



The University of New Mexico
CS 241—Data Organization: Project 2

Private Key Cipher

Joel Castellanos

Due: Tuesday, Feb 10, 2009 at midnight in WebCT

Problem Specification

Write a C program that uses a given key to encrypt or decrypt a given string using the Castellanos algorithm¹ for a private key cipher.

Syntax:

```
cipher -e|-d key string
```

Arguments:

-e	Encrypt the given string.
-d	Decrypt the given string
key	12 octal digit key
string	Null terminated, char array to be either encrypted or decrypted. If string is to be encrypted, then each character in the array must have an ASCII code in the range [32, 126] base 10. The input string has a maximum length of 1024 characters.

Algorithm:

Characters in the allowed encryption range of [32, 126] only use the lower 7 bits of each byte. For encryption, each character has each of these 7 bits scrambled according to the pattern specified by the key.

The 12 octal digits of the key specify 6 pairs of mappings. The first digit in each pair specifies which bit of each unencrypted character is mapped. The second digit specifies the destination bit of the mapping. Only 6 pairs are needed to specify the 7 mappings because once the first 6 source bits and 6 destination bits have been chosen, there remains only one unmapped source bit which must be mapped to the only remaining destination.

For an arbitrary key, this mapping takes each character in the range [32, 126] and produces a character in the range [1, 126]. Characters [1, 31] are non-printing characters. Thus, to produce a fully printable encoded string, 192_{10} is added to each character less than 32_{10} . Since 192 is greater than 126, double mappings will not be created. A double mapping is when two different unencrypted characters map to the same encrypted character.

¹ Cryptography is not my field. Thus, what I think is an original algorithm may, in fact, be in wide use or may be known and dismissed as trash. I did at least think of it independently and thus, until I learn otherwise, will name it.

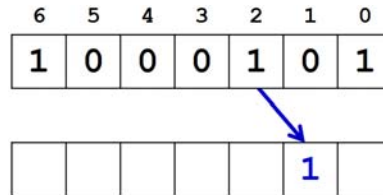
Example:

```
cipher -e 212103553030 "Euler"
```

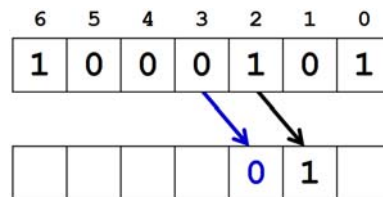
The key, grouped into 6 pairs is: 21 21 03 55 30 30

Each character in the string to be encoded has 7 significant bits. In the case of the 'E', these bits are: 1000101.

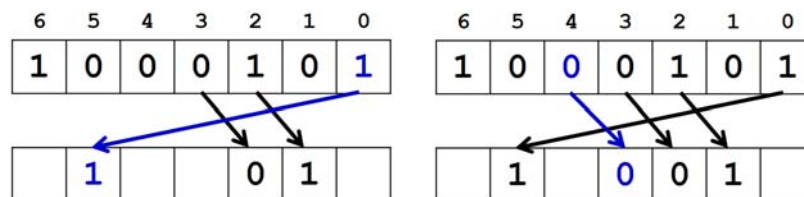
The first key mapping, 21, specifies mapping source bit #2 to destination bit number #1. This is shown below.



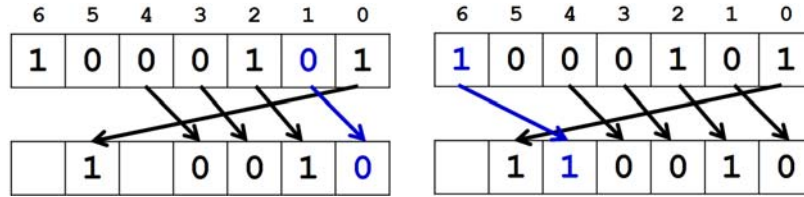
The second key mapping is also 21. To find this mapping, start at bit 0 and count 0, 1, 2, but *only count unmapped bits*. Thus, the 2 in the second mapping refers to source bit 3. Similarly, the 1 in this mapping refers to destination bit 2:



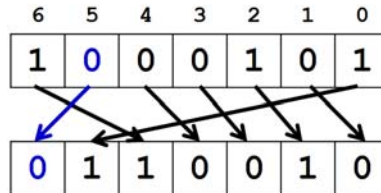
The third key pair is 03 which maps bit 0 to bit 5. Bit 5 is reached by counting: 0, 1, 2, 3 from bit 0 and skipping bits that are already mapped (bits 2 and 3). The fourth key is 55. To count: 0, 1, 2, 3, 4, 5 on unmapped source bits, the counting method is to wrap back to bit 0 after passing bit 6. Thus, the source count 0 starts at the first free bit (which is bit 1). Bits 2, and 3 are already mapped. Thus, the source count 1 lands on bit 4. Source count 2 lands on bit 5. Source count 3 lands on bit 6, and source count 4 wraps back to bit 1 (since bit 0 is already mapped). Finally, source count 5 skips bits 2 and 3 to land on bit 4.



The last two explicit mappings are 30 and 30. The first (using wrap around counting) maps bit 1 to bit 0, and the second 30 maps bit 6 to bit 4.



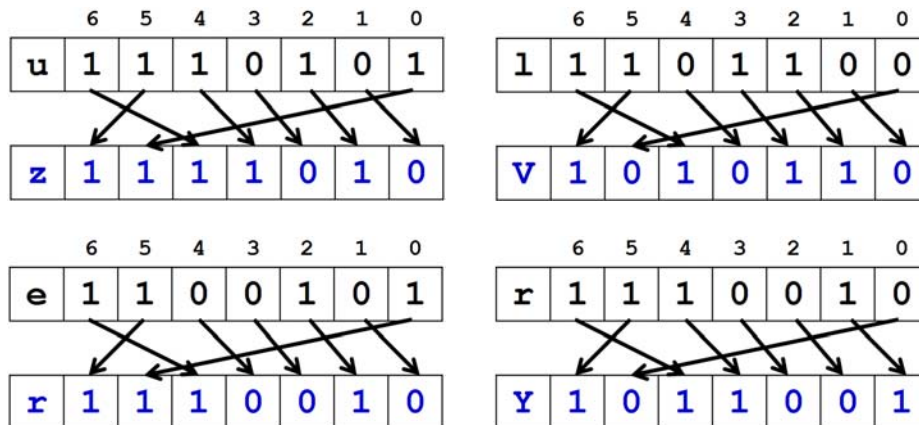
There is now only one possible mapping that remains: 5 to 6.



This same mapping is used on every character. The forward arrows of this mapping (for encryption) are given in the first row of the table below. The reverse arrows (for decryption) are given in the second row. For example, for encryption, bit 0 is moved to bit 5 and bit 1 is moved to bit 0. For decryption, bit 0 is moved to bit 1 and bit 1 is moved to bit 2.

	6	5	4	3	2	1	0
Encryption Map	4	6	3	2	1	0	5
Decryption Map	5	0	6	4	3	2	1

Thus, with the given key, 'E' (1000101) maps to the character '2' (0110010). This same mapping is applied to all characters in the given string:



Hence, cipher -e 212103553030 "Euler" returns the encrypted string: "2zVrY".

Grading:

Your output must be exactly what is shown, as grading will be done by using a script to diff your output with this.

The characters for codes > 126 shown here are the extended ASCII set: ISO-8859-1:1998.

3	<code>cipher -e 212103553030 Euler</code>	<code>2zVrY</code>
3	<code>cipher -d 212103553030 2zVrY</code>	<code>Euler</code>
3	<code>cipher -e 621035533002 Euler</code>	<code>Lntl'</code>
3	<code>cipher -d 621035533002 "Lntl'"</code>	<code>Euler</code>
3	<code>cipher -e 464541032011 Euler</code>	<code>Úz+:f</code>
3	<code>cipher -d 363630030122 47V6Û</code>	<code>Euler</code>
3	<code>cipher -e 552200553344 "Dennis Ritchie"</code>	<code>Dennis Ritchie</code>
3	<code>cipher -d 552200553344 "Dennis Ritchie"</code>	<code>Dennis Ritchie</code>
3	<code>cipher -e 522552255225 Mandelbrot</code>	<code>i,udle46}f</code>
3	<code>cipher -d 522552255225 i,udle46}f</code>	<code>Mandelbrot</code>
3	<code>cipher -e 522552255225 "Allen Newell"</code>	<code>(eeluÄql~lee</code>
3	<code>cipher -e 212103553030 "Alan Turing"</code>	<code>0VpW@ÚzYtWs</code>
3	<code>cipher -d 212103553030 0VpW@ÚzYtWs</code>	<code>Alan Turing</code>
3	<code>cipher -e 621035533002 "Stephanie Forrest"</code>	<code>Ïfl&4,u<l E}''l/f</code>
3	<code>cipher -d 621035533002 "Ïfl&4,u<l E}''l/f"</code>	<code>Stephanie Forrest</code>
3	<code>cipher -e 621035533002 "Kurt Gödel"</code>	<code>Error** Illegal character in encryption string: ö</code>
3	<code>cipher -e 621035533002</code>	<code>Error** cipher expects exactly four arguments.</code>
3	<code>cipher "Kurt Gödel"</code>	<code>Error** cipher expects exactly four arguments.</code>

3	<code>cipher -f 621035533002 "Kurt Gödel"</code>	Error** first argument must be -e or -d.
3	<code>cipher -e 62103553300 "Kurt Gödel"</code>	Error** key must consist of 12 octal digits.
3	<code>cipher -e 621035538002 "Kurt Gödel"</code>	Error** key must consist of 12 octal digits.
3	<code>cipher -e 212103553030 "Another effective debugging technique is to explain your code to someone else. This will often cause you to explain the bug to yourself. Sometimes it takes no more than a few sentences, followed by an embarrassed, Never mind, I see what's wrong. Sorry to bother you. This works remarkably well; you can even use non-programmers as listeners. One university computer center kept a teddy bear near the help desk. Students with mysterious bugs were required to explain them to the bear before they could speak to a human counselor. -- Brian Kernighan and Rob Pike (The Practice of Programming) "</code>	0WwZTrY@rSSrQzT[r@RrQzsstW s@ZrQTWtxzr@ty@Zw@r\XVptW@ wzY@qwRr@Zw@ywvrvWr@rVyrG @ÚTty@{tVV@wSZrW@qpzyr@ wz @Zw@r\XVptW@ZTr@Qzs@Zw@ wz YyrVSG@9wvrZtvry@tZ@Zpury@ Ww@vwYr@ZTpW@p@Sr{@yrWZrWq ryF@SwVVw{rR@Q @pW@rvQpYyp yyrRF@×r[rY@vtWRF@4@yrr@{T pZcy@{YwWsG@9wYY @Zw@QwZTr Y@ wzG@ÚTty@{wYuy@YrvpYupQ V @{rVVm@ wz@qpW@r[rW@zyr@ WwWfXYwsYpvvrYy@py@VtyZrWr YyG@7Wr@zWt[rYytZ @qvwXzZr Y@qrWZrY@urXZ@p@ZrRR @QrpY @WrpY@ZTr@TrVX@RryuG@9ZzRr WZy@{tZT@v yZrYtwzy@Qzsy@{ rYr@YrxztYrR@Zw@r\XVptW@ZT rv@Zw@ZTr@QrpY@QrSwYr@ZTr @qwzVR@yXrpu@Zw@p@TzvpW@qw zWyrVwYG@ff@ÑYtpW@5rYWtsTp W@pWR@ÙwQ@Øtur@DÚTr@ØYpqZt qr@wS@ØYwsYpvvtWsd
4	<code>cipher -d 464541032011 "J? v+::b) b:f6#2/6: j? *f:2;92s) j#:f:<v j#: fz&"</code>	<i><You will know when you get it.></i>
5	Comments are sufficient, accurate, and informative	
10	Algorithm is well organized (no spaghetti code)	
15	Adheres to the UNM CS-241 coding standards	
-10x	Where X is the number of compiler warning messages.	

Extra Credit (up to 20 points):

The algorithm above does a great job of scrambling the bits; however the mapping is the same for every character. For example, if a given key maps an 'e' to an '&', then every 'e' will be mapped to an '&'. This results in a code that is quite easy to crack: If the algorithm is used to encrypted even just a paragraph or two of English, then the most common encrypted symbol is very likely to be from a space. In the test case paragraph on debugging, this is '@'. Knowing the character for a space allows identifying 3 letter words. The most common of these is very likely to be "the". From knowing just these 4 characters (space, t, h and e) it is possible to reconstruct the original 12 digit key (how?).²

The encryption scheme can be made much stronger by using a different map for every character. For extra credit, do this by using the bitwise Exclusive OR operator to generate a new key for each character encoded. The algorithm is as follows:

- 1) Encrypt the first letter of the input string the same as you would with the -e option in the regular part of this project.
- 2) Use the bitwise Exclusive OR operator (^) to combine the first two *destination* octal digits (digits 2 and 4) of the previous key with the least significant 6 bits of the most recently encrypted character. It is important to modify the key by using Exclusive OR because for decryption, the process needs to be reversible. Exclusive OR has the immensely useful property that applying either one of the operands to the result will yield the other operand.
- 3) Using the new key, generate a new map. Changing the first two destination parts of the key will usually affect most of the other mappings. This is due to the map building process of skipping used locations and the wrap-around counting.
- 4) Use the new map to encrypt the next character.
- 5) Return to step 2.

Example: `cipher -E 212103553030 "Euler"`

In this example, the leading 'E' of the input string encrypts as '2', just as it did in the basic assignment. The first two destination octal digits of the original key **1** and **1**. These are XOR'ed with the first encrypted character, '2':

	7	6	5	4	3	2	1	0
Last Key: _1_1*			0	0	1	0	0	1
Last Encrypted Char: 2	0	0	1	1	0	0	1	0
EOR: ^								
Next Key: _7_3*			1	1	1	0	1	1

The resulting key used for generating a map for the second character is then:

272303553030

² The cracking analysis was offered by Yuanpeng Li, a former exchange student at UNM, my Internet Go partner, and assignment critic.

This key encrypts the 'u' in "Euler" as an 'm'. The process is then repeated:

	7	6	5	4	3	2	1	0
Last Key: <u>7</u> <u>3</u> *			1	1	1	0	1	1
Last Encrypted char: m	0	1	1	0	1	1	0	1
EOR: ^	-----							
Next Key: <u>2</u> <u>6</u> *			0	1	0	1	1	0

The first two destination digits of the key become 2 and 6:

222603553030

This key encrypts the 'l' in "Euler" as a 'U'.

Continuing to follow this pattern, `cipher -E 212103553030 "Euler"`

Returns the encrypted string: "2mUiK", with a final key: 212103553030.

For decryption, the reverse path must be followed. Thus, decryption must *start* with the *final key* that had resulted from the encryption, and must process the string in the *reverse order*. For example, an encryption/decryption pair would be:

```
cipher -E 212103553030 "Euler"
```

```
cipher -D 242103553030 "2mUiK"
```

The decryption command was given the key that resulted from XOR of the final character encrypted (the K) with the key that was used to produce the map that transformed the unencrypted r into the encrypted K in the house that Jack built.

The key supplied for decryption, thus, needs to be XORed with the final character in the string (the K), used to decrypt the K and to generate the key needed to be XORed with the second to last character (the i) to decrypt the second to last character.

A few other examples:

```
cipher -E 212103553030 "EEEEEE" → "2)488" 242603553030
```

```
cipher -D 242603553030 "2)488" → "EEEEEE"
```

```
cipher -E 522552255225 "Mandelbrot" →
                                     "iIybiGat}e" 512552255225
```

```
cipher -D 512552255225 "iIybiGat}e" → Mandelbrot
```

```
cipher -E 212103553030 "Alan Turing" →
                                     "0UL[ @í |Uruz" 242203553030
```

The stronger encryption must be an extension of the basic assignment and be evoked by using the options `-E` and `-D` in place of `-e` and `-d`. The `-e` and `-d` options must continue to work as described in the basic assignment.