RSA.java

```
7      public static void main(String[] args) {
8
9          RSA rsa = new RSA();
10
11         //
12         long p = 7;
13         long q = 11;
14         BigInteger m = new BigInteger("6");
15         BigInteger n = rsa.getN(p, q);
16         long e = rsa.pubKey(p, q);
17         long d = rsa.priKey(p, q, e);
18         BigInteger c = rsa.encode(m, e, n);
19         m = rsa.decode(c, d, n);
20         System.out.println("e: " + e + ", d: " + d);
21         System.out.println("m: " + m + ", c: " + c);
22     }
23
24     public BigInteger getN(long p, long q) {
25         BigInteger n = new BigInteger(p * q + "");
26         return n;
27     }
28
```

Console | Problems | Coverage | Declaration | Javadoc

<terminated> RSA [Java Application] C:\my_software\java_place\installation\bin\javaw

```
e: 7, d: 43
m: 6, c: 41
```

RSA.java

```
7      public static void main(String[] args) {
8
9          RSA rsa = new RSA();
10
11         //
12         long p = 11;
13         long q = 13;
14         BigInteger m = new BigInteger("9");
15         BigInteger n = rsa.getN(p, q);
16         long e = rsa.pubKey(p, q);
17         long d = rsa.priKey(p, q, e);
18         BigInteger c = rsa.encode(m, e, n);
19         m = rsa.decode(c, d, n);
20         System.out.println("e: " + e + ", d: " + d);
21         System.out.println("m: " + m + ", c: " + c);
22     }
23
24     public BigInteger getN(long p, long q) {
25         BigInteger n = new BigInteger(p * q + "");
26         return n;
27     }
28
```

Console | Problems | Coverage | Declaration | Javadoc

```
<terminated> RSA [Java Application] C:\my_software\java_place\installation\bin\javaw
e: 7, d: 103
m: 9, c: 48
```

RSA.java

```
    public static void main(String[] args) {

        RSA rsa = new RSA();

        //
        long p = 17;
        long q = 31;
        BigInteger m = new BigInteger("5");
        BigInteger n = rsa.getN(p, q);
        long e = rsa.pubKey(p, q);
        long d = rsa.priKey(p, q, e);
        BigInteger c = rsa.encode(m, e, n);
        m = rsa.decode(c, d, n);
        System.out.println("e: " + e + ", d: " + d);
        System.out.println("m: " + m + ", c: " + c);
    }

    public BigInteger getN(long p, long q) {
        BigInteger n = new BigInteger(p * q + "");
        return n;
    }

```

Console | Problems | Coverage | Declaration | Javadoc

```
<terminated> RSA [Java Application] C:\my_software\java_place\installation\bin\javaw
e: 7, d: 343
m: 5, c: 129
```

Let X' be the bitwise complement of X. Prove that if the complement of the plaintext block is taken and the complement of an encryption key is taken, then the result of DES encryption with these values is the complement of the original ciphertext. That is,

if Y = E(K, X), then Y' = E(K', X).

$y = \text{DES}_k(x)$
$y' = \text{DES}_{k'}(x')$

To prove the statement we make the following observations:

- The initial and final permutations are simple rearrangements and therefore do preserve the complements: $IP(x') = (IP(x))'$ and $IP^{-1}(x') = (IP^{-1}(x))'$
- All rounds are identical; proving that one round generates a complemented output for a complemented input proves that all rounds behave similarly.

In round i, the following is computed

$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i).$$

In the computation of the f-function, the expansion of the input clearly preserves complements since its a simple permutation with some additional repetitions:

$E(x') = (E(x))'$.

Similarly, the key-schedule preserves the complement. The XOR of K_i and $E(x')$ yields $K_i' \oplus E(x') = K_i' \oplus E(x)' = K_i \oplus E(x)$. This follows from a basic property of the XOR function $a' \oplus b' = a \oplus b$. Thus, the input to the S-boxes will be identical to the uncomplemented case. Consequently, we end up with the following interesting property:

$$f(R_{i-1}', K_i') = f(R_{i-1}, K_i)$$

The round computation becomes

$$R_i = L_{i-1}' \oplus f(R_{i-1}', K_i')$$
$$R_i = L_{i-1}' \oplus f(R_{i-1}, K_i)$$
$$R_i = L_{i-1}' \oplus 1 \oplus f(R_{i-1}, K_i) = (L_{i-1} \oplus f(R_{i-1}, K_i))'.$$

This shows that the right half is complemented when the input and the key is complemented. The left half is simply the copy of the right half in the previous round. Hence, the entire output is complemented.