺᛖᚾ ᚠᚱᚴ ᛁᛊ ᚱᚢᛁᚾᛖᛞ, ᚠᛊ ᚲᚨᚾ ᛒᛖ ᛊᛖᛖᚾ ᛁᚾ ᛞᛁᚠᚠᛖᚱᛖᚾᛏ ᛈᛚᚨᚲᛖᛊ ᛁᚾ ᚨᛗᛖᚱᛁᚲᚨ, ᚠᛟᚱ ᛁᚾᛊᛏᚨᚾᚲᛖ.

Solution 8.1 — ruined. Then all is ruined, as can be seen in different places in America, for instance.

8.1.2 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 8.2 — destroyed. ᚠᚠᚱ ᛖᚠᚳᚻ ᛖᛉᚳᛖᚳᛏᚠᛏᛁᚠᚾ ᛏᚻᚠᛏ ᚻᛖ ᚠᚢᛚᚠᛁᛚᛚᛖᛞ, ᛏᚻᛖᚱᛖ ᚹᚠᛋ ᚠᚾᚠᛏᚻᛖᚱ ᛏᚻᚠᛏ ᚻᛖ ᛞᛖᛋᛏᚱᚠᚢᛖᛞ. ■

Solution 8.2 — destroyed. For each expectation that he fulfilled there was another that he destroyed.

8.1.3 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	ᚬ	ˎ	, o, æ	As	"???"
ᚱ	ᚱ	(	r	reið	"ride"
ᚴ	ᚴ	ı	k,g	kaun	"ulcer"
ᚼ	ᚽ	ı	h	hagall	"hail"
ᚾ	ᚿ	ˎ	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	ˏ	t, d	Týr	"???"
ᛒ	ᛓ	ˏ	b, p	björk/bjarkan/bjarken	"birch"
ᛘ	ᛙ	:	m	maðr	"man"
ᛚ	ᛚ	ˎ	l	lögr	"sea"
ᛦ	ᛧ	:	R	yr	"yew"

Challenge 8.3 — damaged. ᛘᛅᚾ ᛁ ᚴᚬᛏ ᛏᛅᛘᛅᚴᛁᛏ ᛏᚬᛏᛅᚢ, ᛅᚾᛏ ᛁᛏ ᚼᚢᚱᛏ ᛚᛁᚴᛅ ᚼᛁᛚᛚ.

Solution 8.3 — damaged. Man, I got damaged today, and it hurt like hell.

Challenge 8.4 — broken. ᚿᚭᛐᚽᛁᚿᚴ ᛓᚱᛆᚴᛌ ᛚᛁᚴ ᛆᚿ ᚽᛆᚱᛐ.

Solution 8.4 — broken. nothing breaks like a heart

Challenge 8.5 — disturbed. ı)ˏˈ |ˎˏ(ˎˏˎˎ ˏ|ˈˏ)(ˏˏ ˏ ıˏ(ˎ| ı(|ˎˎˎˈˈ.

Solution 8.5 — disturbed. He was internally disturbed to hear of her illness

9. 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. The following is only a list of some fictional alphabets with a really short description of their usages.

9.1 Messages from the Space

9.1.1 Aurebesh (Star Wars)

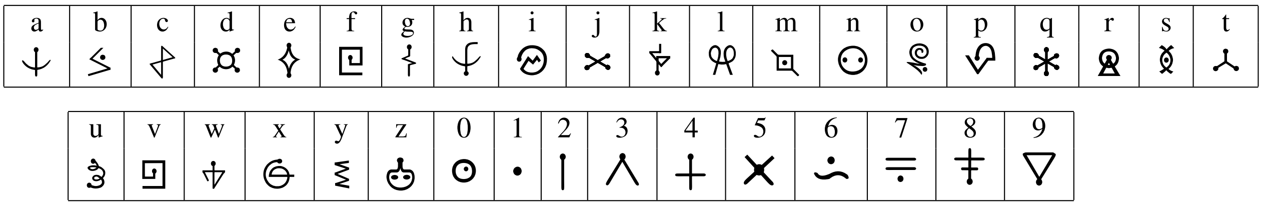
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r
																	

s	t	u	v	w	x	y	z	0	1	2	3	4	5	6	7	8	9
																	

Challenge 9.1 — Light Speed? Really?. 1↓ 1N ↓ΞVI NΞ1U ↓Ξ<K↓ ΛK7VI ↓ΞVI ⊐VINNVIN 7UOΛ 1Λ NVINN ↓Ξ<OΛ ↓OVINYVI UK7NVI'IN.. 1 Ξ<YVI ΔUO↓– 7UOΛ IΛUVI71<N NV↓K7NΞ1UN.. ΛJΔ↓ ↓ΞVI NΔ'I<N ΞUNO 'I7U1NVI7N. Λ1Λ7 VΔU.. 1 <Λ ↓<NO1Λ7 <ΞΔUO↓ ↓ΞVI Ξ17 'IΔ7VINN1<Λ NΞ1UN. ΛΔO.. NΞVI 1N ⫡<NN↓ VIΛΔUU7Ξ ⫡Δ7 VΔU. Δ↓7 Λ<Λ..

Solution 9.1 — 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.

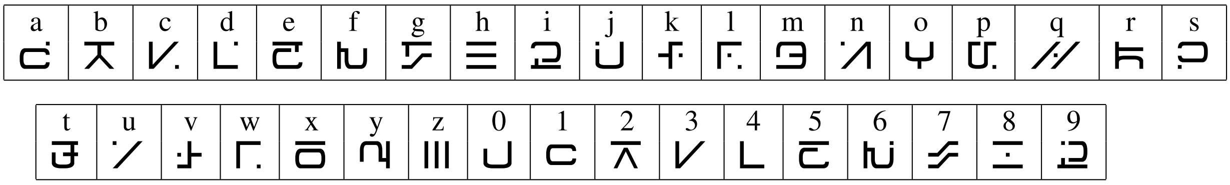
9.1.2 Futurama Alien Alphabet (Used in Futurama cartoon)



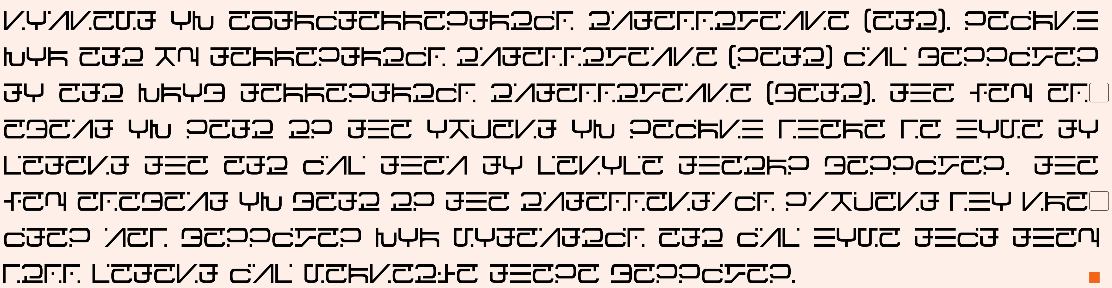
Challenge 9.2 — Less than Hero.

Solution 9.2 — 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.

9.1.3 Visitor (Used in TV series of 2009)

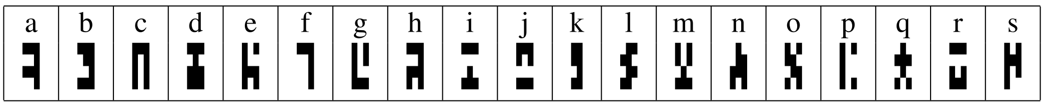


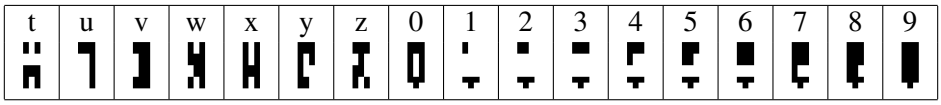
Challenge 9.3 — Problems in communications.

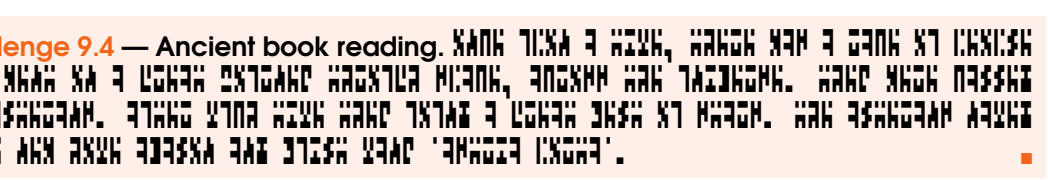


Solution 9.3 — Problems in communications. There are two problems in concept of extraterrestrial intelligence (ETI). Search for ETI by terrestrial intelligence (SETI) and Messages to ETI from terrestrial intelligence (METI). The key element of SETI is the Object of search where we hope to detect the 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.

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

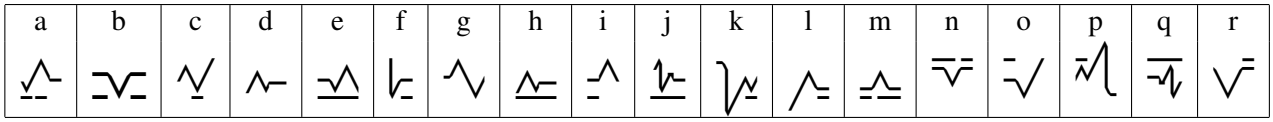


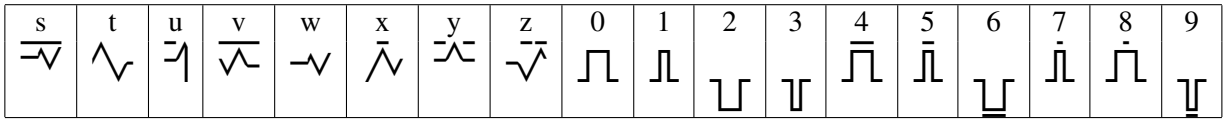


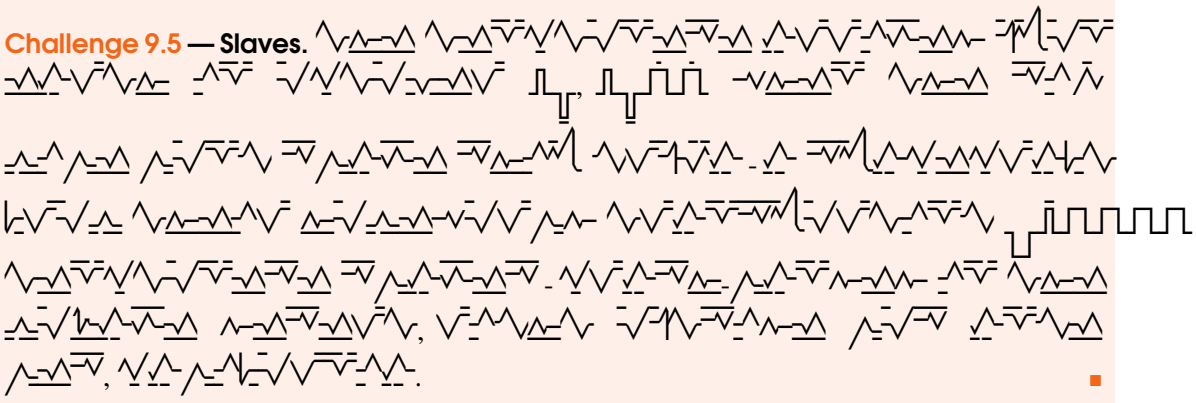
Challenge 9.4 — Ancient book reading. 

Solution 9.4 — 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'.

9.1.5 Tenctonese (Alien Nation)





Challenge 9.5 — Slaves. 

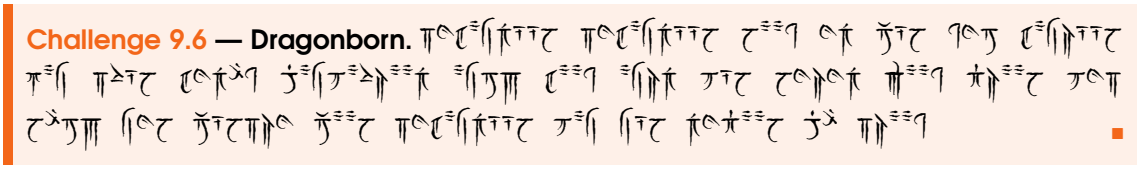
Solution 9.5 — Slaves. The Tenctonese arrived upon Earth in October 19, 1988 when the six-mile-long slave ship Gruza - a spacecraft from their homeworld transporting 250,000 Tenctonese slaves - crash-landed in the Mojave desert, right outside Los Angeles, California.

9.2 Fantasy Messages

9.2.1 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 9.6 — Dragonborn. 

Solution 9.6 — 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

9.2.2 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 9.7 — Dragonborn.

Solution 9.7 — 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.

10. Useful encoding

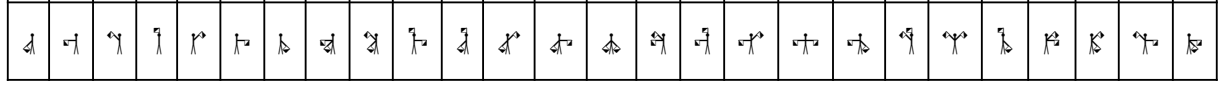
Encoding is referred to the process of converting data into a format required for a number of information processing needs. To not be confused with the concept of encrypting that, differently, aim to hide the original information instead of transforming in order to maximize its coverage and usage.

10.1 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.

10.1.1 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 10.1 — The Flag Man. 

Solution 10.1 — The Flag Man. The big ship sails on the alley-alley-o

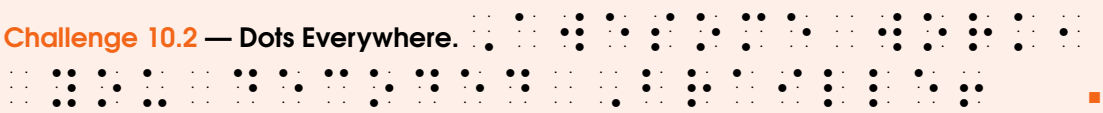
10.2 Braille (1824)

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.

10.2.1 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 10.2 — Dots Everywhere. 

Solution 10.2 — Dots Everywhere. Awesome work, you decoded Braille!

10.2.2 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 10.3 — Less Dots, Longer Words.

Solution 10.3 — Less Dots, Longer Words. In this way lot of space has been saved.

10.2.3 Grade 3

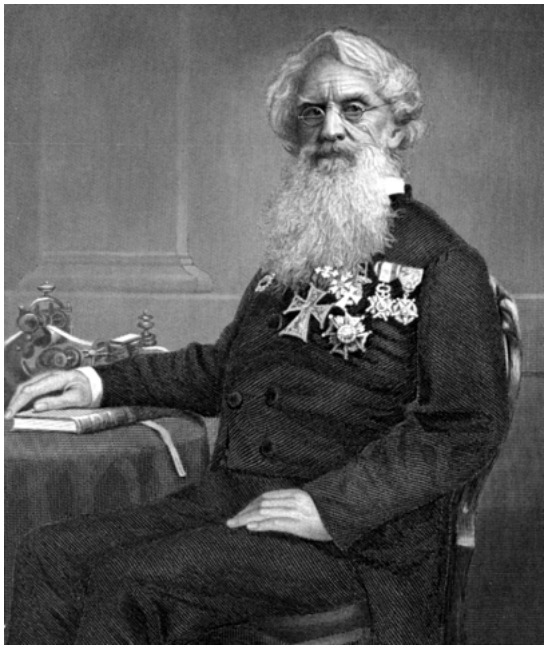
More sophisticate contractions has been developed for every language. There are no official Grade 3 encoding, but can be customized depending on the behaviors. The main idea is to use the first few letters to represent a whole word. What follows is an example of this process:

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.

10.3 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 10.4 Decode: . -..- .- – .–. .-.. . — ..-. – — .-. ... . -.-. — -.. . ■

Solution 10.4 Decoded: example of morse code

10.4 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 10.5 Decode: something in bin ■

Solution 10.5 Decoded: example of morse code

Challenge 10.6 Decode: something in hex ■

Solution 10.6 Decoded: example of morse code

10.4.1 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 10.7 Decode: something in oct ■

Solution 10.7 Decoded: example of morse code

Challenge 10.8 Decode: something in hex ■

Solution 10.8 Decoded: example of morse code

10.5 DTMF (1963)

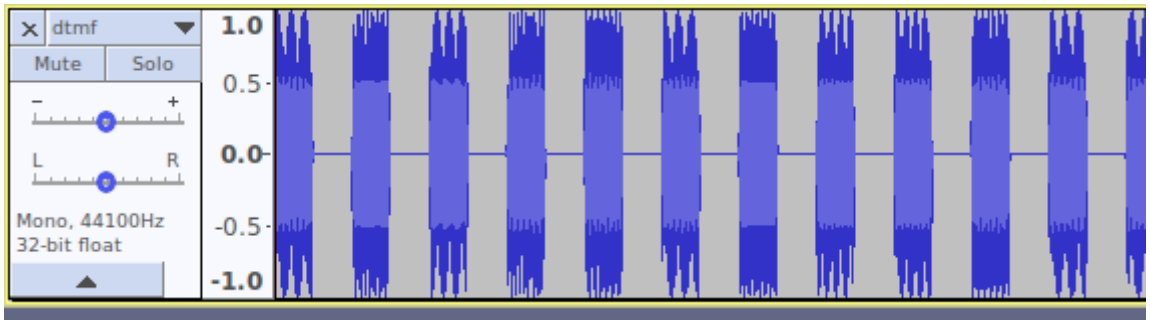
Since the invention of the telegraph in 1837 by Cooke, and next of the telephone in 1849 by Meucci, the main problem was to define an encoding system able to ensure communication between 2 communication devices. One of the first attempt, used since the second half of the 20th century was based on Pulse 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.

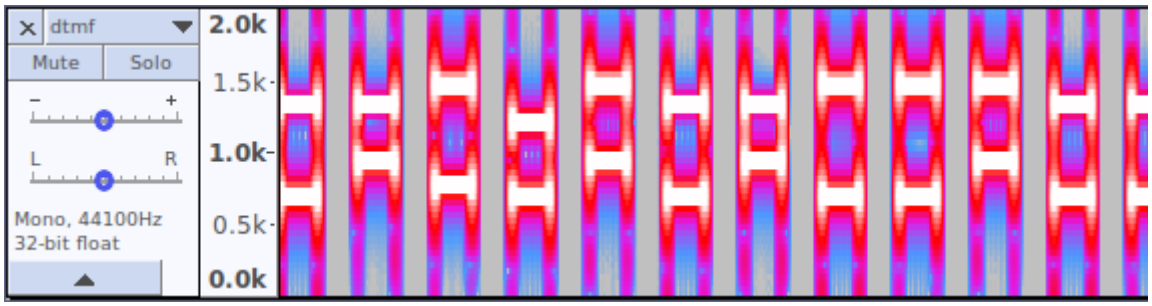
In 1963 engineers at Bell developed a new kind of technology "called Dual-Tone multi-frequency signaling (DTMF)" whose become the new communication standard. The new telephones had no rotors, but a push able 16-keys keypad, friendly called Touch-Tone. This new system allowed to make long-rage calls and 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

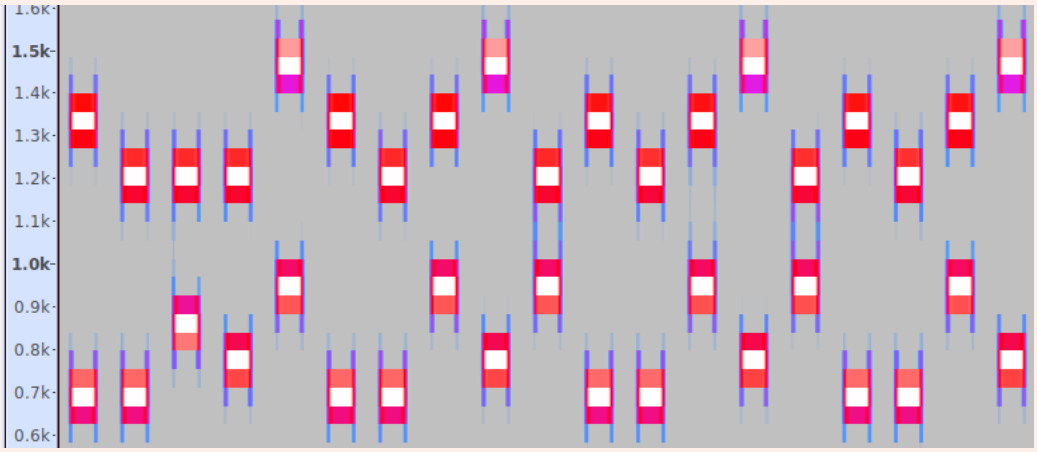
The decoding system was provided by filter banks first, and by digital signal processing then, usually using the Goertzel algorithm. Frequencies of tones of a DTMF communication can be visually seen by inspecting the spectogram of the signal itself:





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 10.9 Can you decode this DTMF signal?



Solution 10.9 Decoded: 2174#2106*2106*2106

10.6 BaseN Encodings (1993)

10.6.1 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 10.10 — superego. What have Freud to say?
FrQXQgdGhlIHRpbWUgYXQgd2hpY2ggdGhlIE9lZGlwdXMgY2
9tcGxleCBnaXZlcyBwbGFjZSB0byB0aGUgc3VwZXItZWdvIH
RoZXkgYXJlIHNvbWV0aGluZyBxdWl0ZSBtYWduaWZpY2VudDs= ■

Solution 10.10 — superego. Such egocentric patients
At the time at which the Oedipus complex gives place to the super-ego they are something quite magnificent

10.6.2 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	0110011101100101xxxxxxxxxxxxxxxxxxxxxxxx

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	1xxxx	xxxxx	xxxxx	xxxxx	xxxxx
Normalized	01100	11101	10010	10000	xxxxx	xxxxx	xxxxx	xxxxx
ASCII	12	29	18	16	?	?	?	?
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 10.11 — ego. Tell me Sigmund!
KRUGKIDFM5XSA2LTEBXG65BAONUGC4TQNR4SA43FOBQXEYLU
MVSCAZTSN5WSA5DIMUQGSZB3EBUXI4ZANRXXOZLSEBYG64TU
NFXW4IDNMVZGOZLTEBUW45DPEBUXILROFYXCAQTVOQQHI2DF
EBZGK4DSMVZXGZLEEBWWK4THMVZSA2LOORXSA5DIMUQGSZBA
MFZSA53FNRWCYIDBNZSCA2LTEBWWK4TFNR4SAYJAOBQXE5BA
N5TCA2LUFYQFI2DFEBZGK4DSMVZXGZLEEBUXGIDPNZWHSIDD
OV2CA33GMYQHG2DBOJYGY6JAMZZG63JAORUGKIDFM5XSAYTZ
EB2GQZJAOJSXG2LTORQW4Y3FOMQG6ZRAOJSXA4TFONZWS33O
HMQGS5BAMNQW4IDDN5WW25LONFRWC5DFEB3WS5DIEB2GQZJA
MVTW6IDUNBZG65LHNAQHI2DFEBUWI===

Solution 10.11 — 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

IV

Cryptography

11 Classical Cipher *(.5pc).71
11.1 Rot-n
11.2 Substitution Ciphers
11.3 Polygraphic Substitution Ciphers

12 Modern Cryptography *(.5pc).79
12.1 RSA (1977)
12.2 Rabin Cryptosystem (1979)
12.3 Block Ciphers (1981)

13 Cryptanalysis *(.5pc).87
13.1 Frequency Analysis

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();
}
```

11.2 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.

11.2.1 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:

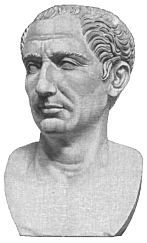
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
i	h	g	f	e	d	c	b	a	r	q	p	o	n	m	l	k	j	z	y	x	w	v	u	t	s

```
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

11.2.2 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:

$$E_n(x) = (x+n) \pmod{26} \tag{11.3}$$

And the decryption algorithm will be:

$$D_n(x) = (x-n) \pmod{26} \tag{11.4}$$

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 11.5 Decode: Ymnx mfx gjjs jshwduyji bnym Hjfxfw hnumjw ■

Solution 11.5 Decoded: This has been encrypted with Ceasar cipher

11.2.3 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:

$$E_{a,b}(x) = (a*x+b) \pmod{26} \tag{11.5}$$

$$D_{a,b}(x) = a^{-1}*(x-b) \pmod{26} \tag{11.6}$$

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

$$1 = a*a^{-1} \pmod{26} \tag{11.7}$$

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.

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:

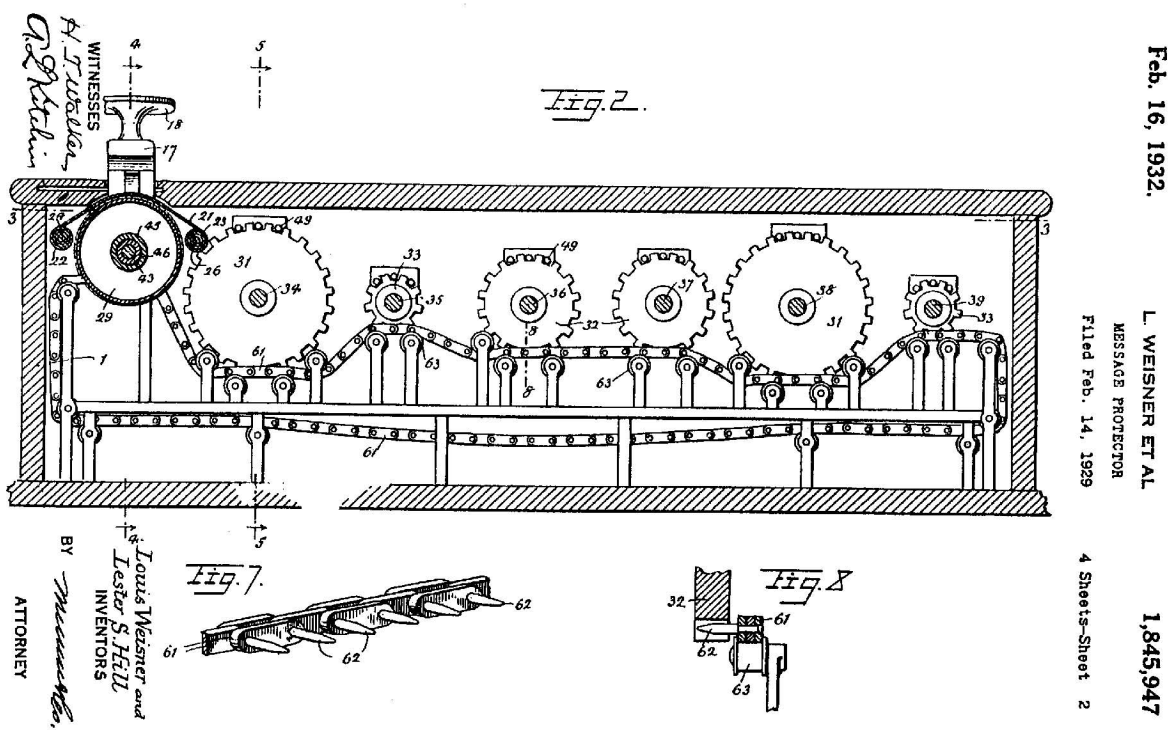
ciphertext	r	g	g	p	e	d	x	p	k	m	d	q
numerical x	17	6	6	15	4	3	23	15	10	12	3	16
$9*(x-17)$	0	-99	-99	-18	-117	-126	54	-18	-63	-45	-126	-9
$9*(x-17) \pmod{26}$	0	5	5	8	13	4	2	8	15	7	4	17
plaintext	a	f	f	i	n	e	c	i	p	h	e	r

11.3 Polygraphic Substitution Ciphers

what is, TODO

11.3.1 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):

$$\begin{bmatrix} e & n & c \\ o & d & i \\ n & g & s \end{bmatrix} = \begin{bmatrix} 4 & 13 & 2 \\ 14 & 3 & 8 \\ 13 & 6 & 18 \end{bmatrix} \tag{11.8}$$

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

$$cryptogrphy = \begin{bmatrix} c \\ r \\ y \end{bmatrix} \begin{bmatrix} p \\ t \\ o \end{bmatrix} \begin{bmatrix} g \\ r \\ a \end{bmatrix} \begin{bmatrix} p \\ h \\ y \end{bmatrix} = \begin{bmatrix} 2 \\ 17 \\ 24 \end{bmatrix} \begin{bmatrix} 15 \\ 19 \\ 14 \end{bmatrix} \begin{bmatrix} 6 \\ 17 \\ 0 \end{bmatrix} \begin{bmatrix} 15 \\ 7 \\ 24 \end{bmatrix} \tag{11.9}$$

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

$$\begin{bmatrix} 4 & 13 & 2 \\ 14 & 3 & 8 \\ 13 & 6 & 18 \end{bmatrix} * \begin{bmatrix} 2 \\ 17 \\ 24 \end{bmatrix} (mod \quad 26) = \begin{bmatrix} 13 \\ 23 \\ 1 \end{bmatrix} = \begin{bmatrix} n \\ x \\ b \end{bmatrix} \tag{11.10}$$

$$\begin{bmatrix} 4 & 13 & 2 \\ 14 & 3 & 8 \\ 13 & 6 & 18 \end{bmatrix} * \begin{bmatrix} 15 \\ 19 \\ 14 \end{bmatrix} (mod \quad 26) = \begin{bmatrix} 23 \\ 15 \\ 15 \end{bmatrix} = \begin{bmatrix} x \\ p \\ p \end{bmatrix} \tag{11.11}$$

$$\begin{bmatrix} 4 & 13 & 2 \\ 14 & 3 & 8 \\ 13 & 6 & 18 \end{bmatrix} * \begin{bmatrix} 6 \\ 17 \\ 0 \end{bmatrix} (mod \quad 26) = \begin{bmatrix} 11 \\ 5 \\ 24 \end{bmatrix} = \begin{bmatrix} l \\ f \\ y \end{bmatrix} \tag{11.12}$$

$$\begin{bmatrix} 4 & 13 & 2 \\ 14 & 3 & 8 \\ 13 & 6 & 18 \end{bmatrix} * \begin{bmatrix} 15 \\ 7 \\ 24 \end{bmatrix} (mod \quad 26) = \begin{bmatrix} 17 \\ 7 \\ 24 \end{bmatrix} = \begin{bmatrix} r \\ h \\ y \end{bmatrix} \tag{11.13}$$

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:

$$\begin{bmatrix} b & e \\ e & r \end{bmatrix} = \begin{bmatrix} 1 & 4 \\ 4 & 17 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 4 \\ 4 & 17 \end{bmatrix}^{-1} = \begin{bmatrix} 17 & -4 \\ -4 & 1 \end{bmatrix} (mod \quad 26) = \begin{bmatrix} 17 & 22 \\ 22 & 1 \end{bmatrix} \tag{11.14}$$

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

$$\begin{bmatrix} 17 & 22 \\ 22 & 1 \end{bmatrix} * \begin{bmatrix} f \\ i \end{bmatrix} \rightarrow \begin{bmatrix} 17 & 22 \\ 22 & 1 \end{bmatrix} * \begin{bmatrix} 5 \\ 8 \end{bmatrix} (mod \quad 26) = \begin{bmatrix} 1 \\ 14 \end{bmatrix} = \begin{bmatrix} b \\ o \end{bmatrix} \tag{11.15}$$

$$\begin{bmatrix} 17 & 22 \\ 22 & 1 \end{bmatrix} * \begin{bmatrix} c \\ s \end{bmatrix} \rightarrow \begin{bmatrix} 17 & 22 \\ 22 & 1 \end{bmatrix} * \begin{bmatrix} 2 \\ 18 \end{bmatrix} (mod \quad 26) = \begin{bmatrix} 14 \\ 10 \end{bmatrix} = \begin{bmatrix} o \\ k \end{bmatrix} \tag{11.16}$$

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

12. Modern Cryptography

Algoritmi a chiave pubblica cosa è? idea...

12.1 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:

$$c \equiv m^e \pmod{n} \tag{12.1}$$

$$m \equiv c^d \pmod{n} \tag{12.2}$$

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 $d \equiv e^{-1} (mod \quad \lambda(n))$

With:

- $\lambda(n)$ is the Charmichael function defined as the smallest positive integer m such that $a^m \equiv 1 (mod \quad n)$ 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: $m \equiv 382506^{608693} (mod 200592) = 18722$

12.1.1 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 9388268292017501827143572996717724504865 9 that are both primes.

12.1.2 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:

$$N = a^2 - b^2 \tag{12.3}$$

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

$$a^2 - b^2 = (a+b) * (a-b) \tag{12.4}$$

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

$$N = \left(\frac{c+d}{2}\right)^2 - \left(\frac{c-d}{2}\right)^2 \tag{12.5}$$

$$a = \frac{c+d}{2} \tag{12.6}$$

$$b = \frac{c-d}{2} \tag{12.7}$$

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 (12.6) and (12.7) to find d and c:

$$\begin{cases} a = \frac{c+d}{2} \\ b = \frac{c-d}{2} \end{cases} \qquad \begin{cases} c = a + b \\ d = a - b \end{cases} \tag{12.8}$$

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 $b = 97$:

$$\begin{cases} c = 1470 + 97 \\ d = 1470 - 97 \end{cases} \qquad \begin{cases} c = 1567 \\ d = 1373 \end{cases} \tag{12.9}$$

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

12.1.3 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:

$$C_A = M_A^{e_A} \tag{12.10}$$

$$C_B = M_B^{e_B} \tag{12.11}$$

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:

$$e_A * u + e_B * v = gcd(e_A, e_B) = 1 \tag{12.12}$$

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:

$$a = q_0 * b + r_0 \tag{12.13}$$
$$b = q_1 * r_0 + r_1 \tag{12.14}$$
$$r_0 = q_2 * r_1 + r_2 \tag{12.15}$$
$$\ldots \tag{12.16}$$
$$r_{n-2} = q_n * r_{n-1} + r_n \tag{12.17}$$
$$r_{n-1} = q_{n+1} * r_n + 0 \tag{12.18}$$

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:

$$r_n = r_{n-2} - q_n * r_{n-1} \tag{12.19}$$
$$\ldots \tag{12.20}$$
$$r_2 = r_0 - q_2 * r_1 \tag{12.21}$$
$$r_1 = b - q_1 * r_0 \tag{12.22}$$
$$r_0 = a - q_0 * b \tag{12.23}$$

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

R 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

$$71 - 53 - 1 * (53 - 2 * 71 + 2 * 53) = 1 \tag{12.24}$$
$$71 - 53 - 53 + 2 * 71 - 2 * 53 = 1 \tag{12.25}$$
$$3 * 71 - 4 * 53 = 1 \tag{12.26}$$

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 (12.10) and (12.11) can be rewritten as:

$$C_A^u = M_A^{e_A^u} = M^{e_A * u} \tag{12.27}$$

$$C_B^v = M_B^{e_B^v} = M^{e_B * v} \tag{12.28}$$

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

$$M^{e_A * u} * M^{e_B * v} = M^{e_A * u + e_B * v} = M^1 = M \tag{12.29}$$

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

$$M = C_A^u * C_B^v \tag{12.30}$$

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

$$M = C_A^u * C_{B,inv}^{-v} \tag{12.31}$$

12.1.4 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:

$$M = \sqrt[e]{C} \tag{12.32}$$

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)

12.1.5 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:

$$\begin{cases} C_1 \equiv M_1^e (mod \quad N_1) \\ C_2 \equiv M_2^e (mod \quad N_2) \\ xxx \\ C_e \equiv M_e^e (mod \quad N_e) \end{cases} \tag{12.33}$$

Solving this system is trivial because the chinese remainder theorem comes in help, telling us that for (12.33) 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:

$$C \equiv M^e (mod \quad N_1, N_2, ..., N_e) \tag{12.34}$$

However, since for short messages $M < N_i \bigvee i$, then also $M^e < N_1 * N_2 * ... * N_e$ and (12.41) can be rewritten as:

$$C \equiv M^e \tag{12.35}$$

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

12.1.6 Wiener's Attack

12.2 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:

$$c \equiv m^2 (mod \quad n) \tag{12.36}$$

$$misoneof \begin{cases} M_1 = a * p * m_q + b * q * m_p \\ M_2 = a * p * m_q - b * q * m_p \\ M_3 = -a * p * m_q + b * q * m_p \\ M_4 = -a * p * m_q - b * q * m_p \end{cases} \tag{12.37}$$

What does it mean? Analysing (12.37) 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 (12.36), but how the decryption algorithm works?
In order to decrypt Rabin, the following problem needs to be solved:

$$m^2 \equiv c (mod \quad N) \tag{12.38}$$

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 (12.38). To do so we need to compute first:

$$m_p = \sqrt{c} (mod \quad p) \tag{12.39}$$

$$m_q = \sqrt{c} (mod \quad q) \tag{12.40}$$

$$a * p + b * q = 1 \tag{12.41}$$

In particular (12.41) can be computed using the extended Euclidean algorithm showed in (12.12). Finally the Chinese remainder algorithm is used to compute the final roots showed in (12.37). Which between M_1, M_2, M_3 and M_4 is the correct plaintext is usually unknown at prior.

12.3 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.

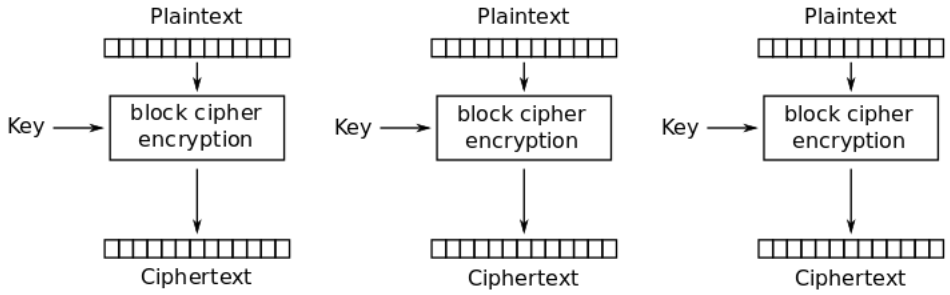
12.3.1 Electronic Codebook (ECB)

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):

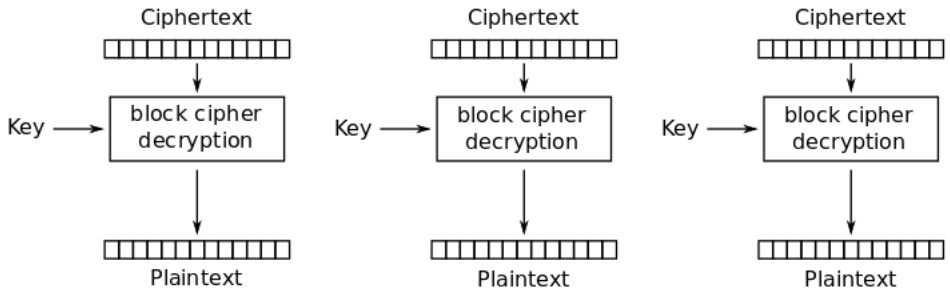
$$C_i = F_{enc}(B_i, K) \tag{12.42}$$

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

$$B_i = F_{dec}(C_i, K) \tag{12.43}$$



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.

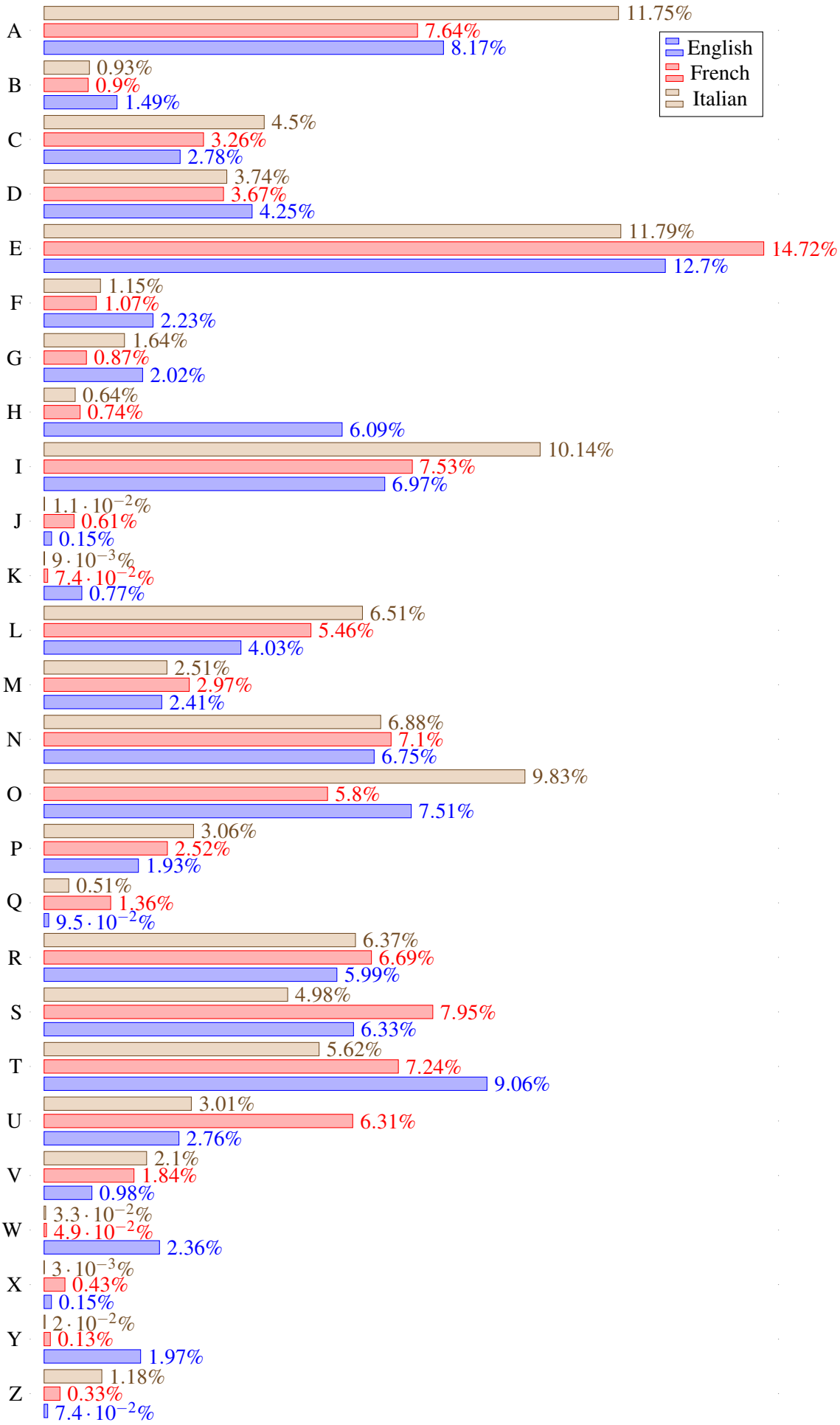
13. Cryptanalysis

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.

13.1 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.


English
French
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%
$7.4 \cdot 10^{-2}\%$

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

13.1.1 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 13.1 — The order matter. This text hides a secret, i am pretty sure...
FrFErFYFrFEFrqFEFrFUrFEFrFqFrEFUFrEFrFqFreFrFEFqrFEFrFUrFEFrFqcrFEFrFeEF
rFqrFUFrFErFqFrFEFreFcrFEFUrFErFqFrFcEFrFqFrFErFNFrFEFqrFEFrFUrFqFrFEeFr
FErFqFrFErFUFrqFrFEFNrFEFrFUFrEFqrFeFrEFrqrFErFUFErFqFrNFrFEFqrFEFrFUFrE
FeFrqFrFEFcFrEFrFqFrFEFrUFrFFqFFrEeFrFEFrFqFrEFUFrFqrErNErUFqFrFrFeFEFrq
FrEFYrFqFrEFUFrEFrFqrFEeFrqrEFUrFEFrFYFrEFqFrFUFEFrFqrFEFeFqFErFUFrFEFqr
FEFrFNFUrFEFrFqFEeFEFrFqrFcFrFEqFrFEFUrFEFeFrFqFrEFUFrFEFrqNFrFqFrEFrFUF
EFcrFqFEFrFUrFEFqFrEFrFeFrEFqFrFNrFEFrFUEFrFqFrFEeFrFqEFrFUFrFErFqFEFrYF
rFqFEFrFUFrFEqFrFEFrFeFqrFEFrFUEFrFqEFrFNeFEFrqFrFEFUrEFrqFYFrFEFrqFrFEF
UFrEFrFeFrqFrFEFUrFEFrFqrFNFEFrqFrFEFrFUEFrFeFrEFqFrcFEFrFqrFEFUrFqFrEFr
FNFrEFUFrFEFqFreFrFEqFrFEFrFUrFE ■

Solution 13.1 — 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

V

Sequences

14 Famous Sequences *(.5pc).93

14.1 Fibonacci Numbers
14.2 Lucas Numbers
14.3 Look-and-say Sequence

15 Figurate Numbers *(.5pc).97

15.1 Triangular Numbers
15.2 Square Numbers

14. 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.

14.1 Fibonacci Numbers

Challenge 14.1 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:

$$Fib(n) = \begin{cases} 0 & \text{n=0} \\ 1 & \text{n=1} \\ Fib(n-1) + Fib(n-2) & \text{n>1} \end{cases} \tag{14.1}$$

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 14.1 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 14.2 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:

$$Fib(n) = \frac{(1+\sqrt{5})^n - (1-\sqrt{5})^n}{2^n * \sqrt{5}} \tag{14.2}$$

Solution 14.2 Thus the solution at the previous challenge is:
$Fib(150) = \frac{(1+\sqrt{5})^{150} - (1-\sqrt{5})^{150}}{2^{150} * \sqrt{5}} = 9969216677189303386214405760200$

14.2 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:

$$Lucas(n) = \begin{cases} 2 & \text{n=0} \\ 1 & \text{n=1} \\ Lucas(n-1) + Lucas(n-2) & \text{n>1} \end{cases} \tag{14.3}$$

Challenge 14.3 Find the missing numbers of the following sequence separating by comma:
2,1,3,4,7,?,18,29,?,76 ... ■

Solution 14.3 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:

$$Lucas(n) = \left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n \tag{14.4}$$

Challenge 14.4 Find the 150th number of the following sequence:
2,1,3,4,7,11,18,29,47,76 ... ■

Solution 14.4 The solution is:
$Lucas(150) = \left(\frac{1+\sqrt{5}}{2}\right)^{150} - \left(\frac{1-\sqrt{5}}{2}\right)^{150} = 22291846172619859445381409012498$

14.3 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

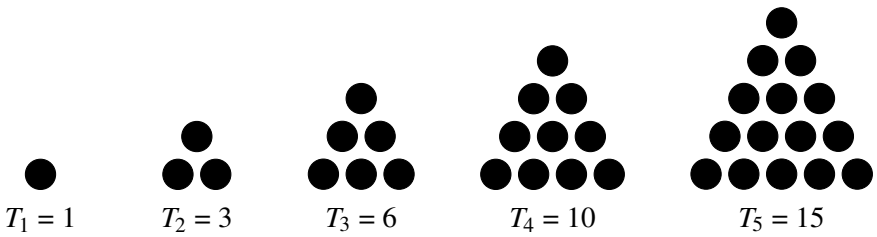
Challenge 14.5 — Just for fun. what is the next number of the sequence? ■

Solution 14.5 — Just for fun. 13211321322113311213211331121113122112132113213211
21113222112311311222113111231133211121321132221131211321 1

15. 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**.

15.1 Triangular Numbers



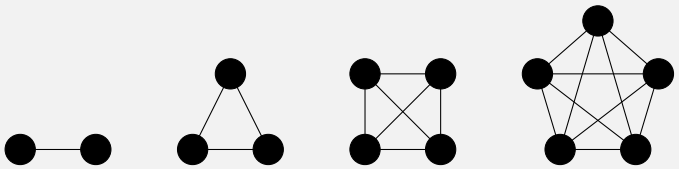
$$T_n = \sum_{k=1}^{n} k \tag{15.1}$$

$$T_n = \frac{n * (n+1)}{2} \tag{15.2}$$

Challenge 15.1 — 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 15.1 — 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:

$$for \quad n \quad delegates \quad we \quad need \quad T_{n-1} \quad interpreters \tag{15.3}$$

To find the number of originally invited delegates we need to find n such that

$$T_{n-1} - T_{n-3} = 31 \tag{15.4}$$

Combining (15.4) with (15.2) we have:

$$\frac{(n-1)*((n-1)+1)}{2} - \frac{(n-3)*((n-3)+1)}{2} = 31 \tag{15.5}$$

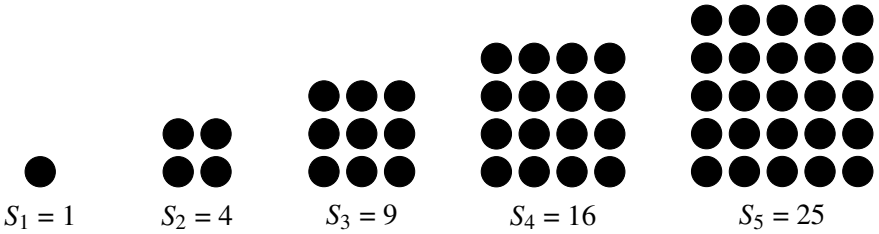
$$\frac{(n-1)*n}{2} - \frac{(n-3)*(n-2)}{2} = 31 \tag{15.6}$$

$$\frac{n^2-n-(n^2-3n-2n+6)}{2} = 31 \tag{15.7}$$

$$\frac{4n-6}{2} = 31 \rightarrow 2n-3 = 31 \rightarrow 2n = 34 \rightarrow \mathbf{n = 17} \tag{15.8}$$

Originally 17 delegates were invited. 2 of them declined the invitation so only 15 delegates will come. Form (15.3) follows that we need $T_{14} = 105$ interpreters.

15.2 Square Numbers



$$T_n = \sum_{k=1}^{n} k \tag{15.9}$$

VI

Geometry

16 Coordinate Systems *(.5pc).101

16.1 Cartesian Coordinate System

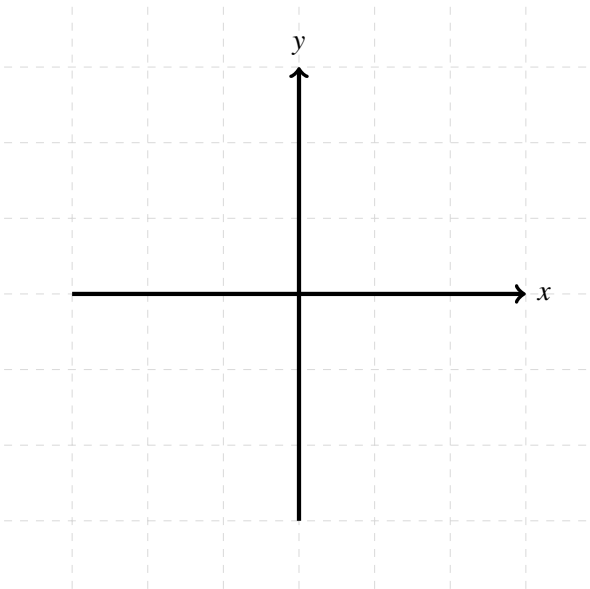
17 Figures *(.5pc).103

17.1 Hexagon

16. Coordinate Systems

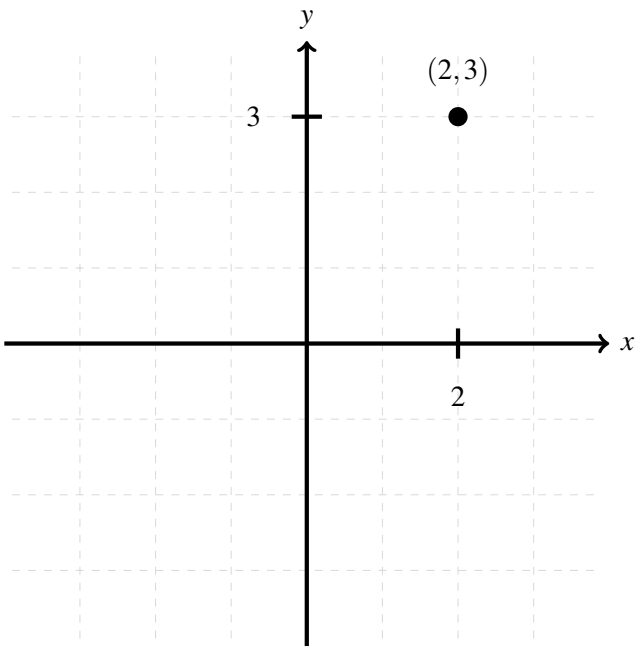
16.1 Cartesian Coordinate System

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. The following figure represent a simple 2-dimensional plane:



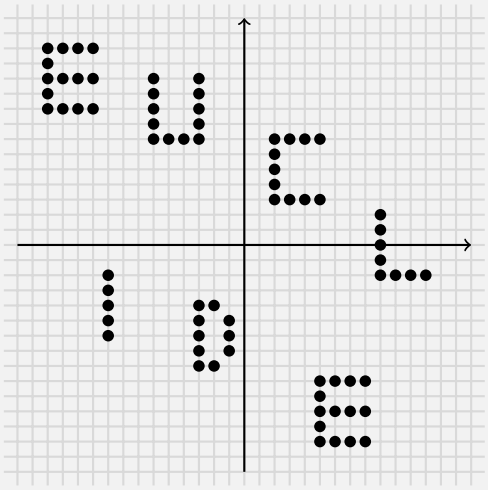
16.1.1 Points

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 16.1 — 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) ■

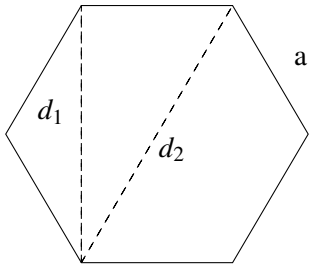
Solution 16.1 — Points. Just draw coordinates:



17. Figures

17.1 Hexagon

Hexagon ...



$$d_1 = \sqrt{3} * a \tag{17.1}$$

$$d_2 = 2a \tag{17.2}$$

$$Perimeter = 6a \tag{17.3}$$

$$Area = \frac{3}{2} * \sqrt{3} * a^2 \tag{17.4}$$

(17.5)

Challenge 17.1 — 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 17.1 — Bees and honey. First of all we must recall that the volume of a regular solid is given by:

$$Volume = h * Area \tag{17.6}$$

So in our case:

$$Volume = 5 * \frac{3}{2} * \sqrt{3} * a^2 \tag{17.7}$$

From the riddle we know that the product between the long diagonal and the short one is 240:

$$d_1 * d_2 = 240 \tag{17.8}$$
$$(\sqrt{3} * a) * (2a) = 240 \tag{17.9}$$
$$2\sqrt{3} * a^2 = 240 \tag{17.10}$$
$$\sqrt{3} * a^2 = 120 \tag{17.11}$$
$$a^2 = \frac{120}{\sqrt{3}} \tag{17.12}$$

Now we can substitute (17.12) into (17.6):

$$Volume = 5 * \frac{3}{2} * \sqrt{3} * \frac{120}{\sqrt{3}} \tag{17.13}$$
$$= 5 * 3 * \sqrt{3} * \frac{60}{\sqrt{3}} \tag{17.14}$$
$$= 5 * 3 * 60 = 900 cm^3 \tag{17.15}$$

Converting to liters we obtain 0.9 liters (1 liter == $1000 cm^3$)

VII

Puzzles

18 Japanese puzzles _____ *(.5pc).107

18.1 Nonogram

19 Mathematical puzzles _____ *(.5pc).109

19.1 Petals Around the Rose

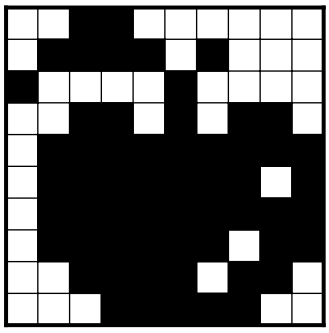
18. Japanese puzzles

18.1 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										

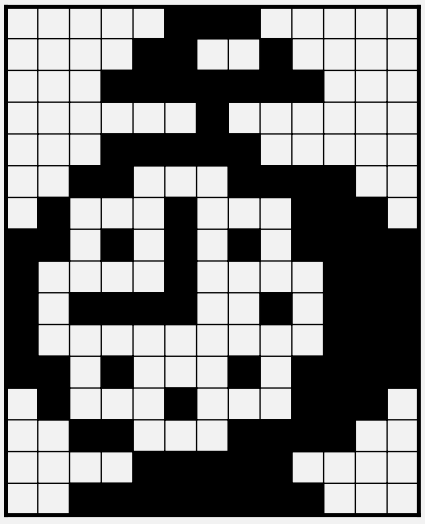


Challenge 18.1 — One more minute please. Solve the following nonogram:

	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 18.1 — One more minute please. Solved:



19. Mathematical puzzles

19.1 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

19.1.1 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.

VIII Programming

20 Esoteric Languages *(.5pc).113
20.1 Brainf*ck Family (1993)

21 Obfuscation *(.5pc).117
21.1 Perl Obfuscation

Bibliography *(.5pc).121
Articles
Books

20. 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.

20.1 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:

```
++++++++[>++++[>++>+++>+++>+<<<<-]>+>+>->>+[<]<-]>>.>---.+++++++..+++.>>.<-.<.
+++.------.--------.>>+.>++.
```

20.1.1 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
```

20.1.2 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

21. Obfuscation

Code obfuscation techniques are very popular nowadays.... What is obfuscation...

21.1 Perl Obfuscation

21.1.1 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,"
```

21.1.2 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
```

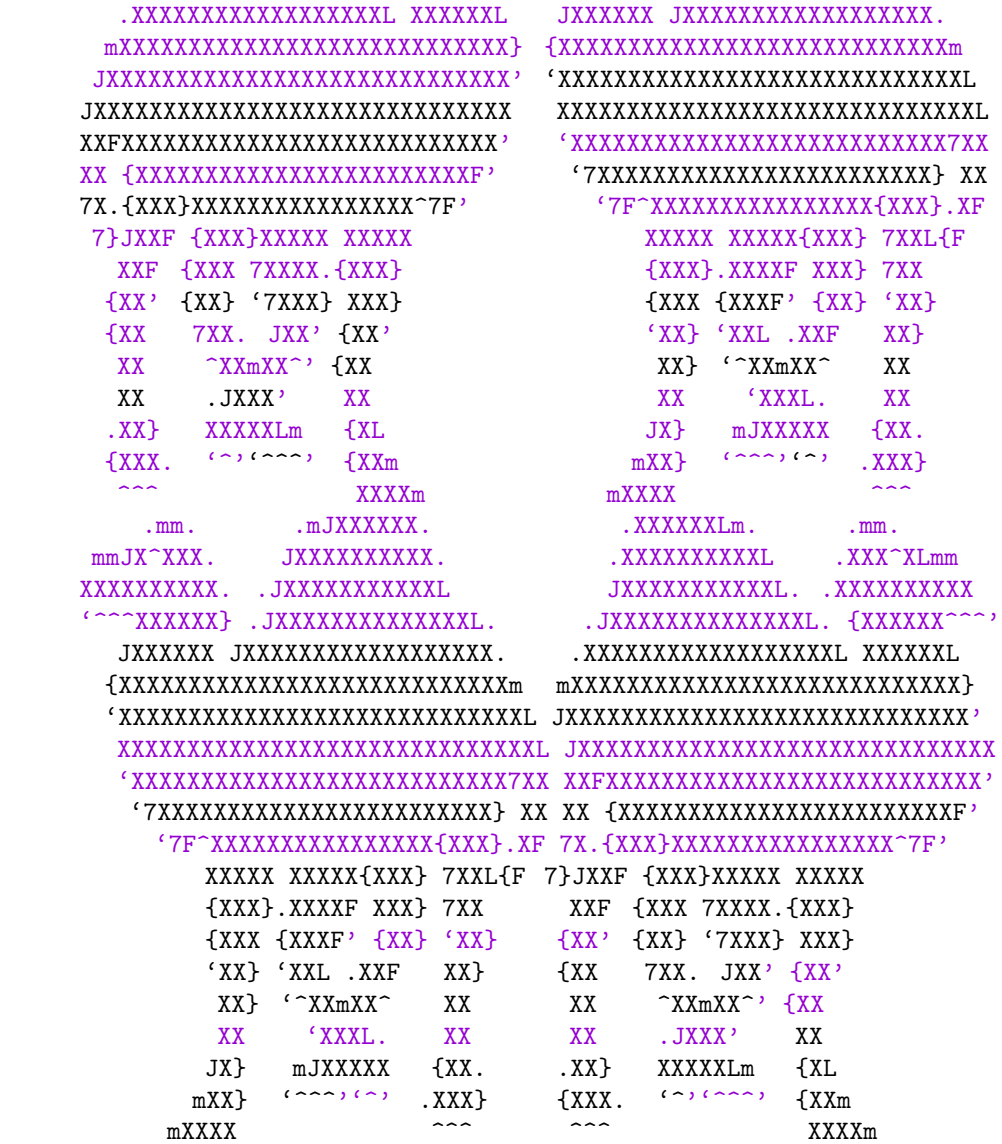
```
    OOOOOOOOOOOOOOO  OOO    DOO      DOODOOOOOO      OODOO
     DOOOO  OOOOO   OOOOOOOOOOOD    OOO  OOOO         OODOO
      OOO    OOO   OOOOODOOOOOOOO   OOO   ODOOOOOOOOOOOOOOO
       0      0   OOOD        OOOO   OOO    OOOOOOOOOOOOOO
```

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
```

```
       .XXXXXXLm.       .mm.          .mm.       .mJXXXXXX.
      .XXXXXXXXXL     .XXX^XLmm  mmJX^XXX.     JXXXXXXXXX.
      JXXXXXXXXXL.  .XXXXXXXXX XXXXXXXXX.  .JXXXXXXXXXL
    .JXXXXXXXXXXXXL. {XXXXX^^^' `^^^XXXXX} .JXXXXXXXXXXXXL.
```



The use of a camel image in association with Perl is a trademark of O'Reilly & Associates, Inc. Used with permission.

Bibliography

Articles

Books