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

REPORT

Laboratory work No. 6

Course: Cryptography and Security

Theme: Hash Functions and Digital Signatures

Author: Corneliu Catlabuga
FAF-213

Checked by: univ. assist.
Cătălin Mîțu

Chisinau 2023

Objective:

Use RSA and ElGamal algorithms to check the digital signature of a text.

Task:

1. Hash a message, encrypt the hash with the private key and check it with the public key.

2. Hash a message, encrypt the hash with the shared secret generated through the ElGamal algorithm.

Theoretical considerations:

Text message: “each new setting of alberti's disk brought into play a new cipheralphabet, in which both the plaintext and the ciphertext equivalents arechanged in regard to one another. there are as many of these alphabetsas there are positions of his disk, and this multiplicity means that albertihere devised the first polyalphabetic cipher.this achievement—critical in the history of cryptology —alberti thenadorned by another remarkable invention: enciphered code. it was forthis that he had put numbers in the outer ring. in a table he permutedthe numbers 1 to 4 in two-, three-, and four-digit groups, from 11 to4444, and used these as 336 codegroups for a small code. "in this table,according to agreement, we shall enter in the various lines at thenumbers whatever complete phrases we please, for example,corresponding to 12, 'we have made ready the ships which we promisedand supplied them with troops and grain.' " these code values did notchange, any more than the mixed alphabet of the disk did. but the digitsresulting from an encoding were then enciphered with the disk just as ifthey were plaintext letters. in alberti's words, "these numbers i theninsert in my message according to the formula of the cipher, representingthem by the letters that denote these numbers." these numbers thuschanged their ciphertext equivalents as the disk turned. hence 341,perhaps meaning "pope," might become mrp at one position and fco atanother. this constitutes an excellent form of enciphered code, and justhow precocious alberti was may be seen by the fact that the majorpowers of the earth did not begin to encipher their code messages until400 years later, near the end of the 19th century, and even then theirsystems were much simpler than this.alberti's three remarkable firsts—the earliest western exposition ofcryptanalysis, the invention of polyalphabetie substitution, and theinvention of enciphered code—make him the father of westerncryptology. but although his treatise was published in italian in acollection of his works in 1568, and although his ideas were absorbed bypapal cryptologists and perhaps influenced the science's development,they never had the dynamic impact that such prodigiousaccomplishments ought to have produced. symonds' evaluation of hiswork in general may both explain why and summarize the modern view of his cryptological contributions:"this man of many-sided genius came into the world too soon for theperfect exercise of his singular faculties. whether we regard him from thepoint of view of art, of science, or of literature, he occupies in eachdepartment the position of precursor, pioneer, and indicator. alwaysoriginal and always fertile, he prophesied of lands he was not privilegedto enter, leaving the memory of dim and varied greatness rather than anysolid monument behind him."polyalphabeticity took another step forward in 1518, with theappearance of the first printed book on cryptology, written by one of themost famous intellectuals of his day. this was johannes trithemius, abenedictine monk whose dabbling in alchemy and other mystic powersmade him one of the most revered figures in occult science, while hismore solid scholarship won him the title of "father of bibiliography." in1518, a year and a half after his death, his polygraphiae libri sex, loannistrithemii abbatis peapolitani, quondam spanheimensis, ad maximilianumcaesarem ("six books of polygraphy, by johannes trithemius, abbot atwurzburg, formerly at spanheim, for the emperor maximilian") waspublished. by far the bulk of the volume consists of the columns ofwords printed in large gothic type that trithemius used in his systems ofcryptography. but in the work's book v appears, for the first time, thesquare table, or tableau. this is the elemental form of polyalphabeticsubstitution, for it exhibits all at once all the cipher alphabets in aparticular system. these are usually all the same sequence of letters, butshifted to different positions in relation to the plaintext alphabet, as inalberti's disk the inner alphabet assumed different positions in regard tothe outer alphabet. the tableau sets them out in orderly fashion—thealphabets of the successive positions laid out in rows one below theother, each alphabet shifted one place to the left of the one above. eachrow thus offers a different set of cipher substitutes to the letters of theplaintext alphabet at the top. since there can be only as many rows asthere are letters in the alphabet, the tableau is square.the simplest tableau is one that uses the normal alphabet in variouspositions as the cipher alphabets. each cipher alphabet produces, inother words, a caesar substitution."

For task 2:

p =
32317006071311007300153513477825163362488057133489075174588434139269806834136210002792056362640164685458556357935330816928829023080573472625273554742461245741026202527916572972862706300325263428213145766931414223654220941111348629991657478268034230553086349050635557712219187890332729569696129743856241741236237225197346402691855797767976823014625397933058015226858730761197532436467475855460715043896844940366130497697812854295958659597567051283852132784468522925504568272879113720098931873959143374175837826000278034973198552060607533234122603254684088120031105907484281003994966956119696956248629032338072839127039

g = 2

Implementation, practical results:

1. Generate p and q:

p =
24485854461595687005606407918654621839752348100902211815670164330924535738615644910919055367515249584982621060375135611802478681654175267676002687339329322513771725185761530765015131226975288847174603442712372258849477578232569602782690352971295189406320457432201408914545302875025322222258124509182949364772033929078805209130790620557446393481180624583009311686973865512518382691706854167122125412098257453602381305716888232962992821904034789734364973244552579152959488295715609139511416142330720028962082712701571869143968832613750274001675705397904390725235713583737611684332184102288547682027067142093734312924869

q =
310317574925389502612308909575219378962878811381964869032991785875769449437586840145927693138457296969884112799387535701852567680513623081466690134737995310649533442437206723922988765711716464500755769690670635579716444420716889031237597699490816880444314760266965365753412872933764911552653307803137
19819827

2. Compute n = p * q

n =
75983909764984024416965226730344521313348653603432399758573059656334006614979266362877724170246732662283167507336279327831498597272631940756508863912622535480033505345538725258754492216758905173449415263473603678051044088590198421446162322024867003726492259397420045243215903121799596900145031947342515557602592789401954599089847835353622576269567937999058507579074888671590947983199034061092378921061889498835357938896258410514984648323263762158145560501115523799639795493831163541455636874629898507669448529390008150318927966165255492774651142261630559870980602893627378333055070865732076453527288692731260538651031531555891883326512722502945452777938205702157290421780591821604300326860190566202846391105315334608779680011842063752089662133555718580645037604093593120352957355613067648461391725118023426064319951114835429684528715971030531502745583645556836363651509952442360801922822622096370634969270936421754006757766

3. Compute f = (p – 1) * (q – 1)

f =
7598390976498402441696522673034452131334865360343239975857305965633400661497926636287772417024673266228316750733627932783149859727263194075650886391262253548003350534553872525875449221675890517344941526347360367805104408859019842144616232202486700372649225939742004524321590312179959690014503194734251555760014420395579503038928719456175711408559270318896828639750787223849849440933746957000047338431036454033709583285874484933473678015784623539054529176718259154826262297525501046495412375193236962295198818511877092443398020834199853249638210696450104093034855715040723744160054057822954423130147621

418129656021738250354722621085184075373581050612360305144557425396043928753
524937043631951292810891745729641050162788427083121593650987718542663471033
763938733782909552519505131916171346672710202232278342585231041221477715364
143346682654483352122656371287714891223234954464659299798320700831381782176
5103768913434920348329 68

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

e = 65537

5. Generate $d = e^{-1} \bmod f$

d =
583702326490092727661033361267558009295594544404657547010903869325439913793
907436263344221333151007199390583663708284291267039516561736890212618635836
394944648410337156285323275706076056689668577644813073466679459157394241432
097826831719807489193784062476585648276878369425613419442560071074604176107
991782867580137655751835596481958315403803750425659100753525288391874490319
468550299053283133776726597998502108898745025483229283812685513033663539222
532923204841768357073666663640669868221049336275488276522449883962065799278
817916278761391329657312328136112180617015922664906446620292501396817808691
486347216571011981233631541854732323339227276089526215153314060478261616996
448005790036934675626964080424169227618682322375143150164929378672053839040
463333621500840462968925734567033072130820683451852436905159784095935053176
187448139802553738538766422211575235342383139028690242890984568768114436481
32222904000317540463 6553

6. Hash the message

SHA-384 of the message:
eda19a7f481563d901f6cdacda0648d04637181db7f466f9fb7f1782b98a49f9c7db74cb7a914e
b8b09a6e6edae91a2f

SHA-384 of the message decimal equivalent:
365747989233356570677807724547169274521594645995690861638741247180175411837
24512787529252580357012737363224472132143

7. Sign the message using the private key $Sign=h^d \bmod n$

Sign =
732742104862523608800898245158560239998430413787732335512298420469192221904
042416888334065823158906573241967823270860377324185337839279458962653035297
998774236465421897561432713380127998108366633611263607214920476989755975965
930146103889054091988458742961298912568058435837739865851360985136645113963
405410983288055052297316109624155473656064067563553029527986227643457395184
367253142327478053590943967431142277335215989563135289351507849346907112279
905364497429117617975138455611118314552394887169810545383019250896990467380
099304278892620200518418526423563506588250662293022493249254343273559044432
393707833999775038298560628206417672920213643264821933296312081520646810839
832515877352983653487430338400645225871255464065716967395413818511918817608
966825893595499964367113482207179867281610564875384777539448729120533601984

119808137329883922406178914669381798499307834576232208634535012084753218868
548272490658123510264509

8. Check the signature using the public key $h = Sign^e \mod n$

h =
365747989233356570677807724547169274521594645995690861638741247180175411837
24512787529252580357012737363224472132143

1.1. Party1 generates p(prime) , g(generator), PrivK1

p =
32317006071311007300153513477825163362488057133489075174588434139269806834136210002792056362640164685458556357935330816928829023080573472625273554742461245741026202527916572972862706300325263428213145766931414223654220941111348629991657478268034230553086349050635557712219187890332729569696129743856241741236237225197346402691855797767976823014625397933058015226858730761197532436467475855460715043896844940366130497697812854295958659597567051283852132784468522925504568272879113720098931873959143374175837826000278034973198552060607533234122603254684088120031105907484281003994966956119696956248629032338072839127039

g = 2

PrivK1 = 857

1.2. Party1 computes PubK1 = g^{PrivK1} mod p; hash the message using SHA-256

PubK1 =
960962154087001629436308185025184878247905227974336452694771122017616840015261658617268505418282416748614914455642299215255384405286687137505466996049907860485085046091635529899088428785810612776957410293491536147754283397719630991265870577566668501257551872

SHA-256 of the message:
29fa156225e639a107657503d5e62a8ffcdf623c786c52f3dcaeec2ca32ef5eb

SHA-256 of the message decimal equivalent:
18986686140847837898350195524353564974279753197334242309549510333764276778475

2. Party1 sends p, g, PubK1 ot Party2

3.1. Party2 generates PrivK2

PrivK2 = 553

3.2. Party2 computes PubK2 = g^{PrivK2} mod; shared secret s2 = $PubK1^{PrivK2}$ mod p

PubK2 =
29484081443918291814387145163970850710288447034503440846689111720668938768688662906922865040450459121417721679927842538279457692421287442441886205089317937841010900992

s2 =
990703868570605322415768597304519663557171412055020807739933742782170761215792589313205471832197102158624849480623470054046054059319826174139026821627076972317815566053341391410593022801172290120890907162496687406353708684438372834288315141633358445939940696497105483894931152410128302704373348086046333063893127240649972707640999003489734487982073098269163834079141538072629567712855989383120576792168051192470334373354571409180596024576980704128729

2330427720165368981809377841278540697970266281240341042615171642252846136570185974749221460243101742576287999231232496797501345784917598858052626582627337474593249904

4. Party2 sends PubK2 to Party1

5.1. Party1 computes the shared secret s1 = $PubK2^{PrivK1}$ mod p

s1 =
990703868570605322415768597304519663557171412055020807739933742782170761215792589313205471832197102158624849480623470054046054059319826174139026821627076972317815566053341391410593022801172290120890907162496687406353708684438372834288315141633358445939940696497105483894931152410128302704373348086046333063893127240649972707640999003489734487982073098269163834079141538072629567712855989383120576792168051192470334373354571409180596024576980704128729233042772016536898180937784127854069797026628124034104261517164225284613657018597474922146024310174257628799923123249679750134578491759885805262658262
7337474593249904

5.2 Party1 signs the message usign Sign = h * s1 mod p

Sign =
165803906474281430618385825271720877554300741597054347859230972692565440156244113586582071969516709033359245940183177193126049720872320660290230440004382750067211539759931966223474703385744469538783098391349577518158903808458418101166130528719918584601996115588250823410777439944581716576064364012409868374323787671257750045616764073512766078301382599810064180734648290258974573595857482516399153296214976072662528399289934254677329807012895044658753292880425045822196282871873515499183331711745484065431301681047774931240009388301024165239277055737237945126751252102773623066254872581856067433555575
9113252387512478

6. Party1 sends the signature to Party2

7. Party2 checks the signature using h = Sign * $s2^{-1}$ mod p

h =
1898668614084783789835019552435356497427975319733424230954951033376427677847
5

Conclusions:

1. The RSA algorithm can be used to sign a document with the private key, this way the signature cannot be forged and it can be checked with the public key.

2. The ElGamal algorithm for digital signatures works by generating a key pair consisting of a private key used for signing and a corresponding public key for verification. Signatures are created by combining the message with a random value and using modular arithmetic operations to produce a unique signature, while verification involves confirming the validity of the signature using the signer's public key and the original message.

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/main/lab6