

# **Cryptography and Security**

## **Laboratory Work 5:** **Public Keys Cryptography**

Elaborated:  
st. gr. FAF-212

Cristian Brinza

Verified:  
asist. univ.

Mitu Catalin

## Objective

Study and implement the following public keys algorithms: RSA, ElGamal and Diffie-Helman. Implement the algorithms using Wolfram Mathematica. 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 ( $\phi$ ):** Euler's totient function  $\phi(n)$  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  $\phi(n)$ . The value of  $e$  must be coprime to  $\phi(n)$ , ensuring  $e$  and  $\phi(n)$  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  $e \times d \bmod \phi(n)$  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  $encryptedMessage = message^e \bmod n$ .
- 7) **Decryption:** The encrypted message is decrypted using the formula  $decryptedMessage = encryptedMessage^d \bmod n$ . 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).

$p=323170060713110073001535134778251633624880571334890751745884341392698068341362100027$   
920563626401646854585563579353308169288290230805734726252735547424612457410262025279165  
729728627063003252634282131457669314142236542209411113486299916574782680342305530863490  
506355577122191878903327295696961297438562417412362372251973464026918557977679768230146  
253979330580152268587307611975324364674758554607150438968449403661304976978128542959586  
595975670512838521327844685229255045682728791137200989318739591433741758378260002780349  
731985520606075332341226032546840881200311059074842810039949669561196969562486290323380  
72839127039

$g=2$

### **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  $y = g^x \bmod p$ .
- 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  $c1 = g^k \bmod p$ . The second part of the ciphertext,  $c2$ , is calculated as  $c2 = \text{message} \times y^k \bmod p$ . 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  $\text{decryptedMessage} = c2 \times c1^{p-1-x} \bmod p$ .
- 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).

$p=323170060713110073001535134778251633624880571334890751745884341392698068341362100027$   
920563626401646854585563579353308169288290230805734726252735547424612457410262025279165  
729728627063003252634282131457669314142236542209411113486299916574782680342305530863490  
506355577122191878903327295696961297438562417412362372251973464026918557977679768230146  
253979330580152268587307611975324364674758554607150438968449403661304976978128542959586  
595975670512838521327844685229255045682728791137200989318739591433741758378260002780349  
731985520606075332341226032546840881200311059074842810039949669561196969562486290323380  
72839127039

$g=2$

### **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  $A = g^a \bmod p$ . Bob computes his public key  $B$  as  $B = g^b \bmod p$ . These public keys are then shared between Alice and Bob.
- 4) Compute Shared Secret: Alice computes the shared secret using Bob's public key:  $sharedSecretAlice = B^a \bmod p$ . Bob computes the shared secret using Alice's public key:  $sharedSecretBob = A^b \bmod p$ . 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

```

"p value: "
96 412 691 989 533 200 550 512 870 579 828 225 086 767 516 441 701 667 677 050 179 479 317 446 445 074 725 120 784 083 388 923 103 009 071 781 526 960 364 190 368 565 304 030 829 \
644 613 652 513 868 850 268 653 817 947 941 597 963 810 228 582 563 277 758 628 324 561 573 223 013 282 423 766 770 044 810 091 594 773 362 503 939 218 034 662 314 605 343 504 \
110 404 469 431 081 116 039 238 581 571 519 402 489 561 "q value: "
145 328 875 552 750 940 109 712 117 969 918 109 245 343 367 426 771 999 305 668 151 014 956 598 593 443 246 843 713 019 093 952 177 556 830 092 763 354 424 300 439 914 919 206 709 \
559 779 395 591 162 899 174 053 935 033 069 577 741 218 215 039 285 456 251 409 402 036 212 679 729 876 124 565 414 508 030 228 521 170 140 135 463 138 877 471 457 206 781 167 \
792 598 329 313 375 782 271 911 284 718 607 090 993 991 "n value: "
14 011 548 115 852 577 946 878 003 557 244 585 536 129 133 659 382 248 707 479 858 062 485 408 882 929 396 773 303 193 134 182 058 837 254 319 111 368 257 220 710 805 249 970 170 \
120 266 940 901 901 441 648 531 740 201 623 270 936 025 883 135 564 138 661 136 257 459 074 546 702 912 908 869 239 666 253 680 889 968 662 028 864 732 275 154 243 086 575 624 \
432 935 225 010 473 827 541 764 999 564 199 424 676 368 142 422 891 554 895 605 083 668 928 497 376 468 307 515 563 365 808 847 111 118 541 891 258 340 208 035 684 986 460 845 \
947 390 572 008 285 387 926 275 744 557 005 634 795 034 211 287 832 893 836 113 839 600 324 450 584 530 673 194 674 986 360 138 631 960 901 436 073 806 566 556 154 492 527 674 \
762 456 470 357 998 684 442 746 323 511 738 861 539 751 987 732 649 537 816 305 347 309 742 742 327 987 491 227 951 "e value (public exponent): "
3 762 932 573 621 876 006 307 688 442 090 314 703 712 318 102 371 220 218 028 266 982 176 651 912 268 256 348 282 506 255 926 542 918 567 743 415 338 932 326 665 731 234 780 731 \
566 621 113 550 628 696 350 049 609 672 724 909 983 819 704 126 503 460 368 539 697 743 509 417 363 508 440 263 993 911 657 457 599 371 786 400 581 527 534 891 268 163 752 003 \
027 869 961 824 137 261 145 671 001 967 415 170 979 911 613 569 819 469 650 675 077 946 387 196 836 343 885 766 490 243 307 246 343 674 097 561 154 041 110 305 594 194 371 905 \
271 592 517 665 142 052 905 039 818 448 290 686 844 516 032 504 907 672 936 464 264 755 026 755 381 316 409 719 806 846 163 026 248 713 312 434 839 226 174 966 428 199 122 560 \
423 793 324 637 123 384 447 005 796 217 971 758 199 084 147 472 101 053 466 883 510 400 117 526 009 138 528 376 333 "d value (private exponent): "
12 646 324 519 617 862 508 914 647 669 445 901 879 187 451 345 298 435 736 135 030 619 367 379 430 722 964 746 174 090 322 753 855 709 993 696 346 350 788 593 085 077 853 509 581 \
212 436 027 752 006 225 571 377 706 929 595 695 217 150 627 479 766 341 680 266 338 498 824 609 059 032 488 563 772 210 585 798 067 826 565 584 779 792 739 790 993 567 849 187 \
573 314 880 227 925 444 730 775 188 934 655 052 449 065 517 019 292 300 334 854 813 807 535 275 397 840 622 426 984 979 631 940 022 336 646 714 749 357 951 396 266 531 173 936 \
010 960 083 130 292 414 337 911 632 610 816 061 495 280 079 876 922 335 831 810 236 988 872 135 875 443 920 094 050 829 433 282 597 829 913 564 342 896 749 088 046 944 620 896 \
652 811 301 235 203 540 028 599 777 821 903 007 516 547 849 811 156 486 372 668 469 722 805 048 025 192 641 094 197 "Decrypted Message: "Zlatovcen Bogdan"
"Encrypted Message: "
{543 729 562 630 587 174 102 078 124 132 730 028 426 548 501 848 384 113 358 707 513 382 711 894 352 881 630 057 631 996 588 129 519 400 848 398 587 160 458 740 070 006 714 272 \
684 718 545 023 517 679 021 342 674 993 323 428 097 012 884 781 786 019 007 658 289 238 964 171 428 424 703 652 223 000 639 685 796 100 614 019 800 109 665 816 135 585 320 500 \
182 377 051 548 552 776 034 689 938 166 191 564 173 884 911 262 168 978 840 630 674 817 688 228 861 446 389 931 586 521 870 446 679 750 478 274 342 335 648 280 171 568 319 886 \
405 879 241 758 908 422 413 237 065 620 150 333 749 246 196 368 310 980 939 349 203 954 035 644 714 780 781 940 306 517 013 121 895 148 178 652 099 604 361 580 847 398 528 826 \
264 199 566 067 967 342 162 631 142 077 677 386 841 799 150 638 416 212 073 851 824 885 507 320 654 338 083 206 140 \
996 283 892 825 397 726 097 478 106 918 831 494 757 123 291 564 218 059 504 124 637 789 116 938 856 514 362 818 088 295 134 404 725 578 726 433 276 647 526 507 541 435 430 446 \
}

```

Figure 1 – Program output for RSA Algorithm

```

("Zlatovcen Bogdan in Decimal Form: "{90, 108, 97, 116, 111, 118, 99, 101, 110, 32, 66, 111, 103, 100, 97, 110})
"Decrypted Message in Decimal Form: "{90, 108, 97, 116, 111, 118, 99, 101, 110, 32, 66, 111, 103, 100, 97, 110})
"Encrypted Message "
{26846 144 973 629 762 357 553 637 820 148 707 036 697 434 803 627 279 669 263 110 742 794 706 966 184 921 699 290 535 309 145 364 731 \
720 347 360 412 622 601 193 443 060 803 626 831 518 710 297 447 498 778 797 670 915 980 224 692 089 549 699 520 440 857 225 947 923 384 \
069 348 610 324 099 278 901 052 479 674 475 105 469 435 775 984 554 556 183 169 404 392 002 689 756 392 055 660 573 124 377 913 705 194 \
191 334 827 457 432 692 802 476 197 722 579 636 452 804 521 913 702 056 757 657 571 541 176 897 252 441 009 919 018 555 240 378 420 682 \
270 668 882 606 370 571 853 815 310 995 278 746 161 694 507 287 737 960 297 578 256 921 296 786 984 321 444 271 024 497 559 243 075 857 \
630 699 855 587 601 748 747 729 920 704 833 240 707 440 118 949 245 881 217 695 073 443 498 636 782 571 011 663 437 409 960 866 571 665 \
095 122,
24 268 084 617 784 501 728 953 002 383 502 146 873 752 199 348 070 957 054 664 011 479 790 533 863 947 406 132 544 401 358 757 550 083 \
}

```

Figure 2 – Program output for ElGamal Algorithm

```
Untitled-1 - Wolfram Mathematica 12.3
File Edit Insert Format Cell Graphics Evaluation Palettes Window Help

"Alice's secret number: "
18 408 056 389 161 473 426 440 326 629 480 315 985 913 000 076 602 619 990 794 047 869 694 041 029 491 739 058 985 816 868 653 634 330 227 515 317 841 311 710 255 891 156 024 833 671 376 014 028 639 242 781 \
038 037 694 782 555 587 525 011 852 743 879 717 192 501 200 635 593 124 842 648 456 927 312 033 765 692 459 827 271 800 644 808 150 246 728 013 505 911 791 471 211 099 380 883 232 177 207 096 505 005 842 \
335 180 324 659 411 080 322 071 508 039 014 205 747 926 945 287 093 348 829 813 477 025 347 340 204 518 154 804 286 480 391 947 061 269 279 900 585 025 200 848 827 618 094 394 366 580 553 543 893 293 330 \
028 820 867 089 176 920 802 195 908 908 689 599 608 886 290 300 402 270 905 052 258 583 403 179 297 754 722 202 705 625 580 005 733 133 870 921 599 470 191 054 380 105 701 321 192 052 081 881 842 455 202 \
394 "Bob's secret number: "
27 282 047 540 408 350 466 771 802 181 862 674 010 453 684 345 681 280 450 386 017 843 060 652 150 265 405 928 563 175 274 733 964 720 622 235 905 483 840 908 289 150 017 673 421 955 783 588 097 513 085 875 \
689 295 143 151 032 784 349 066 304 190 592 549 587 899 166 498 934 256 022 450 227 204 935 393 626 050 245 674 455 897 386 869 755 728 098 531 665 228 380 396 907 011 019 219 340 019 072 692 323 826 597 \
169 286 493 708 552 909 949 228 952 543 649 762 157 724 658 228 647 061 297 612 416 144 224 422 190 583 035 576 113 031 826 771 698 422 365 445 940 568 012 124 412 742 825 786 698 718 328 403 191 803 898 \
002 441 264 848 505 811 146 683 646 478 411 623 025 557 848 252 122 024 223 902 613 399 681 514 181 043 275 660 114 370 169 744 051 957 922 925 550 521 471 034 448 417 390 540 803 252 989 838 828 693 495 \
864 "Alice's public key: "
17 040 816 889 740 300 223 803 204 171 408 718 012 864 399 174 159 380 761 050 003 622 832 441 284 036 320 233 851 656 933 424 600 035 151 629 081 309 894 080 041 729 668 981 550 017 624 787 330 575 400 088 \
167 540 250 859 489 045 772 899 250 305 431 135 370 216 465 744 770 017 169 277 691 397 181 538 794 101 069 325 424 582 800 219 547 538 877 292 994 435 207 872 503 153 510 302 293 036 919 873 265 930 247 \
778 725 485 489 238 749 194 663 757 113 839 687 227 114 509 075 908 414 550 024 935 050 243 633 424 365 189 377 137 627 876 934 785 330 062 776 219 435 495 198 394 818 346 408 596 720 372 993 835 024 389 \
906 034 612 497 266 359 762 943 493 496 927 160 493 655 521 876 371 522 319 142 119 236 431 271 771 542 113 460 585 470 949 951 191 721 730 618 407 616 573 257 828 335 906 052 375 502 518 799 121 637 375 \
629 "Bob's public key: "
8 792 607 362 050 000 772 204 354 239 868 982 087 897 147 461 602 708 070 385 875 736 737 749 481 813 468 006 648 029 486 083 233 687 566 231 941 266 963 440 707 626 530 543 095 855 456 846 815 926 103 223 \
496 846 190 090 139 599 141 300 871 313 062 352 470 158 514 694 662 360 662 799 972 960 183 310 017 866 836 306 650 244 574 727 016 675 123 014 736 564 329 490 347 872 077 462 579 761 636 073 685 984 720 \
879 884 285 277 238 546 997 594 633 007 594 264 181 740 192 324 207 663 892 486 445 832 062 576 564 765 543 615 874 417 004 551 176 633 535 278 529 806 089 715 082 642 668 187 124 400 965 712 817 633 530 \
193 560 594 923 796 083 945 030 988 761 851 524 249 025 924 288 447 374 344 022 944 883 799 946 017 537 247 957 318 156 519 688 567 843 474 554 821 897 241 472 155 667 396 689 848 027 890 881 067 152 373 \
302 "Shared secret computed by Alice: "
4 891 438 820 612 377 567 395 398 814 055 082 174 290 372 319 785 913 642 062 854 987 366 227 093 556 436 316 129 753 391 325 101 821 299 192 857 865 308 767 959 609 962 296 485 526 728 526 419 219 013 173 \
649 223 845 398 235 009 811 701 228 593 163 620 930 520 190 834 002 336 216 259 334 193 722 479 513 871 545 785 111 047 562 293 178 685 706 257 222 531 063 720 482 980 445 497 050 002 496 855 136 497 905 \
640 809 343 721 526 282 270 265 401 692 020 413 786 165 666 895 865 029 371 389 675 759 744 049 458 413 423 045 368 630 391 478 201 707 902 947 276 477 103 160 703 899 115 856 809 553 581 727 300 575 269 \
720 859 847 800 853 819 510 576 594 825 895 661 168 009 563 363 882 731 440 366 754 742 576 574 464 812 932 954 239 386 115 769 897 620 701 572 653 425 660 893 236 030 033 870 394 577 350 516 826 255 430 \
017 "Shared secret computed by Bob: "
4 891 438 820 612 377 567 395 398 814 055 082 174 290 372 319 785 913 642 062 854 987 366 227 093 556 436 316 129 753 391 325 101 821 299 192 857 865 308 767 959 609 962 296 485 526 728 526 419 219 013 173 \
649 223 845 398 235 009 811 701 228 593 163 620 930 520 190 834 002 336 216 259 334 193 722 479 513 871 545 785 111 047 562 293 178 685 706 257 222 531 063 720 482 980 445 497 050 002 496 855 136 497 905 \
640 809 343 721 526 282 270 265 401 692 020 413 786 165 666 895 865 029 371 389 675 759 744 049 458 413 423 045 368 630 391 478 201 707 902 947 276 477 103 160 703 899 115 856 809 553 581 727 300 575 269 \
720 859 847 800 853 819 510 576 594 825 895 661 168 009 563 363 882 731 440 366 754 742 576 574 464 812 932 954 239 386 115 769 897 620 701 572 653 425 660 893 236 030 033 870 394 577 350 516 826 255 430 \
017
```

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>