



Questions

 Bookmark this page

unit testing

3/3 points (graded)

The **RSA cryptography algorithm** needs two prime numbers, p and q , to encrypt and decrypt messages. The encryption key is the product $n=pq$, and the decryption key is p and q .

Suppose you are building an RSA implementation with several modules in it:

```
long randomPrimeNumber();
String encrypt(long n, String message);
String decrypt(long p, long q, String message);
```

Construct a suitable **unit test** for `decrypt()` below by choosing code to replace `code1`, `code2`, and `code3`. Assume the resulting test is correct.

```
String originalMessage = "hello!";
code1
code2
code3
assertEquals(originalMessage, decryptedMessage);
```

code1

☐ `long p = randomPrimeNumber(); long q = randomPrimeNumber();`

☒ `long p = 17; long q = 37;`



code2

☒ `String encryptedMessage = "iy@94RFe"`

☐ `String encryptedMessage = encrypt(p*q, originalMessage);`



code3

☒ `String decryptedMessage = decrypt(p, q, encryptedMessage);`

☐ `String decryptedMessage = "hello!";`




Explanation

A good unit test for `decrypt()` isolates it, calling **only** `decrypt()` rather than the other modules of the program.

How did we come up with the `encryptedMessage` without calling `encrypt()`? For small p and q , we may have done it by hand. Or we may have run `encrypt()` and stored its output once we were confident that it worked, so that this code could detect regressions.

Submit

 Show Answer

 Answers are displayed within the problem