Ministerul Educaţiei și Cercetării al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică

Criptography and Security

Laboratory Work 5:

Public Keys Cryptography

Elaborated:
st. gr. FAF-212 Cristian Brinza

Verified:
asist. univ. Mitu Catalin

Chişinău - 2023

Objective

Study and implement the following public keys algorithms: RSA, ElGamal and Diffe-Helman. Implement the algorithms using Wolfram Matematica. Study the material posted on ELSE.

Theoretical Notes

RSA is a widely used public-key encryption algorithm that utilizes a pair of keys (public and private) for secure communication and digital signatures. Its security is based on the computational difficulty of factoring large numbers. ElGamal is another public-key encryption algorithm and digital signature scheme. It relies on the Diffie-Hellman key exchange for secure key generation and is based on the discrete logarithm problem for its security. Diffie-Hellman is a public-key key exchange protocol that enables two parties to establish a shared secret key over an untrusted channel. Its security relies on the difficulty of calculating discrete logarithms in a finite field, and it forms the basis for many cryptographic protocols.

Task 1

Using the wolframalpha.com platform or the Wolfram Mathematica app, generate the keys and perform the encryption and decryption of the message **m = Last Name First Name** the RSA algorithm. The value of n must be at least 2048 bits.

Implementation

In order to implement this algorithm we need to pass through these steps:

1) **Generate Prime Numbers (p and q):** Two large prime numbers, p and q, are generated. These primes are very large (between 2^{1023} and $2^{1024}-1$), ensuring a high level of security.
2) **Compute the Modulus (n):** The modulus n is calculated by multiplying p and q. The value of n is used as part of the public and private keys and its length (in bits) determines the key size, which in our case is at least 2048 bits.
3) **Calculate Euler's Totient Function (ϕ)**: Euler's totient function ϕ(n) is computed as (p−1)(q−1). This value is used in the calculation of the private key.
4) **Choose Public Exponent (e):** A public exponent e is chosen. It is a prime number selected randomly between 2^{16} and ϕ(n). The value of e must be coprime to ϕ(n), ensuring e and ϕ(n) have no common factors other than 1.
5) **Compute Private Exponent (d):** The private exponent d is calculated as the modular multiplicative inverse of $e \bmod \phi(n)$. This means d is the number such that $e \times d$ *modulo* $\phi(n)$ equals 1.
6) **Encryption:** The message "Cristian Brinza" is converted into a numerical format (using character codes). Each character of the message is then encrypted using the formula *encryptedMessage* = $message^{e}$ *mod n.*
7) **Decryption:** The encrypted message is decrypted using the formula *decryptedMessage* = $encryptedMessage^{d}$ *mod n*. The decrypted numerical values are converted back to characters, reconstructing the original message.

Task 2

Using the wolframalpha.com platform or the Wolfram Mathematica application, generate the keys and perform the encryption and decryption of the message **m = Last Name First Name** by applying the ElGamal algorithm (p and the generator are given below).

p=32317006071311007300153513477825163362488057133489075174588434139269806834136210002
7920563626401646854585563579353308169288290230805734726252735547424612457410262025279165
7297286270630032526342821314576693141422365422094111134862999165747826803423055308634905
0635557712219187890332729569696129743856241741236237225197346402691855797767976823014625
3979330580152268587307611975324364674758554607150438968449403661304976978128542959586
5959756705128385213278446852292550456827287911372009893187395914337417583782600027803497
3198552060607533234122603254684088120031105907484281003994966956119696956248629032338072
839127039
g=2

Implementation

In order to implement this algorithm we need to pass through these steps:

1) Define Parameters: A large prime number p and a base g (usually a small integer like 2) are defined. These are public parameters in the ElGamal system.
2) Generate Private and Public Keys: A private key x is randomly chosen such that $1 \leq x \leq p-2$. The public key y is computed as $y = g^x \mod p$.
3) Prepare the Message: The message "Cristian Brinza" is converted from its hexadecimal representation to a decimal format. Each character of the message is represented in hexadecimal and then converted to its decimal equivalent.
4) Encryption Function: The encrypt function takes a message, the prime p, the base g, and the public key y as inputs. For each character in the message, a random integer k is chosen. The first part of the ciphertext, c1, is calculated as $c1 = g^k \mod p$. The second part of the ciphertext, c2, is calculated as $c2 = message \times y^k \mod p$. The function returns a pair (c1,c2) for each character in the message.
5) Encrypt the Message: Each character of the decimal message is encrypted using the encrypt function, resulting in an array of (c1,c2) pairs.
6) Decryption Function: The decrypt function takes a ciphertext pair (c1,c2), the prime p, and the private key x as inputs. The decrypted message is computed as $decryptedMessage = c2 \times c1^{p-1-x} \mod p$.
7) Decrypt the Message: Each (c1,c2) pair in the encrypted message is decrypted using the decrypt function, reconstructing the original message in decimal form.

Task 3

Using the wolframalpha.com platform or the Wolfram Mathematica app, perform the Diffie-Helman key exchange between Alice and Bob, which uses AES algorithm with 256-bit key. The secret numbers a and b must be chosen randomly according to algorithm requirements (p and generator are given below).

p=32317006071311007300153513477825163362488057133489075174588434139269806834136210002
7920563626401646854585563579353308169288290230805734726252735547424612457410262025279165
7297286270630032526342821314576693141422365422094111134862999165747826803423055308634905
0635557712219187890332729569696129743856241741236237225197346402691855797767976823014625
3979330580152268587307611975324364674758554607150438968449403661304976978128542959586
5959756705128385213278446852292550456827287911372009893187395914337417583782600027803497
3198552060607533234122603254684088120031105907484281003994966956119696956248629032338072
839127039
g=2

Implementation

In order to implement this algorithm we need to pass through these steps:

1) Define Global Public Parameters: A large prime number p and a base g (usually a small integer like 2) are defined. These parameters are public and shared between the parties involved in the key exchange.
2) Generate Private Keys: Both Alice and Bob generate their private keys, a and b, respectively. These are randomly chosen integers in the range [1,p−1].
3) Compute Public Keys: Alice computes her public key A as $A = g^a \ mod\ p$. Bob computes his public key B as $B = g^b \ mod\ p$. These public keys are then shared between Alice and Bob.
4) Compute Shared Secret: Alice computes the shared secret using Bob's public key: $sharedSecretAlice = B^a \ mod\ p$. Bob computes the shared secret using Alice's public key: $sharedSecretBob = A^b \ mod\ p$. Due to the properties of modular exponentiation, both computations result in the same value, even though Alice and Bob use their own private keys and each other's public keys.

Results

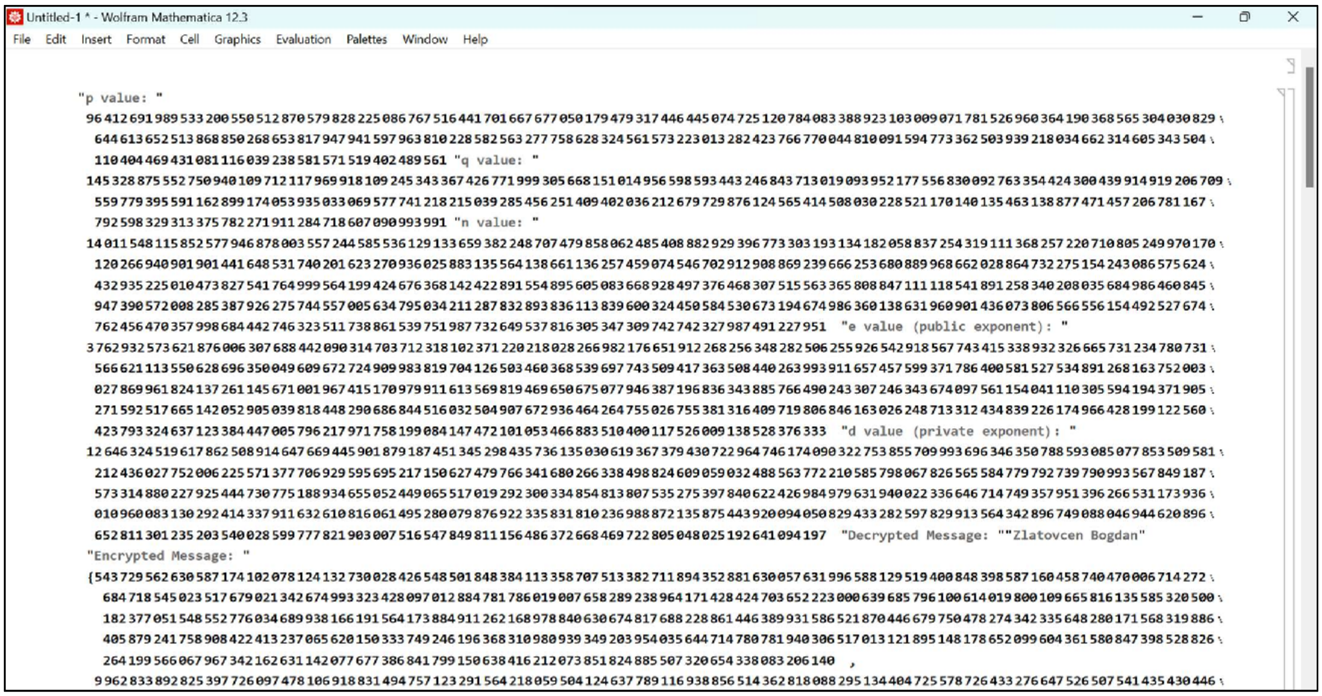

Figure 1 – *Program output for RSA Algorithm*

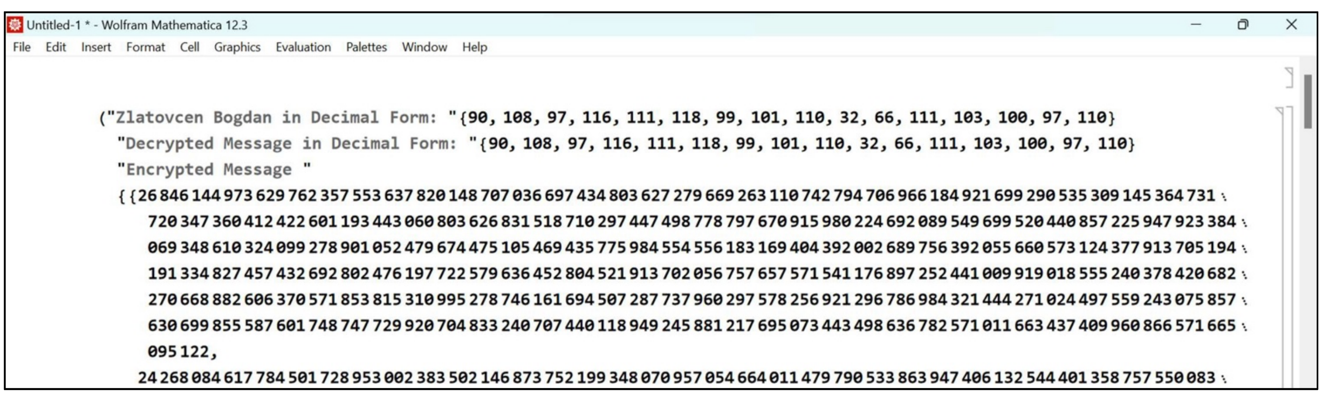

Figure 2 – *Program output for ElGamal Algorithm*

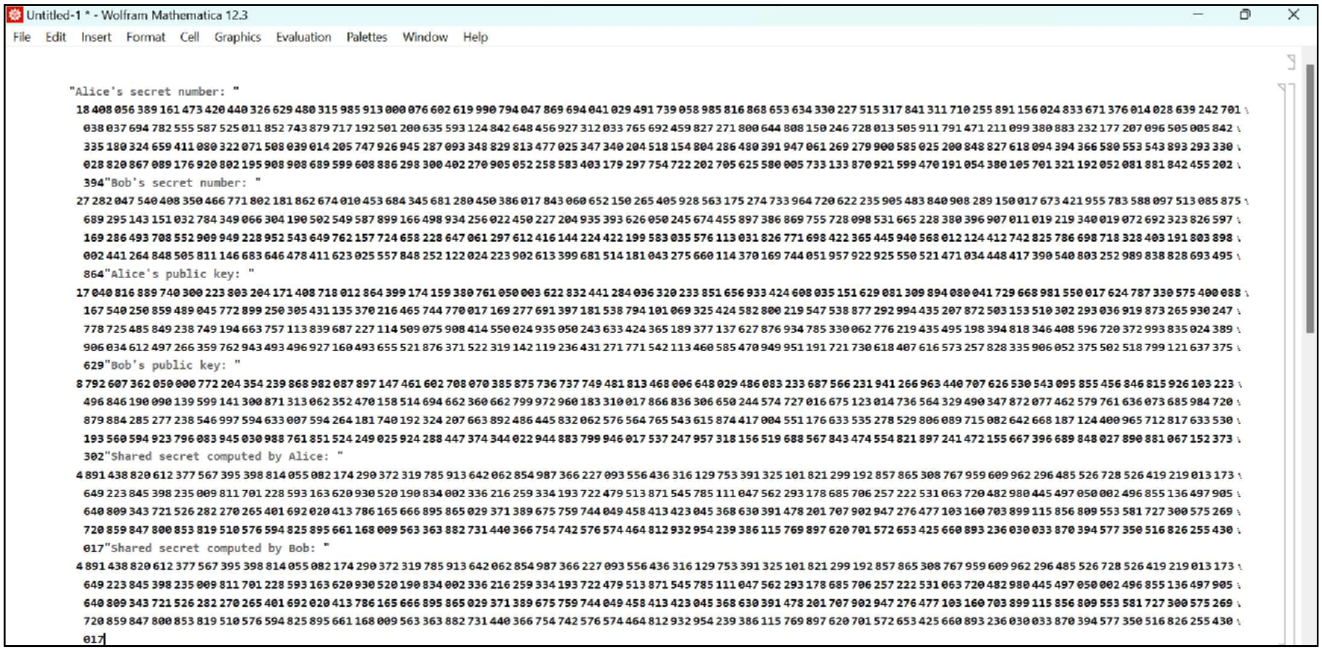

Figure 3 – *Program output for AES Algorithm*

Conclusions

In this laboratory work, we successfully implemented three fundamental cryptographic algorithms in Wolfram Mathematica: RSA, ElGamal, and the Diffie-Hellman key exchange. Each algorithm showcased distinct aspects of modern cryptography, from the RSA's reliance on the difficulty of factoring large numbers to ElGamal's use of discrete logarithms and Diffie-Hellman's secure key exchange over public channels. The practical application of these algorithms in Mathematica provided valuable insights into their operational mechanisms and the underlying mathematical principles.

Code link: https://github.com/CristianBrinza/UTM/tree/main/year3/cs/lab5