Software Security - HW2

הוראות הגשה

- ניתן לכתוב את הקוד באחת משפות תכנות המקובלות. הדוגמאות הן ב-Java.
- אין צורך לתכנן ממשק GUI.
- יש להגיש את כל הפרוייקט.
- הגשה לתיבת הגשה במודל בלבד.
- קבעו לפי נוחיותכם את טיפוסי הפרמטרים שכל פונקציה מקבלת ומחזירה.
- יש לצרף קובץ txt עם דוגמאות הרצה.
- ניתן להגיש בזוגות, תאריך הגשה: 7/12/25.

Use BigInteger and SecureRandom if you write in Java.

Do **not** use BigInteger.modInverse() or BigInteger.modPow()!

Task 1 – RSA

1. Implement the Extended Euclidean Algorithm

The extended Euclidean Algorithm is useful in order to solve an identity such as

$$au + bv = \gcd(a, b)$$

where *a,b,u,v* are integers. There are many methods for solving, the recommended one is the iterative method (https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm#Pseudocode). For example:

$$\gcd(32, 200) = 8 = 32 \cdot -6 + 200 \cdot 1$$

32 and 200 are not co-prime.

$$\gcd(2,17) = 1 = 2 \cdot -8 + 17 \cdot 1$$

2 and 17 are co-prime.

Usage example:

```
BigInteger a = new BigInteger("32"), b = new BigInteger("200");
BigInteger[] g = RSA.exGCD(a, b);
System.out.println(String.format("gcd(%s, %s) = %s = %s * %s + %s * %s", a, b,
g[0], a, g[1], b, g[2]));
```

Output:

```
gcd(32, 200) = 8 = 32 * -6 + 200 * 1
```

```
gcd(2, 17) = 1 = 2 * -8 + 17 * 1
```

Find the multiplicative inverse of a modulo b.
From previous example: 1 is the multiplicative inverse of 17 modulo 2 and -8 (or 9, which is the same element) is the multiplicative inverse of 2 modulo 17.

Usage example:

```
System.out.println(RSA.inverse(a, b));
```

Output:

```
9
```

2. Write a function which generates the public key, $k_{pub} = (n, e)$, and private key, $k_{prv} = (d)$, according to RSA algorithm, for a given length of n in bits.

   Usage example (100 is the length of n in bits):

```
RSA rsa = new RSA(100);
System.out.println(rsa);
```

   Output:

```
public: n=624762232584708828118230949243, e=3
```

```
private: d=4165081550564714765060676256 27
```

3. Write the encryption and decryption functions. Note that you need to implement an exponentiation function yourself, do not use *modPow* or similar built-in.

Usage example:

```
BigInteger c = rsa.encrypt(new
BigInteger("70000000000000000000003"));

System.out.println("encrypted: " + c);
```

Output:

```
encrypted: 516169569473516444683680378301
```

Usage example:

```
BigInteger m = rsa.decrypt(c);

System.out.println("decrypted: " + m);
```

Output:

```
decrypted: 70000000000000000000003
```

Task 2 – Diffie – Hellman

1. Write a function which generates a large prime number and $\alpha = 2$.

   Usage example:

```
DH dh = new DH(100);
System.out.println(dh);
```

   Output:

```
q=764831691931298185996335402667, a=2
```

2. Write a function which generates a private key x and a public key 2^x, for each peer.
3. Write a function which produces shared secret from private key x and public key 2^y

   Usage example:

```
BigInteger[] x = dh.genKey(), y = dh.genKey();
BigInteger s1 = dh.getSecret(x[0], y[1]);
BigInteger s2 = dh.getSecret(y[0], x[1]);
System.out.println(String.format("combining secret x=%s with public a^y=%s: %s",
x[0], y[1], s1));
System.out.println(String.format("combining secret y=%s with public a^x=%s: %s",
y[0], x[1], s2));
```

Output:

```
combining secret x=1256645164254733062576746615 1 with public
a^y=900658169968723275093741269 85: 68645970096599928380626731504 7
```

```
combining secret y=585507590913538628756643211176 with public
a^x=707065304394845855746096227943: 68645970096599928380626731504 7
```