MINISTRY OF EDUCATION, CULTURE AND RESEARCH OF REPUBLIC OF MOLDOVA

TECHNICAL UNIVERSITY OF MOLDOVA

FACULTY OF COMPUTERS, INFORMATICS AND MICROELECTRONICS

DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATICS

Criptography and Security

Laboratory work 5: Cryptography with Public Keys

Elaborated:

st.gr. FAF-211 Grama Alexandru

Verified:

asist.univ. Catalin Mitu

Chișinău, 2023

Content

Algorithms **3**

RSA Algorithm 3

ElGamal Encryption 3

AES Algorithm 3

Implementation **5**

Code 5

Task 2.3 5

Conclusion **8**

RSA Algorithm

The RSA algorithm is one of the first public-key cryptosystems and is widely used for secure data transmission. Named after its inventors, Ron Rivest, Adi Shamir, and Leonard Adleman, it was introduced in 1977. RSA is based on the mathematical difficulty of factoring large integers, which is the product of two large prime numbers.

The RSA algorithm involves four steps: key generation, key distribution, encryption, and decryption.

- **Key Generation**: RSA begins by selecting two large prime numbers, p and q, and calculating their product $N = pq$, which is called the modulus. The totient of N is computed as $\phi(N) = (p-1)(q-1)$. Then, an integer e is chosen such that $1 < e < \phi(N)$ and e is coprime to $\phi(N)$. The pair (e, N) serves as the public key. The private key is computed as d, which is the modular multiplicative inverse of e modulo $\phi(N)$.
- **Key Distribution**: The public key (e, N) is distributed openly, while the private key d is kept secret.
- **Encryption**: To encrypt a message M, the sender computes the ciphertext C using the recipient's public key with $C = M^e \mod N$.
- **Decryption**: The recipient can decrypt the ciphertext C using their private key d with $M = C^d \mod N$, recovering the original message M.

RSA's security relies on the computational difficulty of the integer factorization problem, and as such, the keys used in RSA encryption need to be sufficiently large to prevent factorization by modern computing methods.

ElGamal Encryption

ElGamal encryption is a public-key cryptosystem developed by Taher Elgamal in 1985. It is based on the Diffie-Hellman key exchange and uses the discrete logarithm problem as its foundation, which is considered difficult to solve.

ElGamal encryption consists of three components: key generation, encryption, and decryption.

- **Key Generation**: The user generates a key pair consisting of a private key x and a public key (p, g, h), where p is a large prime number, g is a primitive root modulo p, and $h = g^x \mod p$.
- **Encryption**: To encrypt a message M, the sender selects a random integer k and computes the shared secret $s = h^k \mod p$. The ciphertext is then a pair (C_1, C_2), where $C_1 = g^k \mod p$ and $C_2 = M \cdot s \mod p$.
- **Decryption**: The recipient uses their private key x to compute $s = C_1^x \mod p$ and then retrieves the message M by calculating $M = C_2 \cdot s^{-1} \mod p$.

ElGamal is unique in that it provides an element of randomness in the encryption process, which results in different ciphertexts for the same plaintext message when encrypted multiple times.

AES Algorithm

The Advanced Encryption Standard (AES) is a symmetric key encryption algorithm established as an encryption standard by the U.S. National Institute of Standards and Technology (NIST) in 2001. AES is a variant of the Rijndael cipher developed by two Belgian cryptographers, Vincent Rijmen and Joan Daemen.

AES operates on a fixed block size of 128 bits and uses keys of 128, 192, or 256 bits, known as AES-128, AES-192, and AES-256, respectively. The AES encryption process involves several rounds of processing for each block of data, with the number of rounds depending on the key size: 10 rounds for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys.

Each round consists of four stages:

1. **SubBytes**: A non-linear substitution step where each byte is replaced with another according to a lookup table.
2. **ShiftRows**: A transposition step where each row of the state is shifted cyclically a certain number of steps.
3. **MixColumns**: A mixing operation which operates on the columns of the state, combining the four bytes in each column.
4. **AddRoundKey**: A simple bitwise XOR of the current block with a portion of the expanded key.

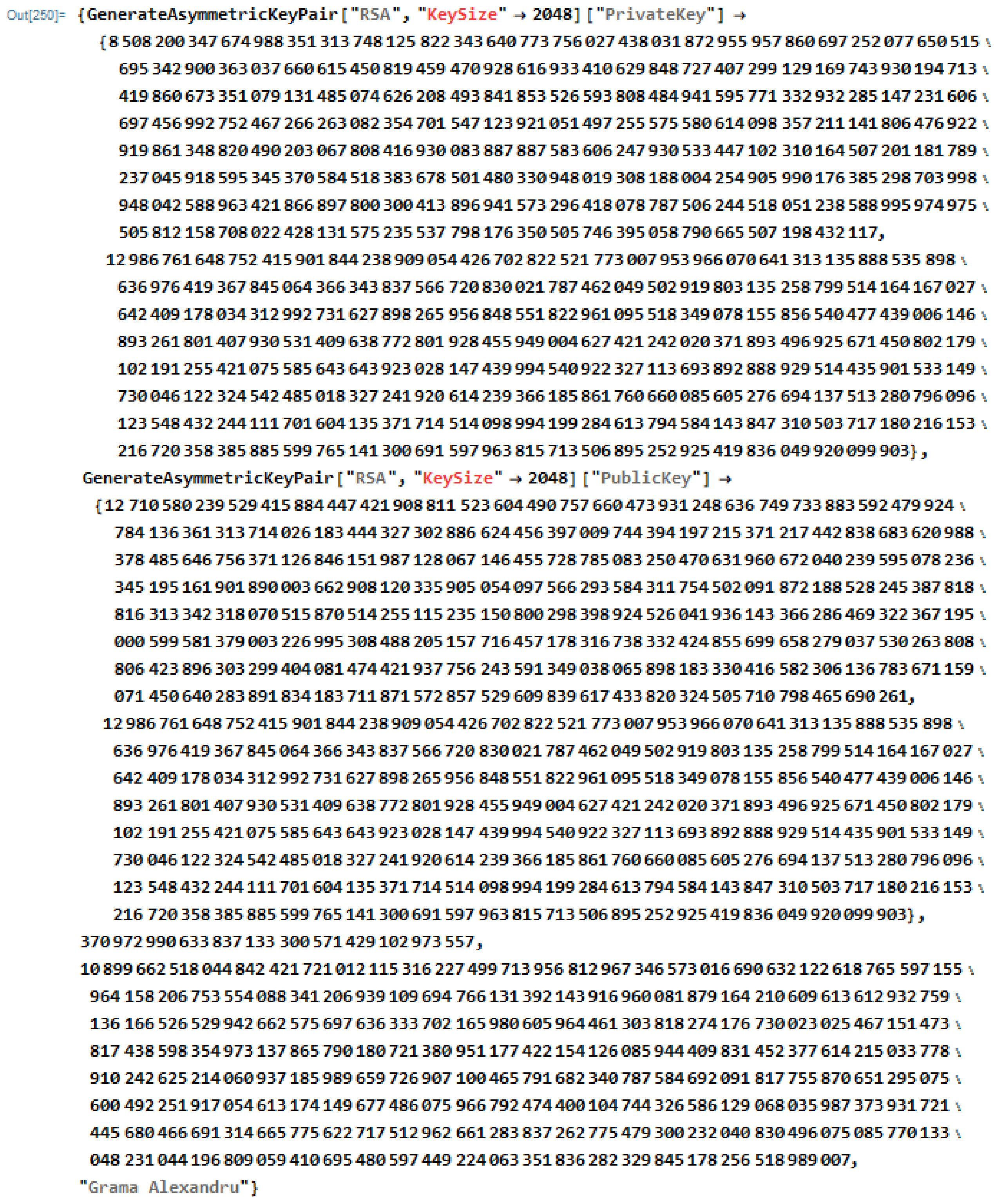
Before these rounds, an initial AddRoundKey stage is applied. After the final round, a slightly modified set of operations is applied to produce the ciphertext.

AES is widely adopted across the globe and is used in various applications to protect sensitive data due to its strength and efficiency in a wide range of hardware and software environments.

Implementation

Task 2.1

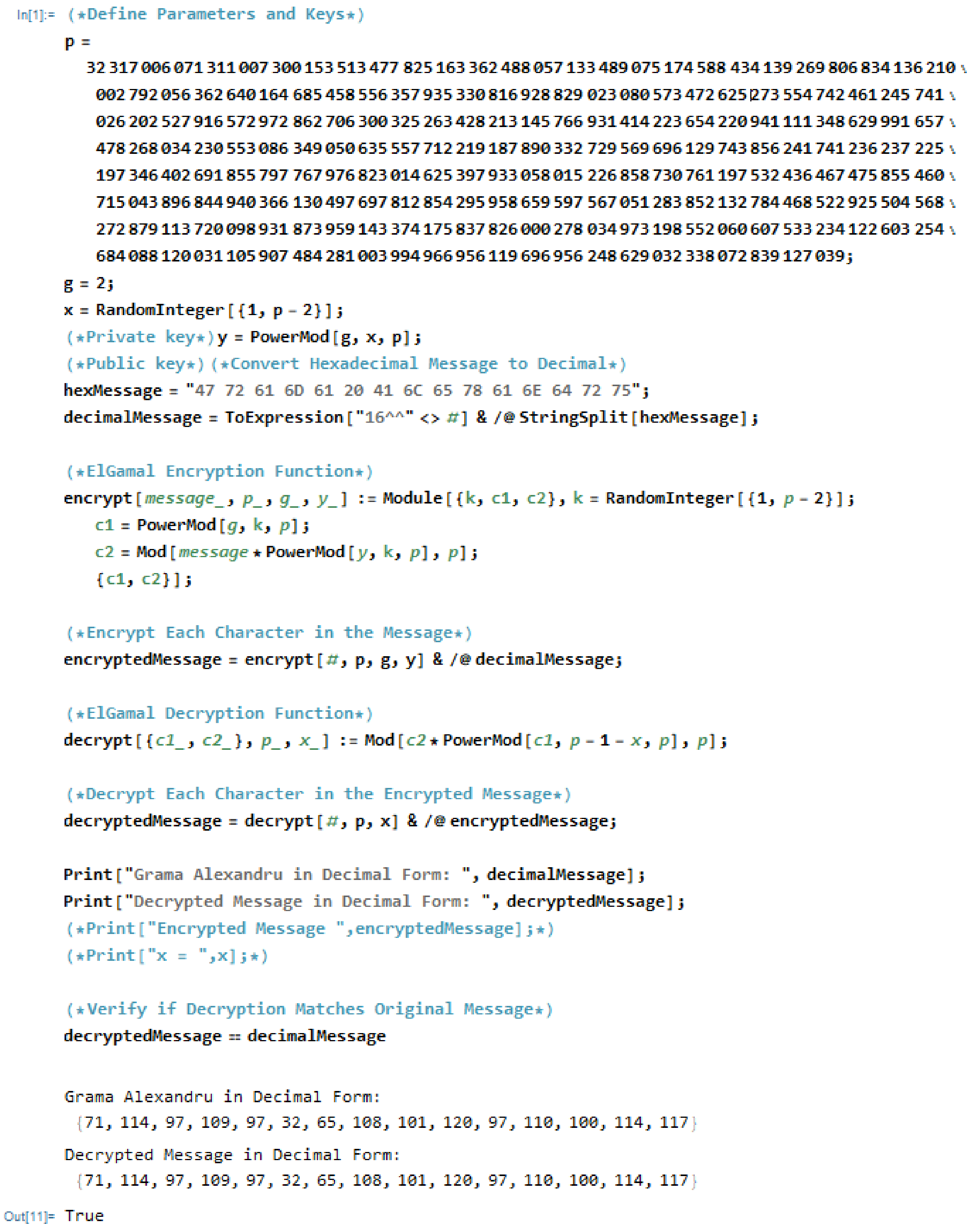
Utilizând platforma wolframalpha.com sau aplicația Wolfram Mathematica, generați cheile și realizați criptarea și decriptarea mesajului m = Nume Prenume aplicând algoritmul RSA. Valoarea lui n trebuie să fie de cel puțin 2048 biți.



```
Out[250]= {GenerateAsymmetricKeyPair["RSA", "KeySize" → 2048]["PrivateKey"] →
            {8 508 200 347 674 988 351 313 748 125 822 343 640 773 756 027 438 031 872 955 957 860 697 252 077 650 515 \
              695 342 900 363 037 660 615 450 819 459 470 928 616 933 410 629 848 727 407 299 129 169 743 930 194 713 \
              419 860 673 351 079 131 485 074 626 208 493 841 853 526 593 808 484 941 595 771 332 932 285 147 231 606 \
              697 456 992 752 467 266 263 082 354 701 547 123 921 051 497 255 575 580 614 098 357 211 141 806 476 922 \
              919 861 348 820 490 203 067 808 416 930 083 887 887 583 606 247 930 533 447 102 310 164 507 201 181 789 \
              237 045 918 595 345 370 584 518 383 678 501 480 330 948 019 308 188 004 254 905 990 176 385 298 703 998 \
              948 042 588 963 421 866 897 800 300 413 896 941 573 296 418 078 787 506 244 518 051 238 588 995 974 975 \
              505 812 158 708 022 428 131 575 235 537 798 176 350 505 746 395 058 790 665 507 198 432 117,
             12 986 761 648 752 415 901 844 238 909 054 426 702 822 521 773 007 953 966 070 641 313 135 888 535 898 \
              636 976 419 367 845 064 366 343 837 566 720 830 021 787 462 049 502 919 803 135 258 799 514 164 167 027 \
              642 409 178 034 312 992 731 627 898 265 956 848 551 822 961 095 518 349 078 155 856 540 477 439 006 146 \
              893 261 801 407 930 531 409 638 772 801 928 455 949 004 627 421 242 020 371 893 496 925 671 450 802 179 \
              102 191 255 421 075 585 643 643 923 028 147 439 994 540 922 327 113 693 892 888 929 514 435 901 533 149 \
              730 046 122 324 542 485 018 327 241 920 614 239 366 185 861 760 660 085 605 276 694 137 513 280 796 096 \
              123 548 432 244 111 701 604 135 371 714 514 098 994 199 284 613 794 584 143 847 310 503 717 180 216 153 \
              216 720 358 385 885 599 765 141 300 691 597 963 815 713 506 895 252 925 419 836 049 920 099 903},
          GenerateAsymmetricKeyPair["RSA", "KeySize" → 2048]["PublicKey"] →
           {12 710 580 239 529 415 884 447 421 908 811 523 604 490 757 660 473 931 248 636 749 733 883 592 479 924 \
             784 136 361 313 714 026 183 444 327 302 886 624 456 397 009 744 394 197 215 371 217 442 838 683 620 988 \
             378 485 646 756 371 126 846 151 987 128 067 146 455 728 785 083 250 470 631 960 672 040 239 595 078 236 \
             345 195 161 901 890 003 662 908 120 335 905 054 097 566 293 584 311 754 502 091 872 188 528 245 387 818 \
             816 313 342 318 070 515 870 514 255 115 235 150 800 298 398 924 526 041 936 143 366 286 469 322 367 195 \
             000 599 581 379 003 226 995 308 488 205 157 716 457 178 316 738 332 424 855 699 658 279 037 530 263 808 \
             806 423 896 303 299 404 081 474 421 937 756 243 591 349 038 065 898 183 330 416 582 306 136 783 671 159 \
             071 450 640 283 891 834 183 711 871 572 857 529 609 839 617 433 820 324 505 710 798 465 690 261,
            12 986 761 648 752 415 901 844 238 909 054 426 702 822 521 773 007 953 966 070 641 313 135 888 535 898 \
             636 976 419 367 845 064 366 343 837 566 720 830 021 787 462 049 502 919 803 135 258 799 514 164 167 027 \
             642 409 178 034 312 992 731 627 898 265 956 848 551 822 961 095 518 349 078 155 856 540 477 439 006 146 \
             893 261 801 407 930 531 409 638 772 801 928 455 949 004 627 421 242 020 371 893 496 925 671 450 802 179 \
             102 191 255 421 075 585 643 643 923 028 147 439 994 540 922 327 113 693 892 888 929 514 435 901 533 149 \
             730 046 122 324 542 485 018 327 241 920 614 239 366 185 861 760 660 085 605 276 694 137 513 280 796 096 \
             123 548 432 244 111 701 604 135 371 714 514 098 994 199 284 613 794 584 143 847 310 503 717 180 216 153 \
             216 720 358 385 885 599 765 141 300 691 597 963 815 713 506 895 252 925 419 836 049 920 099 903},
          370 972 990 633 837 133 300 571 429 102 973 557,
          10 899 662 518 044 842 421 721 012 115 316 227 499 713 956 812 967 346 573 016 690 632 122 618 765 597 155 \
           964 158 206 753 554 088 341 206 939 109 694 766 131 392 143 916 960 081 879 164 210 609 613 612 932 759 \
           136 166 526 529 942 662 575 697 636 333 702 165 980 605 964 461 303 818 274 176 730 023 025 467 151 473 \
           817 438 598 354 973 137 865 790 180 721 380 951 177 422 154 126 085 944 409 831 452 377 614 215 033 778 \
           910 242 625 214 060 937 185 989 659 726 907 100 465 791 682 340 787 584 692 091 817 755 870 651 295 075 \
           600 492 251 917 054 613 174 149 677 486 075 966 792 474 400 104 744 326 586 129 068 035 987 373 931 721 \
           445 680 466 691 314 665 775 622 717 512 962 661 283 837 262 775 479 300 232 040 830 496 075 085 770 133 \
           048 231 044 196 809 059 410 695 480 597 449 224 063 351 836 282 329 845 178 256 518 989 007,
          "Grama Alexandru"}
```

Task 2.2

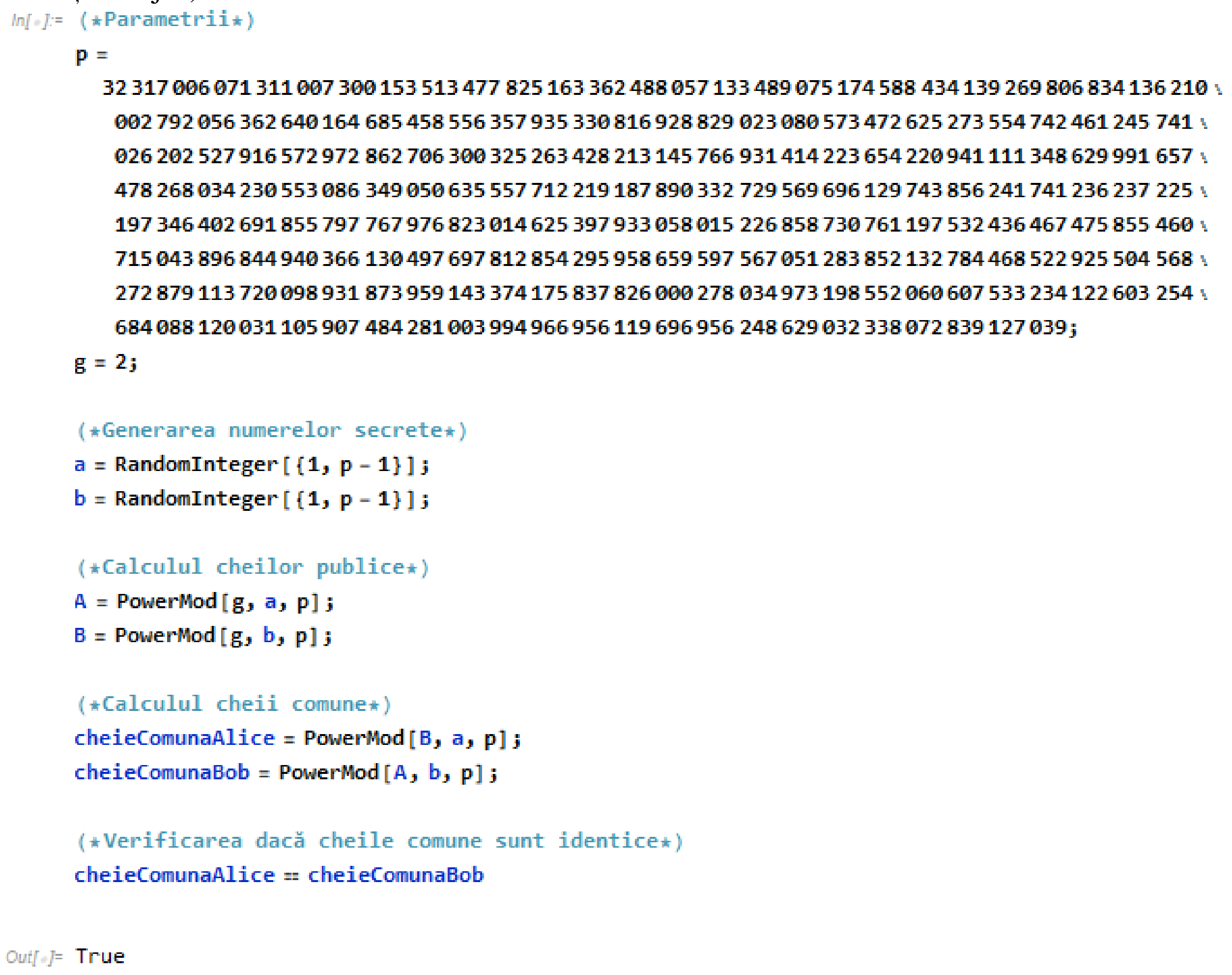
Utilizând platforma wolframalpha.com sau aplicația Wolfram Mathematica, generați cheile și realizați criptarea și decriptarea mesajului m = Nume Prenume aplicând algoritmul ElGamal (p și generatorul sunt dați mai jos).



```
In[1]:= (*Define Parameters and Keys*)
        p =
          32 317 006 071 311 007 300 153 513 477 825 163 362 488 057 133 489 075 174 588 434 139 269 806 834 136 210 \
           002 792 056 362 640 164 685 458 556 357 935 330 816 928 829 023 080 573 472 625 273 554 742 461 245 741 \
           026 202 527 916 572 972 862 706 300 325 263 428 213 145 766 931 414 223 654 220 941 111 348 629 991 657 \
           478 268 034 230 553 086 349 050 635 557 712 219 187 890 332 729 569 696 129 743 856 241 741 236 237 225 \
           197 346 402 691 855 797 767 976 823 014 625 397 933 058 015 226 858 730 761 197 532 436 467 475 855 460 \
           715 043 896 844 940 366 130 497 697 812 854 295 958 659 597 567 051 283 852 132 784 468 522 925 504 568 \
           272 879 113 720 098 931 873 959 143 374 175 837 826 000 278 034 973 198 552 060 607 533 234 122 603 254 \
           684 088 120 031 105 907 484 281 003 994 966 956 119 696 956 248 629 032 338 072 839 127 039;
        g = 2;
        x = RandomInteger[{1, p - 2}];
        (*Private key*)y = PowerMod[g, x, p];
        (*Public key*) (*Convert Hexadecimal Message to Decimal*)
        hexMessage = "47 72 61 6D 61 20 41 6C 65 78 61 6E 64 72 75";
        decimalMessage = ToExpression["16^^" <> #] & /@ StringSplit[hexMessage];

        (*ElGamal Encryption Function*)
        encrypt[message_, p_, g_, y_] := Module[{k, c1, c2}, k = RandomInteger[{1, p - 2}];
           c1 = PowerMod[g, k, p];
           c2 = Mod[message * PowerMod[y, k, p], p];
           {c1, c2}];

        (*Encrypt Each Character in the Message*)
        encryptedMessage = encrypt[#, p, g, y] & /@ decimalMessage;

        (*ElGamal Decryption Function*)
        decrypt[{c1_, c2_}, p_, x_] := Mod[c2 * PowerMod[c1, p - 1 - x, p], p];

        (*Decrypt Each Character in the Encrypted Message*)
        decryptedMessage = decrypt[#, p, x] & /@ encryptedMessage;

        Print["Grama Alexandru in Decimal Form: ", decimalMessage];
        Print["Decrypted Message in Decimal Form: ", decryptedMessage];
        (*Print["Encrypted Message ",encryptedMessage];*)
        (*Print["x = ",x];*)

        (*Verify if Decryption Matches Original Message*)
        decryptedMessage == decimalMessage


        Grama Alexandru in Decimal Form:
         {71, 114, 97, 109, 97, 32, 65, 108, 101, 120, 97, 110, 100, 114, 117}

        Decrypted Message in Decimal Form:
         {71, 114, 97, 109, 97, 32, 65, 108, 101, 120, 97, 110, 100, 114, 117}

Out[11]= True
```

Task 3

Utilizând platforma wolframalpha.com sau aplicația Wolfram Mathematica, realizați schimbul de chei Diffie-Helman între Alisa și Bob, care utilizează algoritmul AES cu cheia de 256 de biți. Numerele secrete a și b trebuie să fie alese în mod aleatoriu în conformitate cu cerințele algoritmului (p și generatorul sunt dați mai jos).



```
In[ ]:= (*Parametrii*)
        p =
          32 317 006 071 311 007 300 153 513 477 825 163 362 488 057 133 489 075 174 588 434 139 269 806 834 136 210 \
           002 792 056 362 640 164 685 458 556 357 935 330 816 928 829 023 080 573 472 625 273 554 742 461 245 741 \
           026 202 527 916 572 972 862 706 300 325 263 428 213 145 766 931 414 223 654 220 941 111 348 629 991 657 \
           478 268 034 230 553 086 349 050 635 557 712 219 187 890 332 729 569 696 129 743 856 241 741 236 237 225 \
           197 346 402 691 855 797 767 976 823 014 625 397 933 058 015 226 858 730 761 197 532 436 467 475 855 460 \
           715 043 896 844 940 366 130 497 697 812 854 295 958 659 597 567 051 283 852 132 784 468 522 925 504 568 \
           272 879 113 720 098 931 873 959 143 374 175 837 826 000 278 034 973 198 552 060 607 533 234 122 603 254 \
           684 088 120 031 105 907 484 281 003 994 966 956 119 696 956 248 629 032 338 072 839 127 039;
        g = 2;

        (*Generarea numerelor secrete*)
        a = RandomInteger[{1, p - 1}];
        b = RandomInteger[{1, p - 1}];

        (*Calculul cheilor publice*)
        A = PowerMod[g, a, p];
        B = PowerMod[g, b, p];

        (*Calculul cheii comune*)
        cheieComunaAlice = PowerMod[B, a, p];
        cheieComunaBob = PowerMod[A, b, p];

        (*Verificarea dacă cheile comune sunt identice*)
        cheieComunaAlice == cheieComunaBob

Out[ ]= True
```

Conclusion

In conclusion, the RSA, ElGamal, and AES algorithms each play a pivotal role in the field of cryptography, safeguarding information in the digital age. RSA's reliance on the factorization of large primes provides a strong basis for public-key encryption and digital signatures. ElGamal's use of the discrete logarithm problem offers a secure alternative for asymmetric encryption with the added benefit of cryptographic randomness. Meanwhile, AES stands as the benchmark for symmetric key encryption, providing a robust and efficient solution for protecting the confidentiality of data worldwide.

Each algorithm demonstrates the intricate balance between mathematical complexity and computational feasibility, embodying the principles that make modern cryptography a cornerstone of digital security. As threats to information security grow more sophisticated, the importance of these cryptographic algorithms cannot be overstated. They are not only fundamental to securing communication and data but also serve as a continuous incentive for the development of new cryptographic techniques and the enhancement of existing ones.

The ongoing evolution of cryptography is vital to meet the security demands of future technologies. As such, the study and advancement of cryptographic algorithms like RSA, ElGamal, and AES will remain a critical area of research in the quest to defend against the ever-evolving landscape of cyber threats.
Github: