Ministry of Education, Research and Culture
Technical University of Moldova
Software Engineering and Automation Departments

REPORT

Laboratory work No. 5

Course: Cryptography and Security

Theme: Public Key Encryption

Author: Corneliu Catlabuga
FAF-213

Checked by: univ. assist.
Cătălin Mîțu

Chisinau 2023

Objective:

Use different types of Public Key Encryption to encrypt, send and decrypt a message.

Task:

1. Encrypt a message using the RSA algorithm, send it and decrypt it on the other side

2. Encrypt a message using the ElGamal algorithm, send it and decrypt it on the other side

3. Execute a private key exchange using the Deiffie-Hleman algorithm

Theoretical considerations:

For task 1 and 2:

Text Message = “Catlabuga Corneliu”

ASCII to Hex = 4361746C616275676120436F726E656C6975

Hex to Decimal = 5869685300210033754487172484267118152149365

For tasks 2 and 3:

p=32317006071311007300153513477825163362488057133489075174588434139269806834136210002792056362640164685458556357935330816928829023080573472625273554742461245741026202527916572972862706300325263428213145766931414223654220941111348629991657478268034230553086349050635557712219187890332729569696129743856241741236237225197346402691855797767976823014625397933058015226858730761197532436467475855460715043896844940366130497697812854295958659597567051283852132784468522925504568272879113720098931873959143374175837826000278034973198552060607533234122603254684088120031105907484281003994966956119696956248629032338072839127039

g=2

Implementation, practical results:

RSA Algorithm:

1. Generate p and q:

p =
118627057758818984582391802186227731251330008694114664532785920230053589489
970684346830022240183229928691904969307726319383151658714265136875401323602
535487092925298958551672745241516985174705936390232366714840354542075868240
267108923739954225838570270738205471509778691919168011637528941048130670233
35009573975944576201511103460657

q =
213584648877534801497198062624997342707178803825815335195171409072934487935
195560050411190072442864914120820588827256751496197186451590566960737398739
605183451093367407205908084543945229016023092297114105992220929248778552270
633335597240930691632046445756236459893847533535486260892807 9

2. Compute n p * q:

n =
253369184787923933003294469911507970782826031251579435753589320488827952323
613390773308873940499938306663864467490191801702736655131702754352848021333
306464749305158520667084279657516307867092927680095515695783300395425550910
742878265968627968598047860902764444793294592275341200697445699726941428344
208174717118857621608605162946462387585865621882234587668623208802674007135
514006271131458430979103785473354754339757428702734441831608048067710307409
439333252421122051150216810515681974469456134350438510578918059232577772294
528947693534174559034268535372892342986766070651920782263889149276893674466
23791802119087903

3. Compute f = (p – 1) * (q – 1):

f =
126684592393961966501647234955753985391413015625789717876794660244413976161
806695386654436970249969153331932233745095900851368327565851377176424010666
653232374652579260333542139828758153933546463840047757847891650197712775455
371439132984313984299023930451382222396647296137670600348722790549941834762
611796162658335696938676916468873457228222541240927407136435369192028229616
445620753841624692551251305180226023338001022279307377008296715260867328217
665580930515360738734484182352665611878815978627587921951111698800376215015
536698856332107130318940450726954920707702224320854723474540087767582265133
56117714203349584

4. Generate e for which gcd(f, e) = 1:

e = 65537

5. Generate $d \equiv e^{-1} \pmod{f}$:

d =
7593884694442522321768179234992134696586142665760027262141810317380720895855129206806414020135602847534438497036294142949701702471007936512772658588338606029934581345258713708230728846418480206344440966426681757599550095951545336201442374193010340960232818942897679614612765902520255084649517776185436019976883337226077232590236781808394053317259944509497466871353142238736106422648755158499850276801690655933788292674611658437685232672321014347603691033075751554002504210157779154475430152868944847630367348861082095273825297708492257818985633427797437796090849447002277255257048228187661854325473811664928246312593

6. Encrypt the message using c ≡ m^e (mod n):

msg = 586968530021003375448717248426711815214936 5
c =
704963103608252899497188307636010329795653990075516819873564308098098969437510733489160974187262713191161034598945504120607820249782622709067973733121424653464266122106851226567967641417113518779353464168193561720641898892314612550574991137335019358770532048788445818388457921301599000873243225345894445129848250332546978213313201993817373810623633310864851877466527518113267606236605676321212830390350341899974200550651685056001640326489974956718746102104588714602839199180858977254490455215571612520312360259396624862873801554149676168461098822336106446660632389218829105296444020206818392353101729801235415024 2166

7. Decrypt the message using m ≡ c^d (mod n):

m = 5869685300210033754487172484267118152149365

ElGamal Algorithm

1. Bob generates p, g and PrivK_B:

p =
323170060713110073001535134778251633624880571334890751745884341392698068341
362100027920563626401646854585563579353308169288290230805734726252735547424
612457410262025279165729728627063003252634282131457669314142236542209411113
486299916574782680342305530863490506355577122191878903327295696961297438562
417412362372251973464026918557977679768230146253979330580152268587307611975
324364674758554607150438968449403661304976978128542959586595975670512838521
327844685229255045682728791137200989318739591433741758378260002780349731985
520606075332341226032546840881200311059074842810039949669561196969562486290
32338072839127039

g = 2

PrivK_B = 963

2. Bob computes PublK_B = b^{PrivK_B} mod p:

PublK_B =
779625120911999926428270591030015064870098148607600602149432516577035895261
314088197249205270560820738024393298512693454676733589216247523726238983705
012273562502215996517842389663172439204291868223967478337470309894847834031
58999565970908923751724902621910424834220376654628719935312887808

3. Bob sends p, g and PublK_B to Alice

4. Alice generates PrivK_A:

PrivK_A = 651

5. Alice computes PublK_A = g^{PrivK_A} mod p:

PublK_A =
934387838489025580777711944847419663338133198284505073782618627665771554244
337128756410943757797662674665945000672134617229046726937689702042145038279
109465754008509308982261776972634572104453324 8

6. Alice computes MaskK = $PublK\_B^{PrivK\_A}$ mod p:

MaskK =
183499442240171205709388431175917212520692692011574667034048057397670750688
810383719428056484789668093552676709779269814901425158871915462082034281429
964619847481432129880129556570402742359060385980858747968767342472773645776
820575219625791805977244005223593856104768074767191802411728118417950263722
978178761376781813883845125576433888444923305518689037461542592626874514588
629756637592327808178023683271556121860798345917737570456906113872631821575
518860685591872593930988412481442445242249225546162053530461194819761878520
406547429428771817156443380199069329259005256108734654048846157464470840662
99447004906951519

7. Alice encrypts the message c = msg * MaskK mod p:

msg = 5869685300210033754487172484267118152149365

c =
107708397871387307243567855353896682439608577761644563005443807667316077425
835579873671204198312033960026736294031260089545653055892915509628846077288
701770473171424583332817838914809813209704190935603086673561372200824146760
892019405197534577468559073491258690162260032126698446105322527028277564436
742448913114316877144255303994928357734949513137662928008378271311488490628
899915126425034635180355144587740588553621187665669962738728282247965126914
382315101386929029251254081528741647574347463953138393149027355530350564652
084230615323949393583624336513541141454728908129678248542165117083298766278
363909890868784552209192725985074560514299919596799701635435

8. Alice sends PublK_A and c to Bob.

9. Bob computes MaskK = $PublK\_A^{PrivK\_B}$ mod p:

MaskK =
183499442240171205709388431175917212520692692011574667034048057397670750688
810383719428056484789668093552676709779269814901425158871915462082034281429
964619847481432129880129556570402742359060385980858747968767342472773645776
820575219625791805977244005223593856104768074767191802411728118417950263722
978178761376781813883845125576433888444923305518689037461542592626874514588
629756637592327808178023683271556121860798345917737570456906113872631821575
518860685591872593930988412481442445242249225546162053530461194819761878520
406547429428771817156443380199069329259005256108734654048846157464470840662
99447004906951519

10. Bob decripts the message msg = c * $MaskK^{-1}$ mod p:

msg = 5869685300210033754487172484267118152149365

Diffie-Helman Exchange

1. Alice and Bob exchange p and g:

p =
323170060713110073001535134778251633624880571334890751745884341392698068341
362100027920563626401646854585563579353308169288290230805734726252735547424
612457410262025279165729728627063003252634282131457669314142236542209411113
486299916574782680342305530863490506355577122191878903327295696961297438562
417412362372251973464026918557977679768230146253979330580152268587307611975
324364674758554607150438968449403661304976978128542959586595975670512838521
327844685229255045682728791137200989318739591433741758378260002780349731985
520606075332341226032546840881200311059074842810039949669561196969562486290
32338072839127039

g = 2

2. Alice generates a and computes PublA = g^a mod p:

a = 62

PublA = 4611686018427387904

3. Bob generates b and computes PublB = g^b mod p:

b = 17

PublB = 131072

4. Bob and Alice exchange PublB and PublA

5. Alice computes (Shared Secret) SSA = PublB^a mod p:

SSA =
193025830561934107162947985381047541665608072055952185017491682078771915023
799273387871154500424503798663213600460826789274033295999330021731389427128
542432710187362934652673115221889249890533772697227171395058697282798274445
240687006095271729621464100656563293799180557568945517759802372156455525060
659659679134121984

6. Bob computes (Shared Secret) SSB = PublA^b mod p:

SSB =
193025830561934107162947985381047541665608072055952185017491682078771915023
799273387871154500424503798663213600460826789274033295999330021731389427128
542432710187362934652673115221889249890533772697227171395058697282798274445
240687006095271729621464100656563293799180557568945517759802372156455525060
659659679134121984

Conclusions:

1. The RSA algorithm can be used to encrypt a message using a publicly know key and decrypted only by the party that generated the public key using the private key. RSA encryption is commonly used in securing sensitive data transmitted over the internet, such as in HTTPS connections for secure communication on websites or in securing emails by providing a method for secure data transmission through public key encryption.

2. The ElGamal encryption algorithm operates by using a recipient's public key to encrypt a message and then requires the recipient's private key to decrypt the message, ensuring secure communication through asymmetric key cryptography. The ElGamal algorithm is utilized in various secure communication systems, including digital signatures, key exchanges, and secure messaging protocols, offering a method for secure encryption and decryption, particularly in scenarios where asymmetric cryptography is required, such as in secure messaging applications or protocols.

3. The Diffie-Helman algorithm can be used to securely create a shared private key without sharing any information that would allow a 3rd party to compute it. The Diffie-Hellman key exchange protocol is used to securely establish a shared secret key between two parties over an insecure channel, commonly employed in VPNs (Virtual Private Networks), secure messaging applications, and cryptographic protocols to enable secure communication and data exchange.

References:

1. Wolfram alpha (used for random num. generation): https://www.wolframalpha.com/

2. Powermod calculator: https://www.dcode.fr/modular-exponentiation

3. Inverse powermod calculator: https://www.dcode.fr/modular-inverse

4. Big number multiplication calculator: https://www.dcode.fr/big-numbers-multiplication

5. Github repo: https://github.com/muffindud/CS_Lab/tree/lab5/lab5