Lada Ivanna Matveeva
AIS ID 99198

Lab 9

Task 1 (ex1.py)

Based on the tables of the initial IP permutation and the final IP permutation, we chose even n and odd m, odd and even m. For task 1.1, we calculated the hash function **h(n)=n mod m**. Using the hash function from task 1.1, we then calculated the hash function for task 1.2.

The results are presented below:

```
N values even  [14, 18, 54, 10, 44]
N values odd  [45, 33, 15, 29, 31]
M values odd  [53, 47, 43, 57, 41]
M values even  [4, 18, 50, 34, 36]
Hash Function Results
    n   m  h(n)  h2(n)               h2(n) inputs          Case
0  14  53    14     52  [54, 10, 44, 14, 18, 18]  Even n, Odd m
1  18  47    18     26  [14, 10, 18, 54, 10, 14]  Even n, Odd m
2  54  43    11     12  [14, 18, 44, 10, 44, 54]  Even n, Odd m
3  10  57    10     23  [14, 18, 54, 54, 44, 10]  Even n, Odd m
4  44  41     3     27  [18, 14, 54, 10, 44, 10]  Even n, Odd m
5  45   4     1      2  [33, 31, 15, 33, 29, 45]  Odd n, Even m
6  33  18    15     10  [45, 15, 15, 33, 33, 31]  Odd n, Even m
7  15  50    15     48  [33, 31, 45, 29, 15, 45]  Odd n, Even m
8  29  34    29      0  [15, 29, 15, 33, 33, 45]  Odd n, Even m
9  31  36    31      4  [45, 31, 31, 15, 29, 33]  Odd n, Even m
```

Task 2 (ex2.py)

I propose an algorithm for the fast transformation of a string of arbitrary length into a hash code.

The algorithm initializes the hash value with a large prime. For each character, the hash value is updated using bitwise operations: (hash_value << 5) + hash_value + ord(char). After processing the string, the hash value is returned as a positive integer by applying a bitwise AND with 0xFFFFFFFF. This ensures that the resulting hash fits within a 32-bit integer range. The generated hash code represents a transformation of the original string.

The results are presented below:

```
Input string: Hello, World!
Hash code: 2531426958
```

Task 3 (ex3.py)
I implemented the Schnorr electronic signature algorithm based on the task.

The algorithm uses a prime number p, a subgroup order q, and a generator g from the Schnorr group. The value x serves as the private key, and the public key y is derived from it.
The get_schnorr_group() function is used to determine suitable generators g that can form a valid Schnorr group.

Signing Process (by Participant A):

- The signer chooses a random number k and computes r = g^k mod p.
- A value e is derived by hashing the concatenation of r and the message M, and then taking the result modulo q.
- The signature (e, s) is created, where s is computed using both the private key x and the random value k.
- The signature (e, s) is sent along with the message to the verifier.

Verification Process (by Participant B):

- The verifier calculates r' based on the received signature and the public key y.
- A value e' is computed by hashing the concatenation of r' and the message M.
- The verifier then checks whether e' matches e from the signature. If they are equal, the signature is valid.

The results are presented below:

```
p: 48731
q: 443
Found 442 possible generators g of a Schnorr group subroup of Z_p^x of order q
Picked generator g: 39649
Private key x: 42566

Participant A
Randomly choose value k:  2936
Public key y: 19958
Calculated r:  40985
Signature: (369, 124)

Message: Hello World

Participant B
Calculated r`:  40985
Calculated e`:  369
Verification result: True
```