**Part A**

1.(a)

p = 35219018721046519018661

q = 12532072192921

Thus, we have n = p\*q = 441367285175991202374244491191098781 and Φ(n) =441367285175955983355510912599887200.

We use the magma to calc:

> phin := 441367285175955983355510912599887200;

> Factorization(phin);

[ <2, 5>, <3, 3>, <5, 2>, <7, 2>, <11, 1>, <13, 1>, <17, 1>, <19, 1>, <23, 1>,

<29, 1>, <31, 1>, <883, 1>, <2633, 1>, <187807381691899, 1>]

We find the smallest ‘e’ to satisfy the gcd(Φ(n), e) =1 is 37.

Thus, RSA public key is 37.

Φ(n)= q1\*e + r1 = 11928845545296107658257051691888843\*37 + 9

e = q2\*r1+r2= 4\*9 + 1

r1= q3\*r2+r3 = 4\*1 + 0

By substitution, we have,

1 = [e-q2r1] mod Φ(n) = (37-4\*9) mod Φ(n)

= [37-4\*(Φ(n)- q1e)] mod Φ(n)

= [37+ (-4) \* (Φ(n) – q1\*37)] mod Φ(n)

= [37+ ((-4) \* Φ(n) + 4\* q1 \*37)] mod Φ(n)

= [(4\*q1+1) \*37 – 4 \* Φ(n)] mod Φ(n)

Thus, d = 4\*q1+1 = 47715382181184430633028206767555373. The corresponding private key for Alice is 47715382181184430633028206767555373.

(b)

I use the magma to help me calculate. I also refer some functions and operations from the magma documentation (Magma, 2017). The code is shown below:

// Two random primes

> p := RandomPrime(600);

> q := RandomPrime(600);

// Find n

> n := p \* q;

// Find Φ(n)

> PHin := (p-1) \* (q-1);

// Find the public key e (increase from 1)

> e := 1;

repeat

e := e + 1;

until (GCD(Phin, e) eq 1);

// Find the private key d

> d := InverseMod(e, PHin);

// A random message m

> m := RandomBits(600);

// A blinded message mb

> r := RandomBits(600);

> Xe := Modexp(r, e, n);

> Mb := (Xe \* m) mod n;

// Signature of m through blinded process

> ri := InverseMod(r, n);

> Mbd := Modexp(Mb, d, n);

> Sb := (Mbd \* ri) mod n;

// Direct signature of m using the private key

> s := Modexp(m, d, n);

> print "p:",p;

> print "q:",q;

> print "Phin:",PHin;

> print "e:",e;

> print "d:",d;

> print "m:",m;

> print "Mb:",Mb;

> print "Sb:",Sb;

> print "s:",s;

And the result is shown below, and the Sb and s of it is identical:

p: 2726716525715103608799979691197107670027509172136530724157890884342553947371\

1005813004167267005662629954891897691421172903716765533096148366431655434289821\

7063117101423598846253847055108883237901389837595970492095890320129459788964928\

5911140157102065126848319432964644035469822126723964634480421933766741851243623\

567539585618008764509482753917951570581221267733

q: 3272378562680604084227922867446466294470484257475660207268353735022279578626\

5300238163747219639492410107599939459807648929965378508152031857829139626794083\

0537061340283067958331842571354391058898932938198022211765304392877691712410000\

7299352684109525435556518555187625655143698772916404762485502203455457792401554\

207211322665948008878528749671365839997243051263

Phin: 8922848705257041172979359673395852435776794303120452638804565070088272193\

3869252366526749853322119627745715580376055469219565997969124465985173532917282\

0907501541919338977354444808094802471451106939435309521014391064804785099104498\

8457314100882027697246602146681299605882474833964338923840160230439010211598154\

3688218035586234372228910892636095235023569810837762880421856901942598144764829\

4768279688923361093497191066695839026584845943951328377267947855698453844597157\

7427689055045295183360216061418614894028759842079143123043885465764932956106336\

8052703662529902068284239801359961227668961275193387163208071190658773686694285\

8701974430284713895352474556450387534232484708884867393891709068392207076670005\

8979427502477784

e: 5

d: 1784569741051408234595871934679170487155358860624090527760913014017654438677\

3850473305349970664423925549143116075211093843913199593824893197034706583456418\

1500308383867795470888961618960494290221387887061904202878212960957019820899769\

1462820176405539449320429336259921176494966792867784768032046087802042319630873\

7643607117246874445782178527219047004713962167552576084371380388519628952965895\

3655937784672218699438213339167805316969188790265675453589571139690768919431548\

5537811009059036672043212283722978805751968415828624608777093152986591221267361\

0540732505980413656847960271992245533792255038677432641614238131754737338857174\

0394886056942779070494911290077506846496941776973478778341813678441415334001179\

5885500495557

m: 5284201530823946655285677212172031784158441625218500923013993881257530986146\

6427585208201855315127940813522521570309091250172015831749902083990162136584690\

5479519964761854530737338518348388080583461681521289732561551296715409726293745\

0366327434564551442228193623490042203529288506904478550825872507112948957334096\

0419481336329585125861358220525430361280070753784443370868395910369323303508969\

06495

Mb: 662434468797973523836734759544338159901748434363222990286139588916665222704\

8100839255667911791122787359817374527390217927739285261471866795839244714929352\

8824366409009963594483328354732888839130799170572380873757102407228564940410497\

3498928560363641729305727337555021943776278617517940833805860252498390025773384\

5959256762017188573092328822946038985815851170111542342929075555708865769947874\

0746537678579059879414126169442327980045513196198901704971266650886536395632465\

1678799440575674057685900923369426114013101369041473499599021285040720644186628\

4198929752209417475811400031629747794573145709115991760838437169654946664200869\

9841794541285912163494563350726039747253962772595648528182608356612864844473946\

26905581193762

Sb: 3975467745199825420550530998864938684481283645068171168480002480620114700386\

7174329526803175186782896726838798283329983144607832521728799870784992155320599\

6856984371322042495470787151466835238847586927747581848018328290323225226690684\

7802752069480903987632299235833972270058687728171415040935315884262381026589515\

0901418911236297629346100891376523761032164824122738971672765762285920468525711\

8294804653070294411188506887949525834786832933064329162513729305411842563117102\

2805107642927280624456601508640687301721309715674746795517316846921083534959143\

2147686900016496384704460647436412696194192205859930745156354854921739561434648\

7600895149788456154644027102365989838003605198884235803377575229883657728872195\

6219913329076

s: 3975467745199825420550530998864938684481283645068171168480002480620114700386\

7174329526803175186782896726838798283329983144607832521728799870784992155320599\

6856984371322042495470787151466835238847586927747581848018328290323225226690684\

7802752069480903987632299235833972270058687728171415040935315884262381026589515\

0901418911236297629346100891376523761032164824122738971672765762285920468525711\

8294804653070294411188506887949525834786832933064329162513729305411842563117102\

2805107642927280624456601508640687301721309715674746795517316846921083534959143\

2147686900016496384704460647436412696194192205859930745156354854921739561434648\

7600895149788456154644027102365989838003605198884235803377575229883657728872195\

6219913329076

(c)

This is not a secure method.

The known plaintext attack will be against this encryption method. The attacker could try to calculate all the represents of 26 alphabets by the RSA encryption algorithm Cm = (m)e mod n (0<=m<26). Then decrypt the ciphertext C = {c1,c2,c3…cn} by calculating the decryption algorithm D(ck)=i for all 0<=i<26, 1<=k<=n and ck could be obtained from the last step.

And the countermeasure is, modifying the encryption algorithm to Cm = (m)me mod n (0<=m<26) for each alphabet.

2.(a)

One-way property: The hash function does not satisfy the requirement. Supposed we have the h(M)=x where 0<=x<n. We could easily find the message M = {x,0,0,0…0} where 0<=x<n is a valid preimage.

Second image resistance: The hash function does not satisfy the requirement. Supposed we have the message M1 = {x,0,0,0…0} where 0<=x<n. Thus, we get the h(M1) =x. However, if we have the message M2 = {0,0,x,0…0} or {0,0,0…0,x}, we also get the h(M2) = x, but M1≠M2.

Collision resistance: Since the second image resistance is a kind of collision resistance and the second image resistance does not satisfy the requirement, this hash function does not satisfy this requirement.

(b)

One-way property: The hash function satisfies the requirement as it is difficult to calculate the m when we have h(m)=x because the hash function consists of the square root and modulo.

Second image resistance: The hash function does not satisfy the requirement. Supposed we have the message M1 = {x1, x2…xn} and we get the h(M1) =y. However, if we have the message M2 = {n-x1, n-x2…n-xn}, we also get the h(M2) = y, but M1≠M2.

Collision resistance: Since the second image resistance is a kind of collision resistance and the second image resistance does not satisfy the requirement, this hash function does not satisfy this requirement.

(c)

i. Message authentication codes

We can prevent the man-in-the-middle attack as MAC is a hash function. Both of Alice and Bob could authentic the opposite identity by using the promissory secret key to encrypt the message. Meanwhile, because of the confidentiality of the secret key, it is impossible for the attacker in the middle to intercept the message.

ii. Public key digital signature

Digital signature could also prevent the man-in-the-middle attack. Both of Alice and Bob could sign the public keys through their private keys. Then sending the message mutually. However, the attacker in the middle does not have the access to private keys. Therefore, he cannot intercept the message.

iii. Hash functions

Hash function does not provide the authentication function. Thus, it cannot secure DH protocol from man-in-the-middle attack.

3.(a)

In the First step, A sends message, identity IDA, master key Ka and encrypted identifier Na.

In the second step, B sends message, identify IDA and IDB, master key Ka and Kb and identifier Na and Nb to KDC to request a session key for protecting the connection with A.

After receiving the request from B, in the third step, KDC replies to B with two messages encrypted by Ka and Kb respectively. This two messages contains one-time session key Ks, identify IDA and IDB and identifier Na and Nb.

And in the last step, A receives the message encrypted by the master key which KDC shares with A and Ka, including with the identity of B IDB, identifier Na, and one-time session key Ks.

After these four steps, A and B begin to know each other through the identifiers and the message is originated from KDC, and are able to start their protected connection.

(b)

①The security level of these two scheme are high.

②The steps of the scheme given above in the figure 1 is 4, which means it has better efficiency.

③Their orders of the detection of replay attack are different. The attack in the scheme in figure 1 will be detected in the end. Correspondingly, the attack will be detected at the beginning.

(c)

The scheme is secured. ①Identifier Na and Nb are encrypted by the master key Ka and Kb, which ensures that the request is not modified by others before receiving by KDC. ②Through the scheme, A knows that it connects with B, and message is originated from KDC.

(d)

pros:

①High security level and efficiency.

cons:

①Once the KDC is threatened, the connection will be risky.

②It is possible that the KDC simulates as the sender or receiver.

(e)

Supposed there are n senders and responders who want to communicate with each other (n users and n\*(n-1)/2 communications). Since one pair of users need a session key, the number of session keys stored in the KDC is n\*(n-1)/2 and the number of master key is n. Thus, the memory requirement of KDC is n + n\*(n-1)/2.

And for each user, it is required to store one master key and (n-1) session keys. Hence, the memory requirement for each user is n.

**Reference:**

1. Magma, (2017). [online] Available at: <https://magma.maths.usyd.edu.au/magma/handbook/text/167> [Accessed 19 Sep. 2017].
2. Magma, (2017). [online] Available at: <https://magma.maths.usyd.edu.au/magma/handbook/text/32> [Accessed 19 Sep. 2017].