Hanna Rakhsha
CS 4371 – Gu
Homework 5

1. Key is: 23. Plain text is: WHENINTHECOURSEOFHUMANEVENTS

i	Φ(i)	i	Φ(i)	i	Φ(i)
0	0.0365	9	0.0367	18	0.0328
1	0.0426	10	0.0470	19	0.0416
2	0.0358	11	0.0417	20	0.0266
3	0.0366	12	0.0375	21	0.0251
4	0.0348	13	0.0441	22	0.0407
5	0.0282	14	0.0365	23	0.0695
6	0.0406	15	0.0312	24	0.0435
7	0.0351	16	0.0373	25	0.0344
8	0.0409	17	0.0416		

2. A Caesar cipher is not a public key system, even though its encryption/decryption keys are different, because the private key is not computationally infeasible to derive from the public key. If k is the encryption key, it's well known that the decryption key is 26 – k.

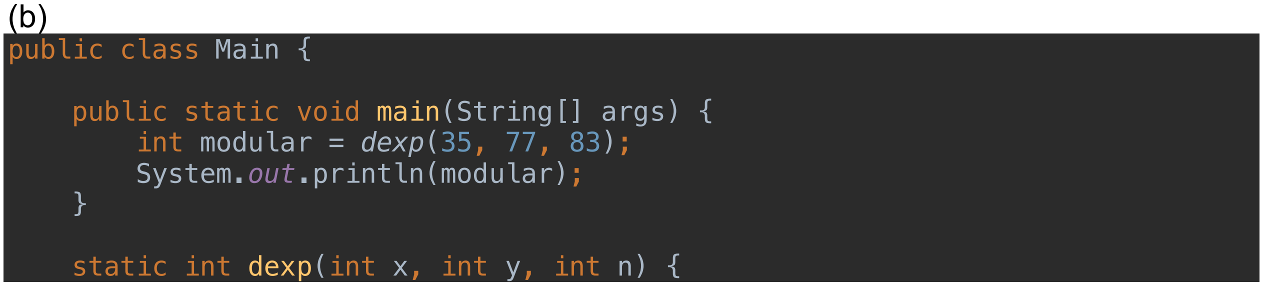
3(a) 77 = 1001101 in binary
$77 = 2^0 + 2^2 + 2^3 + 2^6$
$77 = 1 + 4 + 8 + 64$

$35^{77}\%83 = 35^1 + 35^4 + 35^8 + 35^{64}$

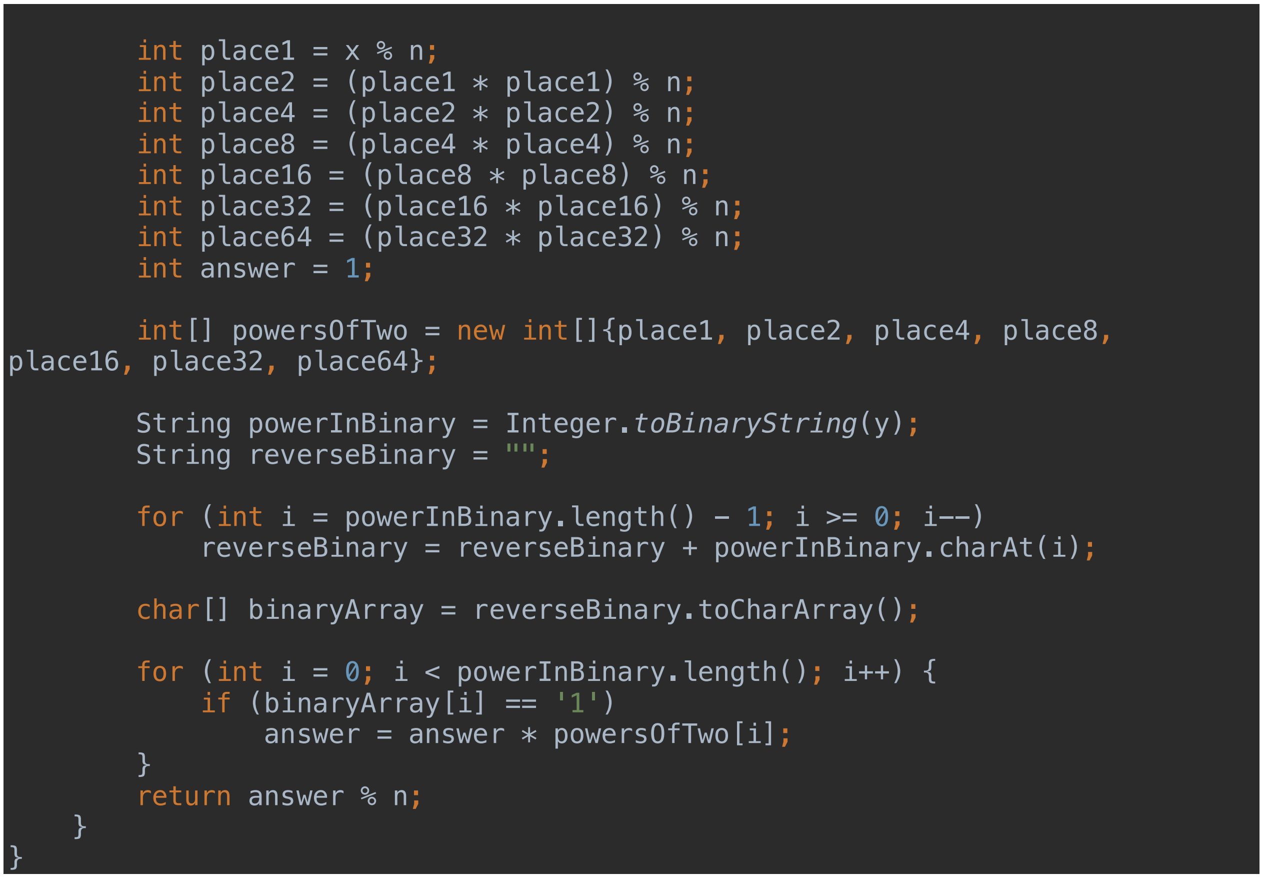
$35^1 \%83 = 35$
$35^2 \%83 = (35 * 35)\%83 = 63$
$35^4 \%83 = (63 * 63)\%83 = 68$
$35^8 \%83 = (68 * 68)\%83 = 59$
$35^{16} \%83 = (59 * 59)\%83 = 78$
$35^{32} \%83 = (78 * 78)\%83 = 25$
$35^{64} \%83 = (25 * 25)\%83 = 44$

$35^{77}\%83 = (35 * 68 * 59 * 44)\%83 = 43$

(b)

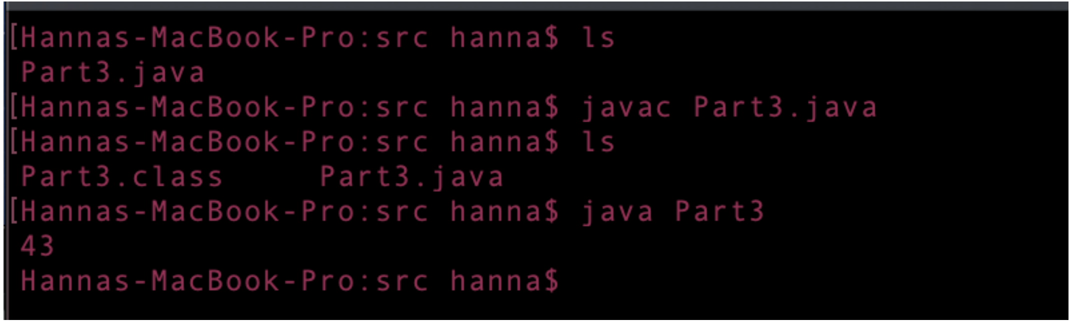


```
public class Main {

    public static void main(String[] args) {
        int modular = dexp(35, 77, 83);
        System.out.println(modular);
    }

    static int dexp(int x, int y, int n) {
```



```
        int place1 = x % n;
        int place2 = (place1 * place1) % n;
        int place4 = (place2 * place2) % n;
        int place8 = (place4 * place4) % n;
        int place16 = (place8 * place8) % n;
        int place32 = (place16 * place16) % n;
        int place64 = (place32 * place32) % n;
        int answer = 1;

        int[] powersOfTwo = new int[]{place1, place2, place4, place8,
place16, place32, place64};

        String powerInBinary = Integer.toBinaryString(y);
        String reverseBinary = "";

        for (int i = powerInBinary.length() - 1; i >= 0; i--)
            reverseBinary = reverseBinary + powerInBinary.charAt(i);

        char[] binaryArray = reverseBinary.toCharArray();

        for (int i = 0; i < powerInBinary.length(); i++) {
            if (binaryArray[i] == '1')
                answer = answer * powersOfTwo[i];
        }
        return answer % n;
    }
}
```

(c)

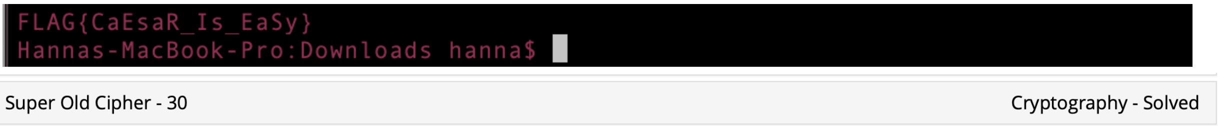


```
[Hannas-MacBook-Pro:src hanna$ ls
 Part3.java
[Hannas-MacBook-Pro:src hanna$ javac Part3.java
[Hannas-MacBook-Pro:src hanna$ ls
 Part3.class      Part3.java
[Hannas-MacBook-Pro:src hanna$ java Part3
 43
 Hannas-MacBook-Pro:src hanna$
```

4. No, this byte-sum program is not a secure hash function.
1010101010101010 XOR 0000000000000000 = 1010101010101010
produces the same one-byte hash as
1010101011111111 XOR 0000000000000000 = 1010101011111111
Both having matching first byte (10101010) showing this is not secure.

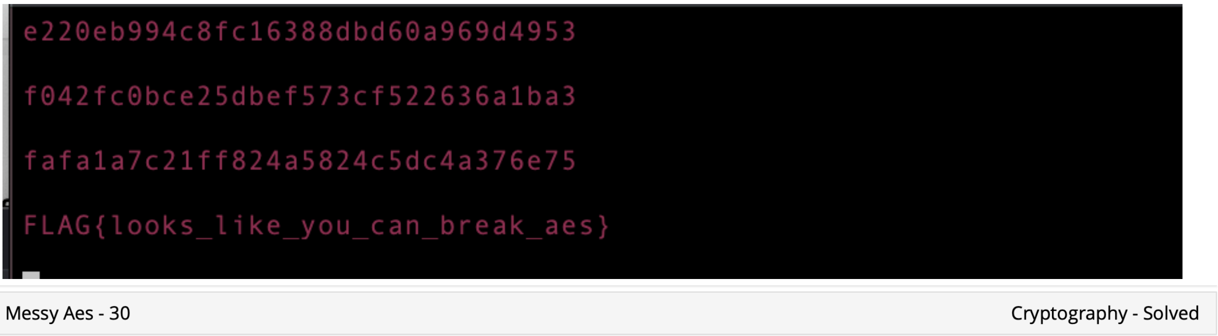
5.
0xb197d3afe713816582ee988b276f635800f728f118f5125de1c7c1e57f2738351de8ac6
43c118a5480f867b6d8756021911818e470952bd0a5262ed86b4fc4c2b7962cd197a8bd
8d8ae3f821ad712a42285db67c85983581c4c39f80dbb21bf700dbd2ae9709f7e307769b
5c0e624b661441c1ddb62ef1fe7684bbe61d8a19e7

6. P35 = 20016431322579245244930631426505729
   P35 = 17963604736595708916714953362445519

7. 

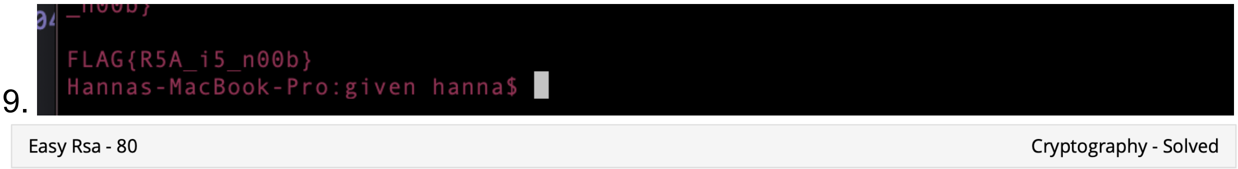
```
FLAG{CaEsaR_Is_EaSy}
Hannas-MacBook-Pro:Downloads hanna$
```

Super Old Cipher - 30 Cryptography - Solved

8. 

```
e220eb994c8fc16388dbd60a969d4953

f042fc0bce25dbef573cf522636a1ba3

fafa1a7c21ff824a5824c5dc4a376e75

FLAG{looks_like_you_can_break_aes}
```

Messy Aes - 30 Cryptography - Solved

9. 

```
_n00b}

FLAG{R5A_i5_n00b}
Hannas-MacBook-Pro:given hanna$
```

Easy Rsa - 80 Cryptography - Solved