Ministry of Education, Culture and Research of the Republic of Moldova

Technical University of Moldova

Department of Software and Automation Engineering

REPORT

Laboratory Work Nr.5
Discipline: Cryptographic methods of information protection

Realised by: st.gr. FAF-213
Iațco Sorin

Checked by : asist.univ.
Mîțu Cătălin

Chișinău 2023

Subject: Public Key Cryptography

Tasks:

Task 2.1. Using the wolframalpha.com platform or the Wolfram app Mathematica, generate the keys and perform the encryption and decryption of the message m = Name First name applying the RSA algorithm.

The value of n must be at least 2048 bits.

Task 2.2. Using the wolframalpha.com platform or the Wolfram app Mathematica, generate the keys and perform the encryption and decryption of the message m = Name First name applying the ElGamal algorithm (p and generator are given below).

Task 3. Using the wolframalpha.com platform or the Wolfram app Mathematica, 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).

Note:

For tasks 2.1 and 2.2 use the decimal numerical representation of a the message, reaching it through the hexadecimal representation of the characters, in according to ASCII encoding. For convenience in conversion you can use the page https://www.rapidtables.com/convert/number/hex-to decimal.html.

For tasks 2.2 and 3 considered

p=32317006071311007300153513477825163362488057133489075174588434139269806834136210002792056362640164685458556357935330816928829023080573472625273554742461245741026202527916572972862706300325263428213145766931414223654220941111348629991657478268034230553086349050635557712219187890332729569696129743856241741236237225197346402691855797767976823014625397933058015226858730761197532436467475855460715043896844940366130497697812854295958659597567051283852132784468522925504568272879113720098931873959143374175837826000278034973198552060607533234122603254684088120031105907484281003994966956119696956248629032338072839127039,

which has 2048 bits and the generator g=2.

RSA Algorithm:

For the RSA algorithm firstly I generated 2 primes, p1 of 308 digits and another p2 of 309 digits using WolframAlpha. Next the n is computed by multiplying p1 and p2, next $\varphi(n)$ is computed by the formula. The e is chosen from a random interval from 1 to $\varphi(n)$ -1, the e value is checked if it's valid and is saved. Next d value is get by formula $e^{-1} mod\ \varphi(n)$.

The encryption process is straight forward, $m^n\ mod\ n$. The decryption as follows, $c^d\ mod\ n$.

RSA Results:

The python code made by me automates the process by generating random e, storing p1 and p2 as coonstants and doing the steps in their order.

m = Iatco Sorin

decimal m = 887118013679999105411587 66

n =

130547083559876264956548638012288752334615588780278943219794880454152206020
557143188986221651337486815378933425029952960725364317931305258045922694883
524623020499846269431063798896890911538551658446741505512589367251176861734
896951481105046841072700018388643791109297435582755204091995510018648904416
766291775849997963006083417025660171220290762231099508131232113363555040394
926556621075491487274529968721005918942062361719100293083406054787146341035
446392360315986605427492778217264729834022181651418359838524287545111452208
687426598358623219010183875101918349165494796589381948560971984634639998019
03584320692084739

n length in decimal: 617 ≡ 2049 bits

e =

119938521456905590594783945045310152922389682249306519862287460560520012605
871571949970854397481397348929744513490690801611645867684573869305209028668
445729955084649944018383455305337262289410495019191787799129387555767694408
791224757599533425724434740658181744103686636356544915454178349558101793674
280113996690325297424878812604263838784525688952449195792644498280808750153
945597932742602966335349222904655146123588463600472690262916858935123489655
749806745690250389394324926582397819835440334430787136599970354038282936487
607680791680147157418018038978517186034118446303878203521912355266212678065
01651535922712835

$\varphi(n)$ =

13054708355987626495654863801228875233461558878027894321979488045415220602055714318898622165133748681537893342502995296072536431793130525804592269488352462302049984626943106379889689091153855165844674150551258936725117686173489695148110504684107270001838864379110929743558275520409199551001864890441676629177292117514198385690468274513329650303466399961887772952672832664293878951706636030619535417465370527543546002417010526450052292375496875038102103701830538097903345123839007711407872719567677570435948593521450758399200232036390861262276094598661303206442524439337482755377098481132528760179213984334562578355624

d =

55483713251312355652023821211549310389574983014812295405608613470976206662371946748571419375489813474708680005281828890476782025481135053131018829119267229764754489352803428748069091482139570106232565808066613065073026559846105740825483976987022067246253542303049629856052155226384754769026973342521810293913301732600770414993800228244012834797894118748011780350776010477573020227487291886404398782783383811050640369048018244406916543494182997997289925811392352990525635893013759036599026116698860180094725146462519133735495127692045604167386184216564246628692520945910529020303538977266882041789373969908221607712991

encrypted m =

83802682353921651884452022389007317095654134614815963673370079428663229063515623164709804145335818038036379843759290225070969356041708891466633930896874692672360368896858779990268212993766887900630721853723393806114186016597984655342790539167445416467616663340285503782077726908909991435619980070819994547378889692684517347879601148248609277469867331464616427382973222854824691602571215131001305074122113091487372845670315851112478338744037427157774708334569730617348330360303097299195688458023145649827800135681172723257336328599670994781534638823515526495912164133686657288674277045364394521016954075673329186121581

ElGamal Algorithm:

In this approach I generated a random nr k depending on p, then generated the public key based on the generator, previous k and p. Then the beta generator which uses the previous

public key, generator and p. The encryption is made by creating 2 cyphers, then in the decryption process are used to decrypt using the fromula $r^t \bmod p$.

The message and the decimal equivalent remains the same from RSA, because the methods for generating that are the same.

ElGamal Results:

k =

1230275801781155985724718997128702217918335972345013843387846714419800227984056211746000030420669062012864801213258635876002196009905047551622126935722025981733552633748672119100587329602807870622104511905846749535244166130362022629628112022935425565655678427403443311628920017081269105174609002811843502248211594105429152925872642618747519050612400250252435203775882217073741817708891255627323515058818371347587616891198797644746936733593775338188910157263633191108168224678910921012631378985464249958771807833902334464002474874873406034802327470908525912161565229421928372237924064928069970268457902834949894905606881

public key =

2930092802035827480847444893601818246366341524263014154677956177490019408633703205946594780352280620886395217085396495601950703633569475138153218413729869144854280605129907891626753261866970470902719690307287076727215067262855357890155153861890732986795048404477656010535252723662009954796246341276435911194164458403887802314699820835608015540074933056993656058725673835888976081357679442226942776454122022035413562615593233251203509761640441072757844940324204765612749804326607875162501053260544786616727287963026162310269933401706189030580392033319586717657034564352303553453577012270487733994525782648236473710628 7

beta generator =

2252279666542395747398236044948577958568024507250062434215014287692298655363819750120019389046024753743210228957150930980394721394545273652621949674102413019924502396793549328179728228869063842161637864734633643517935143162566860654692692882404038401324498344295875518276955237020176675151796385039835921620594036432727656047384455084209256349324566852048712492756675711532718352381430664593076488214853690297293951674326462536345569506724862874972946355393418018803486898604422154540663726473947059282567849695454175223054318974992467432368199132769343176021589579999570394039671313063828972095729732709117703054235 7

r =

12923232013376114917970419257830761925261579218661024046318736432027807244059695744071902276743830584826867505760476543207531013981466471835894000438866888800905701177505331440571511948828135191284442729656238341510378914669847593748245409817433215788904338760368941097094408185179610672411697779937268368297885636673871373842005719696361517576777891867629623910311178822498734377054212573713943472970656924727563099174377865340607117802747558297522532902583028998024456412890954363653910530841068356222298518178870696216829717214887354189692119817075822125139916169438275749237343858027867474621074240532445847117 60

t =

53185925638677879541906294867311602293533100031980457916087456552922794085203107739001120452431338601983611678822514774088668934411498218915571324720639488249292796273498393608381453691347622623261863168855420291375672199185062458542520551457086201534957077087216408534389953240577733962989306543716735554057936691361534420530577274549826936628260458864875321956483848925771234501076569671526990655950553921632473148623012289824896040577300189800831205662613226636622733034478388207372840744475969147230321309447992027227870582752147685224479468292032933759627101469264263491723841371559232487250237255052836233047 53

Diffie-Helman Algorithm:

In this implemetation the random a and b are generated using the formula random.randint(2, p - 2). Public keys are generated as follows, pow(g, private1, p) and the shared secrets using pow(public2, private1, p). After all this generation the keys are prepared for AES to use and therefore are converted to 256 bits.

Diffie-Helman Results:

public key =

31192108220013081561998885229910113319392740217247345623728546863060401297470776424950532806284634237347544140004347932431689847948739188118041501520962424525318434308066846859544572825419559992835555053431671731229155745706790311755664987332618998436381222189822126996367844815468110752751717434675334256083757877289671306069650163831377443328659078364750566179495827104858853013383369129494255490429608403890849981176616797344938113126217223803848036353035352224512726171019174090670967306843826872925845144261176561787705 22

685171559144883169598570209718877186914558272650833586420385096928585377391
20709434198500 59

shared key =

450212466083097035206637388728097511100160290614074887330978777282671288518
92

Conclusion:

In conclusion, this laboratory work not only provided a hands-on experience with the implementation of RSA, ElGamal, and Diffie-Hellman but also fostered a deeper appreciation for the principles that underpin modern cryptographic protocols. The exploration of these widely used algorithms enhances my understanding of secure communication methods and prepares me to address the ongoing challenges and advancements in the field of cryptography.

Resources:

CS-Labs/lab5 at main · Syn4z/CS-Labs (github.com)