Simple Cipher in Scala

Readme (readme) | Test Suite (../simple-cipher)

Simple Cipher

Implement a simple shift cipher like Caesar and a more secure substitution cipher

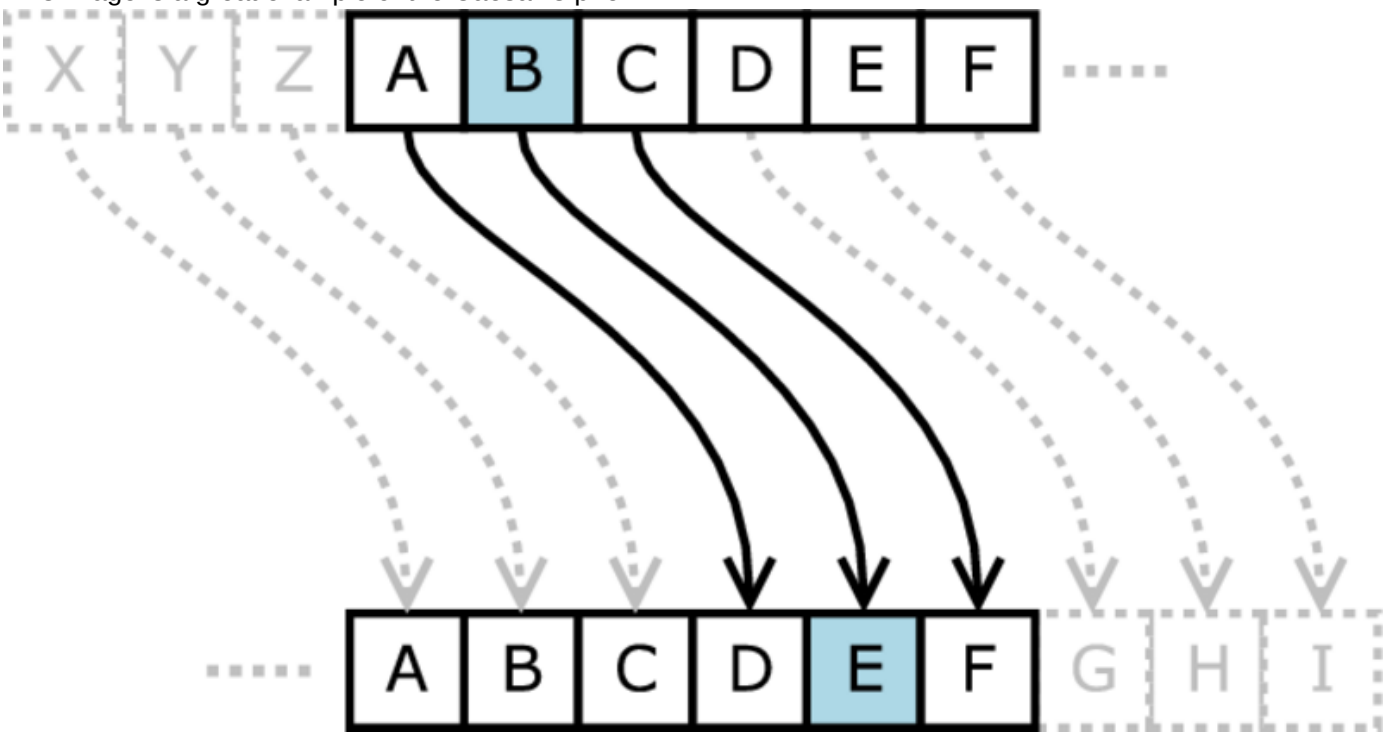
Step 1

"If he had anything confidential to say, he wrote it in cipher, that is, by so changing the order of the letters of the alphabet, that not a word could be made out. If anyone wishes to decipher these, and get at their meaning, he must substitute the fourth letter of the alphabet, namely D, for A, and so with the others."
—Suetonius, Life of Julius Caesar

Ciphers are very straight-forward algorithms that allow us to render text less readable while still allowing easy deciphering. They are vulnerable to many forms of cryptoanalysis, but we are lucky that generally our little sisters are not cryptoanalysts.

The Caeser Cipher was used for some messages from Julius Caesar that were sent afield. Now Caeser knew that the cipher wasn't very good, but he had one ally in that respect: almost nobody could read well. So even being a couple letters off was sufficient so that people couldn't recognize the few words that they did know.

Your task is to create a simple shift cipher like the Caesar Cipher.
This image is a great example of the Caesar Cipher:



Here are some examples:

```
1 @cipher = Cipher.new
2 @cipher.encode("iamapandabear") #=> "ldpdsdqgdehdu"
3 @cipher.decode("ldpdsdqgdehdu") #=> "iamapandabear"
```

Step 2

Shift ciphers are no fun though when your kid sister figures it out. Try amending the code to allow us to specify a key and use that for the shift distance. This is called a substitution cipher.

Here's an example:

```
1 @cipher = Cipher.new("aaaaaaaaaaaaaaaaaa")
2 @cipher.encode("iamapandabear") #=> "iamapandabear"
3 @cipher = Cipher.new("ddddddddddddddddd")
4 @cipher.encode("imapandabear") #=> "lpdsdqgdehdu"
```

In the example above, we've set a = 0 for the key value. So when the plaintext is added to the key, we end up with the same message coming out. So "aaaa" is not an ideal key. But if we set the key to "dddd", we would get the same thing as the Caesar Cipher.

Step 3

The weakest link in any cipher is the human being. Let's make your substitution cipher a little more fault tolerant by providing a source of randomness and ensuring that they key is not composed of numbers or capital letters.

If someone doesn't submit a key at all, generate a truly random key of at least 100 characters in length, accessible via Cipher#key (the # syntax means instance variable)

If the key submitted has capital letters or numbers, throw an ArgumentError with a message to that effect.

Some examples:
@cipher (/cipher) = Cipher.new
@cipher (/cipher).key #=>
"duxrceqyaimciuucnelkeoxjhdyduucpmrxmaivacmybmsdrzwqxvbxsygzsabdjmdjabeorttiwinfrpmpogvabiofqexnohrqu"

Extensions

Shift ciphers work by making the text slightly odd, but are vulnerable to frequency analysis. Substitution ciphers help that, but are still very vulnerable when the key is short or if spaces are preserved. Later on you'll see one solution to this problem in the exercise "crypto-square".

If you want to go farther in this field, the questions begin to be about how we can exchange keys in a secure way. Take a look at Diffie-Hellman on Wikipedia (http://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange) for one of the first implementations of this scheme.

The Scala exercises assume an SBT project scheme. The exercise solution source should be placed within the exercise directory/src/main/scala. The exercise unit tests can be found within the exercise directory/src/test/scala.

To run the tests simply run the command `sbt test` in the exercise directory.

For more detailed info about the Scala track see the help page (http://help.exercism.io/getting-started-with-scala.html).

Source

Substitution Cipher at Wikipedia view source (http://en.wikipedia.org/wiki/Substitution_cipher)

 (https://bugsnag.com/blog/bugsnag-loves-open-source)

 (http://www.rackspace.com/)

 (http://www.shopify.com/)