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 Gazea Sandu
Verified:
asist.univ. Catalin Mitu

Chișinău, 2023

# Content

**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.

```
{privateKey →
  {4 808 863 132 193 686 774 348 693 631 716 557 568 084 075 554 494 250 053 152 594 104 149 217 133 218 119 306 645 506 518 122 038 207 457 772 585 ↖
    497 589 261 390 083 074 489 737 056 874 403 206 656 680 610 136 318 431 700 664 073 724 362 031 223 563 461 998 827 835 005 465 280 888 376 206 ↖
    089 330 269 900 925 650 558 117 581 637 150 640 067 038 926 510 319 343 533 078 345 322 792 136 589 984 862 648 495 742 400 027 731 575 321 785 ↖
    575 421 216 701 292 442 685 193 137 157 264 876 587 979 387 736 056 643 879 862 479 043 223 803 782 751 011 712 688 511 718 284 963 431 765 863 ↖
    696 561 904 881 955 184 721 624 483 908 860 537 596 791 741 323 278 735 537 185 275 284 765 511 400 633 811 869 918 576 626 472 145 045 694 462 ↖
    139 740 332 063 489 173 551 728 988 946 752 620 049 796 966 325 425 783 629 407 064 458 776 019 431,
   19 497 597 681 204 530 974 849 555 070 347 320 373 176 874 368 611 084 740 923 476 307 872 983 874 957 026 449 445 927 290 186 664 562 907 680 ↖
    422 565 571 565 274 536 934 938 388 904 431 750 176 071 269 005 763 991 901 486 060 165 777 119 177 100 885 290 933 385 568 567 009 390 202 568 ↖
    408 360 011 334 066 517 295 836 817 774 530 916 641 433 414 810 745 667 798 803 289 770 156 810 097 239 705 264 722 195 167 540 114 192 799 704 ↖
    952 656 280 875 629 184 613 908 978 874 512 130 217 508 298 707 786 375 191 017 184 186 799 282 129 812 476 805 759 014 628 950 019 586 695 398 ↖
    371 579 728 749 872 981 999 818 950 361 345 482 877 161 455 073 064 685 345 104 710 961 478 996 572 418 158 083 072 295 928 690 157 551 474 366 ↖
    868 544 483 530 542 683 661 410 978 809 348 580 596 042 253 033 551 780 574 293 164 486 450 287 187 119},
 publicKey →
  {17 579 945 479 347 829 299 075 027 372 946 415 887 375 911 227 429 371 595 048 206 685 374 479 787 691 108 353 317 420 801 971 200 412 978 852 ↖
    001 309 404 240 253 554 024 836 662 551 393 937 080 776 574 254 580 393 034 926 633 623 052 754 409 090 755 448 115 682 842 406 262 906 248 491 ↖
    321 520 063 772 250 001 143 617 448 437 188 397 237 000 942 688 983 787 547 209 125 591 907 326 922 997 693 968 350 205 988 013 418 452 533 345 ↖
    114 996 908 784 608 000 172 467 966 240 213 346 393 287 655 501 893 242 337 820 006 168 038 957 778 733 312 906 124 558 022 372 581 507 592 880 ↖
    045 003 856 167 790 384 052 839 640 439 131 198 204 313 891 762 663 278 708 262 402 614 646 836 650 154 469 192 731 392 108 824 006 343 935 739 ↖
    482 951 629 589 796 054 172 726 606 045 291 347 721 771 231 529 362 244 531 836 036 136 132 465 001 863,
   19 497 597 681 204 530 974 849 555 070 347 320 373 176 874 368 611 084 740 923 476 307 872 983 874 957 026 449 445 927 290 186 664 562 907 680 ↖
    422 565 571 565 274 536 934 938 388 904 431 750 176 071 269 005 763 991 901 486 060 165 777 119 177 100 885 290 933 385 568 567 009 390 202 568 ↖
    408 360 011 334 066 517 295 836 817 774 530 916 641 433 414 810 745 667 798 803 289 770 156 810 097 239 705 264 722 195 167 540 114 192 799 704 ↖
    952 656 280 875 629 184 613 908 978 874 512 130 217 508 298 707 786 375 191 017 184 186 799 282 129 812 476 805 759 014 628 950 019 586 695 398 ↖
    371 579 728 749 872 981 999 818 950 361 345 482 877 161 455 073 064 685 345 104 710 961 478 996 572 418 158 083 072 295 928 690 157 551 474 366 ↖
    868 544 483 530 542 683 661 410 978 809 348 580 596 042 253 033 551 780 574 293 164 486 450 287 187 119},
 86 294 060 549 409 632 615 818 357,
 1 827 011 372 356 416 794 485 188 223 571 935 035 410 910 779 346 110 513 401 376 866 171 192 436 327 569 965 363 960 312 982 620 778 854 324 376 ↖
  654 632 230 942 464 006 856 708 882 327 528 100 331 039 557 758 845 935 434 096 102 074 197 253 031 652 457 060 668 177 511 979 795 268 303 807 ↖
  209 246 203 077 047 434 269 322 390 049 998 425 659 421 189 451 257 375 540 034 796 697 757 078 068 668 108 336 418 992 740 693 981 527 915 656 ↖
  684 866 481 361 551 811 063 022 753 334 758 766 252 811 280 825 217 772 843 991 507 509 621 836 557 744 554 177 494 663 608 647 485 090 595 030 ↖
  091 127 113 665 058 420 201 731 647 878 250 425 510 044 699 123 236 406 722 781 452 194 962 572 776 036 553 664 161 398 648 460 966 089 728 704 ↖
  875 580 845 678 229 070 171 978 189 292 603 689 325 135 855 598 795 354 563 157 274 815 107 427 564, "Gazea Sandu"}
```

### 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[76]:=
```
(*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 61 7A 65 61 20 53 61 6E 64 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["Gazea Sandu 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
```

```
Gazea Sandu in Decimal Form: {71, 97, 122, 101, 97, 32, 83, 97, 110, 100, 117}
Decrypted Message in Decimal Form: {71, 97, 122, 101, 97, 32, 83, 97, 110, 100, 117}
```

Out[86]=
```
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: