

Cryptography: Ciphers and One-time Pads

K. Levasseur
Math. Sciences
UMass Lowell

Overview

The basic premise behind cryptography is that two people want to share some information, but do not want anyone else to get it. Here are some of the classic scenarios that come to mind.

Julius Caesar sending a message to his generals and worrying about whether his enemies capture the messenger and learn of his plans.

In World War II, German headquarters sending instructions by radio to U-boat commanders. Can the Allies listen and anticipate the U-boat movements?

You send your credit card information to an e-retailer so that you can buy that cool shirt. Will a hacker be able to intercept your card number?

The "solution" to all of these is to use encryption. The general idea in all cases is the same. We start by assuming that a message is a sequence of letters, digits, spaces and other punctuation that are, these days, associated with the ASCII code. The *Mathematica* functions **ToCharacterCode** and **FromCharacterCode** will be used throughout these notes. For example, the lower case A is encoded to the number 97, while the number 77 is the code for capital M.

```
In[3]:= ToCharacterCode["a"]
```

```
Out[3]= {97}
```

```
In[4]:= FromCharacterCode[77]
```

```
Out[4]= M
```

■ Alphanumeric Character Codes

Most text messages would use characters with codes between 32 (blank) and 127. Here is an array of these numbers matched with the corresponding character.

```
In[5]:= {#, FromCharacterCode[#]} & /@ Range[32, 127] // Partition[#, 8] &
```

```
Out[5]= {{32, " "}, {33, "!"}, {34, "\""}, {35, "#"}, {36, "$"}, {37, "%"}, {38, "&"}, {39, "'"}, {40, "("}, {41, ")"}, {42, "*"}, {43, "+"}, {44, ","}, {45, "-"}, {46, "."}, {47, "/"}, {48, "0"}, {49, "1"}, {50, "2"}, {51, "3"}, {52, "4"}, {53, "5"}, {54, "6"}, {55, "7"}, {56, "8"}, {57, "9"}, {58, ":"}, {59, ";"}, {60, "<"}, {61, "="}, {62, ">"}, {63, "?"}, {64, "@"}, {65, "A"}, {66, "B"}, {67, "C"}, {68, "D"}, {69, "E"}, {70, "F"}, {71, "G"}, {72, "H"}, {73, "I"}, {74, "J"}, {75, "K"}, {76, "L"}, {77, "M"}, {78, "N"}, {79, "O"}, {80, "P"}, {81, "Q"}, {82, "R"}, {83, "S"}, {84, "T"}, {85, "U"}, {86, "V"}, {87, "W"}, {88, "X"}, {89, "Y"}, {90, "Z"}, {91, "["}, {92, "\""}, {93, "]"}, {94, "^"}, {95, "_"}, {96, "`"}, {97, "a"}, {98, "b"}, {99, "c"}, {100, "d"}, {101, "e"}, {102, "f"}, {103, "g"}, {104, "h"}, {105, "i"}, {106, "j"}, {107, "k"}, {108, "l"}, {109, "m"}, {110, "n"}, {111, "o"}, {112, "p"}, {113, "q"}, {114, "r"}, {115, "s"}, {116, "t"}, {117, "u"}, {118, "v"}, {119, "w"}, {120, "x"}, {121, "y"}, {122, "z"}, {123, "{"}, {124, "|"}, {125, "}"}, {126, "~"}, {127, "\u00A0"}}
```

No matter what are message is, we simply think of it as a sequence of numbers. Initially we'll think of a message as a sequence of numbers between 0 and 127 later we'll use a larger range of values.

The encryption/decryption process makes use of two functions, an encryption function (E) and a decryption function

(D). There are two assumptions that we make about these functions. The first is that if a is a number that represents part of a message, then $D(E(a)) = a$. In other words, D inverts E . The other is that the sender knows the encryption function while the receiver knows the decrypting function. As we'll see later, what else is known can depend on the specific system being used. Here are the basic steps in the process.

1. The message number, a , is encrypted by the sender to $b = E(a)$
2. The sender transmits the encrypted message, b , to the receiver. This is the point in the process when it is assumed that a third party could read b . The key issue is what else this third party knows.
3. The receiver decrypts b using the decrypting function $D(b) = D(E(a)) = a$ and has the message.

■ A message

We'll illustrate this process with Lincoln's Gettysburg Address.

```
In[6]:= messageText =
"Four score and seven years ago our fathers brought forth on this continent, a new
nation, conceived in Liberty, and dedicated to the proposition that all men
are created equal. Now we are engaged in a great civil war, testing whether
that nation, or any nation so conceived and so dedicated, can long endure.
We are met on a great battle-field of that war. We have come to dedicate a
portion of that field, as a final resting place for those who here gave their
lives that that nation might live. It is altogether fitting and proper that
we should do this. But in a larger sense, we can not dedicate-- we can not
consecrate-- we can not hallow-- this ground. The brave men, living and dead,
who struggled here, have consecrated it, far above our poor power to add or
detract. The world will little note, nor long remember what we say here, but
it can never forget what they did here. It is for us the living, rather, to be
dedicated here to the unfinished work which they who fought here have thus far
so nobly advanced. It is rather for us to be here dedicated to the great task
remaining before us-- that from these honored dead we take increased devotion
to that cause for which they gave the last full measure of devotion-- that we
here highly resolve that these dead shall not have died in vain-- that this
nation, under God, shall have a new birth of freedom-- and that government of
the people, by the people, for the people, shall not perish from the earth.";
```

We turn this string of characters into a list of numbers. Most of the time, we'll just show you a few of the numbers in this process using the function **Short**.

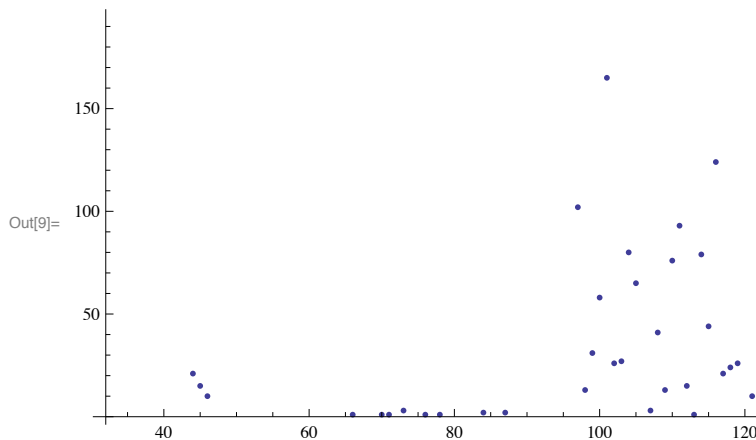
```
In[7]:= messageNumbers = ToCharacterCode[messageText];
Short[messageNumbers]
```

```
Out[8]//Short= {70, 111, 117, 114, 32, 115, 99, 111, 114, 101, 32, 97, 110, 100,
32, <<1445>>, 114, 111, 109, 32, 116, 104, 101, 32, 101, 97, 114, 116, 104, 46}
```

Note : The number inside the `<<>>` 'S' indicates how many entries are skipped in displaying the list.

Notice that certain characters appear much more frequently than others in this message.

```
In[9]:= ListPlot[Tally[messageNumbers]]
```



Ciphers

■ Caesar Ciphers

One way to encode messages is to replace each character in the message with another character. Such an encoding rule is called a cipher. One of the oldest known ciphers, based a shift transformation is attributed to Julius Caesar, and is now called the *Caesar cipher*. The way this cipher works is that each letter in the alphabet is replaced by another letter using a rule which shifts the alphabet a uniform amount to the right. For example, let's consider a shift of six units to the right. This is accomplished with modular arithmetic. The function `ShiftRight` just adds **k** to each number and returns that sum mod 128 so that each encrypted number is in our ASCII range.

```
In[10]:= ShiftRight[a_, k_] := Mod[a + k, 128]
```

In *Mathematica*, we can apply the **ShiftRight** function with **k** equal to 6 by Mapping the pure function **ShiftRight**[#,6]& over the list of message numbers.

```
In[11]:= CaesarEncrypted = Map[ShiftRight[#, 6] &, messageNumbers];  
Short[CaesarEncrypted]
```

Out[12]//Short= {76, 117, 123, 120, 38, 121, 105, 117, 120, 107, 38, 103, 116, 106,
 <<1446>>, 120, 117, 115, 38, 122, 110, 107, 38, 107, 103, 120, 122, 110, 52}

Here is what the encrypted message would look like if it is converted to ASCII.

```
In[13]:= FromCharacterCode[CaesarEncrypted]
```

Out[13]= Lu{x&yiux>j&y&k|kt&∇.kgx&gmu&u{x&lgnkxy&hxu{mnz&luxzn&ut&znoy&iutzotkt2&g&tk}&tgzout2&iutikol&j&ot&Rohkxz∇>j&|kjoigz&j&zu&znk&vxuvuyozout&zngz&grr&skt&g&xk&ix&g&j&k&w{gr4&=Tu}&}&k&g&xk&ktmgmkj&ot&g&mxkgz&io|or&}&jx2&zk&y&zotm&}&nkznkx&zngz&tgzout2&ux>∇.&tgzout&yu&iutikol&j>j&yu&jkjoigz&j2&igt&rutm&ktj{xk4&=}&j&k&g&xk&skz&ut&g&mxkgz&hgzzrk3&lokrj&ul&zngz&}&jx4&=}&j&k&ngl&k&iusk&zu&jkjoigz&k&g&vuxzout&ul&zngz&lokrj2&gy&g&lotgr&xkyzotm&vrgik&lux&znuyk&}&jnu&nkxk&mgl&k&znkox&rol&y&zngz&zngz&tgzout&somnz&rol&k4&=Oz&oy&grzum&kznk&lozzotm>j&vxuv&x&zngz&}&k&ynu{rj&ju&znoy4&=H{z&ot&g&rgxm&kx&yktyk2&}&k&igt&tuz&jkjoigz&k33&}&k&igt&tuz&iuty&kix&gzk33&}&k&igt&tuz&ngrru{33&znoy&mxu{tj4&=Znk&hx&g&k&skt2&rol&otm>j&jkgj2&}&nu&yzx{mmrkj&nkxk2&ngl&k&iuty&kix&g&j&k&oz2&lgx&ghul&k&u{x&vuux&vu}&kx&zu&gij&ux&jkxz&giz4&=Znk&}&juxrj&}&orr&rozzrk&tuz&k2&tux&rutm&xksksh&x&}&ngz&}&k&y&g∇.&nkxk2&h{z&oz&igt&tkl&x&luxmkz&}&ngz&znk∇.&joj&nkxk4&=Oz&oy&lux&}&y&znk&rol&otm2&xgz&znkx2&zu&hk&}&jkjoigz&j&nkxk&zu&znk&}&{tlotoy&nk&}&juxq&}&noin&znk∇.&}&nu&lu{mnz&nkxk&ngl&k&zn{y&lgx&yu&tuhr∇.&gjl>j&k4&=Oz&oy&xgz&znk&lux&}&y&zu&hk&nkxk&jkjoigz&j&zu&znk&mxkgz&zgyq&xksgototm&hkl&lux&}&y33&zngz&lx&g&znkyk&nutux&j&}&jkgj&}&k&gzq&otix&kg&j&k}&jkluzot&zu&z&zngz&ig{y&k&lux&}&noin&znk∇.&mgl&k&znk&rgyz&}&lrr&sk&gy{x&k&ul&jkluzot33&zngz&}&j&nkxk&noinr∇.&x&kyurj&k&zngz&znkyk&}&jkgj&yngrr&tuz&ngl&k&jok&j&ot&}&got33&zngz&znoy&tgzout2&}&tj&kx&Muj2&yngrr&ngl&k&g&tk}&}&hoxzn&ul&l&kk&jus33>j&j&zngz&mul&kxtsktz&ul&znk&vkuvrk2&h∇.&znk&vkuvrk2&lux&znk&vkuvrk2&yngrr&tuz&v&xoyn&l&xus&znk&g&gxzn4

If the encryption function E is a shift right by k , then the decryption function D will be a shift left by k (or a shift right by $-k$). Here we just verify that the decryption function does get us back to the original ASCII code numbers.

```
In[14]:= MessageBack = Map[ShiftRight[#, -6] &, CaesarEncrypted];
MessageBack == messageNumbers
```

```
Out[15]= True
```

■ Affine Ciphers

A somewhat more general family of ciphers are the *affine ciphers*. The encryption function (and the decryption function) of an affine cipher is an invertible linear function to some modulus: $E(a) = ja + k \pmod{n}$ for some fixed integers j , k , and n . In our case, n will be 128. In order for this function to be invertible, we also need n and j to be relatively prime.

If we set $j = 51$ and $k = 30$,

```
In[16]:= AffineEncrypted = Map[Mod[51 # + 30, 128] &, messageNumbers];
Short[AffineEncrypted]
```

```
Out[17]//Short= {16, 59, 109, 84, 126, 7, 87, 59, 84, 61, 126, 113, 8, 10, 126, 7, 61,
<<1440>>, 86, 126, 112, 84, 59, 85, 126, 58, 86, 61, 126, 61, 113, 84, 58, 86, 72}
```

Again, the encrypted message looks like gibberish if it is converted to ASCII. In fact it looks even worse because we pick up some control characters in the 0-127 ASCII range. Here are the first 50 characters.

```
In[18]:= FromCharacterCode[Take[AffineEncrypted, {1, 50}]]
```

```
Out[18]= !;mT~!W;T=~q
~!= =~9=qT!~q#;~;mT~pq;V=T!~$T;m#V:
```

The decryption function associated with the affine cipher is the inverse of the encryption function. You can get the formula for the decryption function by solving $b = 51a + 30$ in the ring $\mathbb{Z}_{128}$.

■ Permutation Ciphers

An even more general cipher is the *permutation cipher* (or *transposition cipher*) that is based on a random permutation of the ASCII range. Here is an example of one such permutation that is randomly selected.

```
In[19]:= p = RandomPermutation[128] - 1
```

```
Out[19]= {102, 105, 34, 19, 12, 64, 97, 124, 56, 41, 20, 108, 55, 51, 53, 118, 96, 70, 81, 62, 3, 76, 23, 90,
39, 52, 44, 127, 103, 17, 111, 119, 82, 73, 48, 114, 61, 104, 6, 74, 11, 84, 10, 40, 36, 5, 22, 30, 89, 49,
38, 31, 50, 91, 87, 99, 60, 54, 4, 110, 67, 35, 29, 42, 66, 98, 25, 100, 123, 63, 45, 113, 117, 120, 1, 95,
86, 28, 26, 37, 116, 69, 21, 72, 14, 65, 94, 126, 47, 33, 46, 13, 107, 59, 16, 78, 112, 32, 85, 27, 58, 7,
115, 101, 77, 68, 93, 79, 109, 43, 88, 9, 121, 80, 122, 24, 75, 106, 83, 8, 125, 0, 71, 18, 92, 57, 15, 2}
```

Here are the replacement rules it defines

```
In[20]:= RepRules = Map[(# -> p[[# + 1]]) &, Range[0, 127]]
```

```
Out[20]= {0 -> 102, 1 -> 105, 2 -> 34, 3 -> 19, 4 -> 12, 5 -> 64, 6 -> 97, 7 -> 124, 8 -> 56, 9 -> 41, 10 -> 20, 11 -> 108, 12 -> 55,
13 -> 51, 14 -> 53, 15 -> 118, 16 -> 96, 17 -> 70, 18 -> 81, 19 -> 62, 20 -> 3, 21 -> 76, 22 -> 23, 23 -> 90, 24 -> 39,
25 -> 52, 26 -> 44, 27 -> 127, 28 -> 103, 29 -> 17, 30 -> 111, 31 -> 119, 32 -> 82, 33 -> 73, 34 -> 48, 35 -> 114,
36 -> 61, 37 -> 104, 38 -> 6, 39 -> 74, 40 -> 11, 41 -> 84, 42 -> 10, 43 -> 40, 44 -> 36, 45 -> 5, 46 -> 22, 47 -> 30,
48 -> 89, 49 -> 49, 50 -> 38, 51 -> 31, 52 -> 50, 53 -> 91, 54 -> 87, 55 -> 99, 56 -> 60, 57 -> 54, 58 -> 4, 59 -> 110,
60 -> 67, 61 -> 35, 62 -> 29, 63 -> 42, 64 -> 66, 65 -> 98, 66 -> 25, 67 -> 100, 68 -> 123, 69 -> 63, 70 -> 45, 71 -> 113,
72 -> 117, 73 -> 120, 74 -> 1, 75 -> 95, 76 -> 86, 77 -> 28, 78 -> 26, 79 -> 37, 80 -> 116, 81 -> 69, 82 -> 21, 83 -> 72,
84 -> 14, 85 -> 65, 86 -> 94, 87 -> 126, 88 -> 47, 89 -> 33, 90 -> 46, 91 -> 13, 92 -> 107, 93 -> 59, 94 -> 16, 95 -> 78,
96 -> 112, 97 -> 32, 98 -> 85, 99 -> 27, 100 -> 58, 101 -> 7, 102 -> 115, 103 -> 101, 104 -> 77, 105 -> 68, 106 -> 93,
107 -> 79, 108 -> 109, 109 -> 43, 110 -> 88, 111 -> 9, 112 -> 121, 113 -> 80, 114 -> 122, 115 -> 24, 116 -> 75,
117 -> 106, 118 -> 83, 119 -> 8, 120 -> 125, 121 -> 0, 122 -> 71, 123 -> 18, 124 -> 92, 125 -> 57, 126 -> 15, 127 -> 2}
```

```
In[21]:= PermutationEncrypted = messageNumbers /. RepRules;
Short[PermutationEncrypted]
```

```
Out[22]//Short= {45, 9, 106, 122, 82, 24, 27, 9, 122, 7, 82, 32, 88, 58, 82, 24, 7, 83,
<<1438>>, 24, 77, 82, 115, 122, 9, 43, 82, 75, 77, 7, 82, 7, 32, 122, 75, 77, 22}
```

You can just reverse all of the rules to get the inverse permutation:

```
In[23]:= InverseRules = RepRules /. { (a_ -> b_) -> (b -> a) }

Out[23]= {102 -> 0, 105 -> 1, 34 -> 2, 19 -> 3, 12 -> 4, 64 -> 5, 97 -> 6, 124 -> 7, 56 -> 8, 41 -> 9, 20 -> 10, 108 -> 11, 55 -> 12,
51 -> 13, 53 -> 14, 118 -> 15, 96 -> 16, 70 -> 17, 81 -> 18, 62 -> 19, 3 -> 20, 76 -> 21, 23 -> 22, 90 -> 23, 39 -> 24,
52 -> 25, 44 -> 26, 127 -> 27, 103 -> 28, 17 -> 29, 111 -> 30, 119 -> 31, 82 -> 32, 73 -> 33, 48 -> 34, 114 -> 35,
61 -> 36, 104 -> 37, 6 -> 38, 74 -> 39, 11 -> 40, 84 -> 41, 10 -> 42, 40 -> 43, 36 -> 44, 5 -> 45, 22 -> 46, 30 -> 47,
89 -> 48, 49 -> 49, 38 -> 50, 31 -> 51, 50 -> 52, 91 -> 53, 87 -> 54, 99 -> 55, 60 -> 56, 54 -> 57, 4 -> 58, 110 -> 59,
67 -> 60, 35 -> 61, 29 -> 62, 42 -> 63, 66 -> 64, 98 -> 65, 25 -> 66, 100 -> 67, 123 -> 68, 63 -> 69, 45 -> 70, 113 -> 71,
117 -> 72, 120 -> 73, 1 -> 74, 95 -> 75, 86 -> 76, 28 -> 77, 26 -> 78, 37 -> 79, 116 -> 80, 69 -> 81, 21 -> 82, 72 -> 83,
14 -> 84, 65 -> 85, 94 -> 86, 126 -> 87, 47 -> 88, 33 -> 89, 46 -> 90, 13 -> 91, 107 -> 92, 59 -> 93, 16 -> 94, 78 -> 95,
112 -> 96, 32 -> 97, 85 -> 98, 27 -> 99, 58 -> 100, 7 -> 101, 115 -> 102, 101 -> 103, 77 -> 104, 68 -> 105, 93 -> 106,
79 -> 107, 109 -> 108, 43 -> 109, 88 -> 110, 9 -> 111, 121 -> 112, 80 -> 113, 122 -> 114, 24 -> 115, 75 -> 116,
106 -> 117, 83 -> 118, 8 -> 119, 125 -> 120, 0 -> 121, 71 -> 122, 18 -> 123, 92 -> 124, 57 -> 125, 15 -> 126, 2 -> 127}

In[24]:= original = PermutationEncrypted /. InverseRules;
Short[original]

Out[25]//Short= {70, 111, 117, 114, 32, 115, 99, 111, 114, 101, 32, 97, 110, 100,
32, <<1445>>, 114, 111, 109, 32, 116, 104, 101, 32, 101, 97, 114, 116, 104, 46}
```

■ Limitations of Ciphers

The problem with any simple cipher is that it doesn't take long to figure out how to decode it. The simplest technique is to do a frequency count of the various characters in a message. The distribution of letters that are used in any language is fairly consistent. For example almost any typical text in English will have lots of letters like e, t, o, n or s and relatively few j's, q's and z's, so some educated guessing can produce the decoding rules pretty quickly. Of course this is not possible for very short messages, but a ciphered message that is roughly the length of the Gettysburg address isn't hard to figure out.

One Time Pads

A *one time pad* overcomes the weakness of ciphers by having a separate replacement rule associated with each character in the message. A one time pad consists of a sequence of random integers between 0 and 127. You need as many integers as there are characters in the message.

Computers can quickly generate these integers, but there is a danger in doing this with extremely long messages because most number generators are not truly random. In World War II, the Allied forces are said to have generated random numbers by selecting balls from a "bingo cage."

```
In[26]:= pad = RandomInteger[{0, 127}, Length[messageNumbers]]; Short[pad]

Out[26]//Short= {19, 27, 66, 44, 91, 19, 4, 89, 51, 18, 29, 107, 54, 8, 59, 65, 77,
32, <<1439>>, 121, 35, 22, 96, 122, 6, 100, 5, 117, 40, 62, 51, 106, 27, 19, 4, 66}
```

This "pad" must be shared between sender and receiver. The sender takes the message numbers and the pad, and lines them up so that the first numbers in each list are added together (mod 128), the two second numbers are added, etc. This produces the encrypted list.

```
In[27]:= paddedMessage = Mod[messageNumbers + pad, 128];
Short[messageNumbers]
Short[pad]
Short[paddedMessage]

Out[28]//Short= {70, 111, 117, 114, 32, 115, 99, 111, 114, 101, 32, 97, 110, 100,
32, <<1445>>, 114, 111, 109, 32, 116, 104, 101, 32, 101, 97, 114, 116, 104, 46}

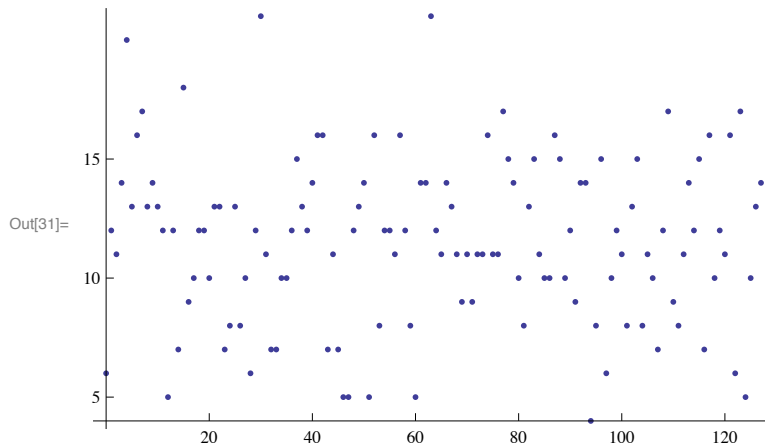
Out[29]//Short= {19, 27, 66, 44, 91, 19, 4, 89, 51, 18, 29, 107, 54, 8, 59, 65, 77,
32, <<1439>>, 121, 35, 22, 96, 122, 6, 100, 5, 117, 40, 62, 51, 106, 27, 19, 4, 66}

Out[30]//Short= {89, 10, 55, 30, 123, 6, 103, 72, 37, 119, 61, 76, 36, 108, 91, 52, 50,
<<1440>>, 97, 67, 124, 82, 105, 115, 4, 121, 93, 13, 94, 24, 75, 13, 7, 108, 112}
```

It's virtually impossible to crack this code if you don't have the pad. Of course, the only problem is that sharing the pad is not always feasible. For example, if you want to send a message to your friend in Thailand, how do you send the pad? Do you encrypt it?

If a pad *can* be shared ahead of time, the kind of statistical guessing that can be done on a cipher is impossible. Here is a frequency plot of the padded message above. You can see that there are no distinctive choices for the common letters.

```
In[31]:= paddata = Map[{#, Count[paddedMessage, #]} &, Range[0, 127]] //
ListPlot[#, Ticks -> {Range[20, 130, 20], {5, 10, 15}}] &
```



The decrypting process is very simple - you just subtract the pad (mod 128).

```
In[32]:= messageBack = Mod[paddedMessage - pad, 128];
Short[paddedMessage]
Short[pad]
Short[messageBack]
```

```
Out[33]//Short= {89, 10, 55, 30, 123, 6, 103, 72, 37, 119, 61, 76, 36, 108, 91, 52, 50,
<<1440>>, 97, 67, 124, 82, 105, 115, 4, 121, 93, 13, 94, 24, 75, 13, 7, 108, 112}
```

```
Out[34]//Short= {19, 27, 66, 44, 91, 19, 4, 89, 51, 18, 29, 107, 54, 8, 59, 65, 77,
32, <<1439>>, 121, 35, 22, 96, 122, 6, 100, 5, 117, 40, 62, 51, 106, 27, 19, 4, 66}
```

```
Out[35]//Short= {70, 111, 117, 114, 32, 115, 99, 111, 114, 101, 32, 97, 110, 100,
32, <<1445>>, 114, 111, 109, 32, 116, 104, 101, 32, 101, 97, 114, 116, 104, 46}
```

```
In[36]:= FromCharacterCode[messageBack]
```

```
Out[36]= Four score and seven years ago our fathers brought forth on this continent, a new nation, conceived in Liberty, and dedicated to the
proposition that all men are created equal. Now we are engaged in a great civil war, testing whether that nation, or any
nation so conceived and so dedicated, can long endure. We are met on a great battle-field of that war. We have come to
dedicate a portion of that field, as a final resting place for those who here gave their lives that that nation might live. It is
altogether fitting and proper that we should do this. But in a larger sense, we can not dedicate-- we can not consecrate--
we can not hallow-- this ground. The brave men, living and dead, who struggled here, have consecrated it, far above
our poor power to add or detract. The world will little note, nor long remember what we say here, but it can never forget
what they did here. It is for us the living, rather, to be dedicated here to the unfinished work which they who fought here
have thus far so nobly advanced. It is rather for us to be here dedicated to the great task remaining before us-- that from
these honored dead we take increased devotion to that cause for which they gave the last full measure of devotion--
that we here highly resolve that these dead shall not have died in vain-- that this nation, under God, shall have a new
birth of freedom-- and that government of the people, by the people, for the people, shall not perish from the earth.
```

■ Enigma and related machines

Generating one time pads mechanically from two different locations was done during WW2, but not securely, which lead to Allied victory.

```
In[37]:= Import["http://static.bbc.co.uk/history/img/ic/640/images/resources/topics/enigma.jpg"]
```

Out[37]=



Mathematica Initializations

```
In[38]:= Needs["Combinatorica`"]  
Off[General::"spell1"]
```