**CSC-321 Module 3 – Assignment Hints**

**Public Key Cryptography Implementation Requirements**

**Task 1: Implement Diffie-Hellman Key Exchange**

**Task 1, Part 1. Small Parameter Test Requirements**

- Implement Diffie-Hellman protocol with q=37 and g=5
- Generate random private keys XA and XB
- Compute public keys YA and YB
- Calculate shared secret ‘s’ for both parties
- Apply SHA256 to ‘s’ and truncate to 16 bytes to get key ‘k’
- Implement AES-CBC encryption with the derived key
- Verify Alice and Bob compute identical symmetric keys
- Exchange encrypted messages "Hi Bob!" and "Hi Alice!"

## Task 1, Part 1 Diagram:

```
Alice                                                  Bob
  |                                                     |
  |   Agree on parameters: q = 37, g = 5                |
  |<--------------------------------------------------->|
  |                                                     |
  |   Generate:                      Generate:          |
  |   XA = private_key               XB = private_key   |
  |   YA = g^XA mod q                YB = g^XB mod q    |
  |                                                     |
  |              Exchange public keys                   |
  |            YA -------------------------->           |
  |            <-------------------------- YB           |
  |                                                     |
  |   Compute:                       Compute:           |
  |   s = YB^XA mod q                s = YA^XB mod q    |
  |                                                     |
  |   Derive:                        Derive:            |
  |   k_alice = SHA256(s)[:16]       k_bob = SHA256(s)[:16]
  |                                                     |
  |   Create message:                                   |
  |   m = "Hi Bob!"                                     |
  |                                                     |
  |   Encrypt:                                          |
  |   iv_alice = random(16 bytes)                       |
  |   c0 = AES_CBC(k_alice, iv_alice, m)                |
  |                                                     |
  |          Send encrypted message                     |
  |        (iv_alice, c0) -------------------->         |
  |                                                     |
  |                                  Decrypt:           |
  |                                  m = AES_CBC_decrypt(k_bob, iv_alice, c0)
  |                                                     |
  |                                  Read: "Hi Bob!"    |
  |                                                     |
  |                                  Create message:    |
  |                                  m = "Hi Alice!"    |
  |                                                     |
  |                                  Encrypt:           |
  |                                  iv_bob = random(16 bytes)
  |                                  c1 = AES_CBC(k_bob, iv_bob, m)
  |                                                     |
  |          Receive encrypted message                  |
  |        <-------------------- (iv_bob, c1)           |
  |                                                     |
  |   Decrypt:                                          |
  |   m = AES_CBC_decrypt(k_alice, iv_bob, c1)          |
  |                                                     |
  |   Read: "Hi Alice!"                                 |
  |                                                     |
```

**Task 1, Part 1 Output:**

```
Diffie-Hellman Protocol (q=37, g=5)
--------------------------------------
Alice's private key (XA): 8
Alice's public key (YA): 16
Bob's private key (XB): 15
Bob's public key (YB): 29
Alice's computed shared secret: 10
Bob's computed shared secret: 10
Alice's derived key: 4a44dc15364204a80fe80e9039455cc1
Bob's derived key: 4a44dc15364204a80fe80e9039455cc1
Alice and Bob have the same key: True

Alice's message: Hi Bob!
Alice's IV: fce18d818434861a9307e7e0e7940c51
Alice's ciphertext: dc59b94600d658fa2846cdae8987d063
Bob's decrypted message: Hi Bob!

Bob's message: Hi Alice!
Bob's IV: 11df7bbfe2e83cb79000c51e8690c097
Bob's ciphertext: ace960d0793d66dbc72e0d716ea09b78
Alice's decrypted message: Hi Alice!
```

### Task 1, Part 2. Real-Life Parameters Requirements

- Modify implementation to use IETF-suggested 1024-bit parameters
- Handle the large values of ‘q’ and ‘g’ provided in the assignment
- Verify both parties can derive the same symmetric key
- Successfully exchange encrypted messages

### Task 1, Part 2 Diagram:

```
 Alice                                              Bob
   |                                                 |
   |  Agree on IETF 1024-bit parameters:             |
   |  q = B10B8F96A080E01D... (1024-bit prime)       |
   |  alpha = A4D1CBD5C3FD3412... (generator)        |
   |<----------------------------------------------->|
   |                                                 |
   |  Generate:                     Generate:        |
   |  XA = random(1, q-1)           XB = random(1, q-1)|
   |  YA = alpha^XA mod q           YB = alpha^XB mod q|
   |                                                 |
   |            Exchange public keys                 |
   |          YA ------------------------>           |
   |          <-------------------------- YB         |
   |                                                 |
   |  Compute:                      Compute:         |
   |  s = YB^XA mod q               s = YA^XB mod q  |
   |                                                 |
   |  Derive:                       Derive:          |
   |  k_alice = SHA256(s)[:16]      k_bob = SHA256(s)[:16]
   |                                                 |
   |  Generate shared IV:                            |
   |  iv = os.urandom(16)                            |
   |                                                 |
   |  Create message:                                |
   |  m0 = "Hi Bob!"                                 |
   |                                                 |
   |  Encrypt:                                       |
   |  c0 = AES_CBC(k_alice, iv, m0)                  |
   |                                                 |
   |        Send encrypted message                   |
   |      (iv, c0) ------------------------->        |
   |                                                 |
   |                                Decrypt:         |
   |                                m0 = AES_CBC_decrypt(k_bob, iv, c0)
   |                                                 |
   |                                Read: "Hi Bob!"  |
   |                                                 |
   |                                Create message:  |
   |                                m1 = "Hi Alice!" |
   |                                                 |
   |                                Encrypt:         |
   |                                c1 = AES_CBC(k_bob, iv, m1)
   |                                                 |
   |        Receive encrypted message                |
   |      <------------------------ (iv, c1)         |
   |                                                 |
   |  Decrypt:                                       |
   |  m1 = AES_CBC_decrypt(k_alice, iv, c1)          |
   |                                                 |
   |  Read: "Hi Alice!"                              |
   |                                                 |
```

**Task 1, Part 2 Output**:

Diffie-Hellman Protocol (IETF 1024-bit parameters)

--------------------------------------

Alice's private key (XA):
61337061524162160389099585484767944908502059393706460989725775719741333903865841494692172357810265507232335786386061805772092325421206099550873746836534854791220211200202521273607874723574542373175598826989040814463543735220951597859534957950651557150053732769841807874382066522980828356322811531146902331037

Alice's public key (YA):
96954913999705214115993500408089480683892900266164899968904446318303458380674530350005102926085361016324479393573629409110826911490477967509393300833630768067075103582005335894717057709689299978911858951286452385439424881431580696255214448978522597797793619365064223946613301275992572684594355199363780040744

Bob's private key (XB):
95761121201629030444071883330481174255771204637765106793567459337375886550040189716268650569931684116935515543851570449286973694285570449224821431250362462407683542075502878861830039265879419593065101150561924759935059286563779517777161863487169681661333722887116748510024921971314791967793964941693709997803

Bob's public key (YB):
89082303554651127986197428173068482832548557063687875057832443084319108227121427643350365367887613961847043062075999650521889617469828780126271964163736200687052782657610713164480197245075937229157416507587554202727126971348439630068073209250100075479511395504115744335890770986527369310796957435934011307414

Alice's computed shared secret:
10661327871683693611019330687378504146365547484940861374704420956596958483103693734789706055050067778333366483258013131311797867739083983659772898234745270794450049717798062048250193877382420932602623380238388005940797872097519301535300800547605984357810278671212472294306416332430865727197831073125603751761

Bob's computed shared secret:
10661327871683693611019330687378504146365547484940861374704420956596958483103693734789706055050067778333366483258013131311797867739083983659772898234745270794450049717798062048250193877382420932602623380238388005940797872097519301535300800547605984357810278671212472294306416332430865727197831073125603751761

Alice's derived key: 9f52a3fba869507873a0afd3789e1481

Bob's derived key: 9f52a3fba869507873a0afd3789e1481

Alice and Bob have the same key: True

Alice's message: Hi Bob!

Alice's IV: 1258c7ec8f6cc56ae467be4818dc2321

Alice's ciphertext: 4fe25116b3fabc29d5a9f302bd057ff5

Bob's decrypted message: Hi Bob!

Bob's message: Hi Alice!

Bob's IV: 1258c7ec8f6cc56ae467be4818dc2321

Bob's ciphertext: b5ca77b71f8af1f712c7cd0b18e6352c

Alice's decrypted message: Hi Alice!

**Task 2: Implement MITM Key Fixing & Negotiated Groups**

**Task 2, Part 1: MITM Attack with Parameter Substitution Requirements**

- Modify the original implementation to simulate a MITM attack
- Implement Mallory as the attacker who replaces ‘YA’ and ‘YB’ with ‘q’
- Demonstrate how Mallory can determine the shared secret s
- Show Mallory can decrypt messages ‘c0’ and ‘c1’

**Task 2, Part 1Diagram:**

```
Alice                Mallory (MITM)                Bob
  |                        |                         |
  |   Agree on IETF 1024-bit parameters:             |
  |   q = B10B8F96A080E01D... (1024-bit prime)       |
  |   alpha = A4D1CBD5C3FD3412... (generator)        |
  |<--------------------|---------------------------|->|
  |                     |                           |
  |   Generate:         |                Generate:  |
  |   XA = random(1, q-1)|               XB = random(1, q-1)
  |   YA = alpha^XA mod q|               YB = alpha^XB mod q
  |                     |                           |
  |        Exchange public keys (intercepted)       |
  |        YA --------->|                           |
  |                     |<---------------------- YB |
  |                     |                           |
  |                     | Modify public keys:       |
  |                     | YA_modified = q           |
  |                     | YB_modified = q           |
  |                     |                           |
  |     Modified keys forwarded                     |
  |<-- YB_modified = q  |                           |
  |                     | YA_modified = q --------->|
  |                     |                           |
  |   Compute:          |                Compute:   |
  |   s = q^XA mod q = 0|                s = q^XB mod q = 0
  |                     |                           |
  |   Derive:           | Derive:        Derive:    |
  |   k_alice = SHA256(0)[:16] | k_mallory = SHA256(0)[:16] | k_bob = SHA256(0)[:16]
  |                     |                           |
  |   Generate shared IV:                           |
  |   iv = os.urandom(16)                           |
  |                     |                           |
  |   Create message:   |                           |
  |   m0 = "Hi Bob!"    |                           |
  |                     |                           |
  |   Encrypt:          |                           |
  |   c0 = AES_CBC(k_alice, iv, m0)                 |
  |                     |                           |
  |      Send encrypted message (intercepted)       |
  |        (iv, c0) --->|                           |
  |                     | Decrypt:                  |
  |                     | m0 = AES_CBC_decrypt(k_mallory, iv, c0)
  |                     |                           |
  |                     | Read: "Hi Bob!"           |
  |                     |                           |
  |                     | Forward: (iv, c0) ------->|
  |                     |                           |
  |                     |                Decrypt:   |
  |                     |                m0 = AES_CBC_decrypt(k_bob, iv, c0)
```

```
|                    |                    |
|                    |                 Read: "Hi Bob!"
|                    |                    |
|                    |                 Create message:
|                    |                 m1 = "Hi Alice!"
|                    |                    |
|                    |                 Encrypt:  |
|                    |                 c1 = AES_CBC(k_bob, iv, m1)
|                    |                          |
|     Receive encrypted message (intercepted)   |
|                    |<--------- (iv, c1)       |
|                    |                          |
|                    | Decrypt:                 |
|                    | m1 = AES_CBC_decrypt(k_mallory, iv, c1)
|                    |                          |
|                    | Read: "Hi Alice!"        |
|                    |                          |
|    <-- (iv, c1) ----|                         |
|                    |                          |
|   Decrypt:         |                          |
|   m1 = AES_CBC_decrypt(k_alice, iv, c1)       |
|                    |                          |
|   Read: "Hi Alice!" |                         |
|                    |                          |
```

### Task 2, Part 1 Output:

MITM Key Fixing Attack (IETF 1024-bit parameters)

-------------------------------------

Alice's private key (XA):
7594037035822406806222706448509968401426209013786733764040920363952030610383906525611396124393300287459745763597643819223171784598092696861587198160925219064418737140124626871568200391966853869328309399119542347735944668933282030735310282662327918398142964139533014327687302988102060585270929371802514528302 8

Alice's public key (YA):
2671962853469389506813664118204693468236799679135957551293870230367438080329177848938698328548154139005810800954008331441149484334423486877402136789736875692683990414903198082794560336703191393230807608525641102959178322636822615939022470404095690003632903777249028789968786660082156959380084303964335750889 9

Bob's private key (XB):
1589389875033061001865027605347417159729392567430123935280624400164116383768015962070908140320642098960124540353142364667995653967235082337634947046061954416855510142060735022645576658750838618652109381293315282979686879476347901205592833654802566255460906465080390628381810325828887830929832027735463846629 2

Bob's public key (YB):
3315142639879087171766027878996381798719782210351497498415516794643229562080526469702460849808549477802479877802888680408148374354378476465003508631082110204189801195626247545375879190457600032709883563910662696012919378761850517287537101236111061851711996436464837388318619563595593166104460924385106404264 3

Mallory intercepts and modifies the public keys:

Modified YA (sent to Bob):
124325339146889384540494091085456630009856882741872806181731279018491820800119460022367403769795008250021191767583423221479185609066059226301250167164084041279837566626881119772675984258163062926954046545485368458404445166682380071370274810671501916789361956272226105723317679562001235501455748016154805420913

Modified YB (sent to Alice):
124325339146889384540494091085456630009856882741872806181731279018491820800119460022367403769795008250021191767583423221479185609066059226301250167164084041279837566626881119772675984258163062926954046545485368458404445166682380071370274810671501916789361956272226105723317679562001235501455748016154805420913

Alice's computed shared secret: 0

Bob's computed shared secret: 0

Alice's derived key: 5feceb66ffc86f38d952786c6d696c79

Bob's derived key: 5feceb66ffc86f38d952786c6d696c79

Mallory's derived key: 5feceb66ffc86f38d952786c6d696c79

Mallory determines the shared secret (s): 0

All parties have the same key: True

Alice's message: Hi Bob!

Alice's IV: d2501585b027b81fbbdfdcf9a8c1836d

Alice's ciphertext (c0): 58ee69b8634f46863eb17109677dca15

Mallory decrypts c0: Hi Bob!

Bob's message: Hi Alice!

Bob's IV: d2501585b027b81fbbdfdcf9a8c1836d

Bob's ciphertext (c1): 9b4215c71b8fd9942d398e22d9c153db

Mallory decrypts c1: Hi Alice!

**Task 2, Part 2: Generator Tampering Requirements**

- Implement another MITM attack where Mallory tampers with generator ‘g’
- Test setting ‘g’ to ‘1’,  ‘q’, and ‘q-1’
- Demonstrate how Mallory can recover ‘m0’ and ‘m1’ from ciphertexts

Diagram:

```
Alice                  Mallory (MITM)               Bob
  |                          |                        |
  |   Agree on IETF 1024-bit parameters:              |
  |   q = B10B8F96A080E01D... (1024-bit prime)        |
  |<---------------------|--------------------------|->|
  |                      |                          |
  |                      | Tamper with generator:   |
  |                      | alpha = 1 or q or q-1    |
  |                      |                          |
  |   Generate:          |                Generate: |
  |   XA = random(1, q-1)|                XB = random(1, q-1)
  |   YA = alpha^XA mod q|                YB = alpha^XB mod q
  |                      |                          |
  |        Exchange public keys                     |
  |        YA --------->|----------------------> YA |
  |        YB <---------|<---------------------- YB |
  |                      |                          |
  |   Compute:           |                Compute:  |
  |   s = YB^XA mod q    |                s = YA^XB mod q
  |                      |                          |
  |   Derive:            |                Derive:   |
  |   k_alice = SHA256(s)[:16]     k_bob = SHA256(s)[:16]
  |                      |                          |
  |                      | Determine shared secret: |
  |                      | If alpha = 1: s = 1      |
  |                      | If alpha = q: s = 0      |
  |                      | If alpha = q-1: s = 1 or q-1
  |                      |                          |
  |                      | Derive:                  |
  |                      | k_mallory = SHA256(s)[:16]|
  |                      |                          |
  |   Generate shared IV:                           |
  |   iv = os.urandom(16)                           |
  |                      |                          |
  |   Create message:    |                          |
  |   m0 = "Hi Bob!"     |                          |
  |                      |                          |
  |   Encrypt:           |                          |
  |   c0 = AES_CBC(k_alice, iv, m0)                 |
  |                      |                          |
```

```
|      Send encrypted message (intercepted)        |
|        (iv, c0) --->|-----------------------> (iv, c0)
|                     |                            |
|                     | Decrypt:                   |
|                     | m0 = AES_CBC_decrypt(k_mallory, iv, c0)
|                     |                            |
|                     |                Decrypt:    |
|                     |                m0 = AES_CBC_decrypt(k_bob, iv, c0)
|                     |                            |
|                     |                Create message:
|                     |                m1 = "Hi Alice!"
|                     |                            |
|                     |                Encrypt:    |
|                     |                c1 = AES_CBC(k_bob, iv, m1)
|                     |                            |
|      Receive encrypted message (intercepted)     |
|    (iv, c1) <-------|<---------- (iv, c1)        |
|                     |                            |
|                     | Decrypt:                   |
|                     | m1 = AES_CBC_decrypt(k_mallory, iv, c1)
|                     |                            |
|   Decrypt:          |                            |
|   m1 = AES_CBC_decrypt(k_alice, iv, c1)          |
|                     |                            |
| For alpha = q-1, Mallory may need to try both s = 1 and s = q-1
```

### Task 2, Part 2 Output:

MITM Generator Attack (alpha = 1)

--------------------------------------

Mallory tampers with the generator: alpha = 1

Alice's private key (XA):
4804496315351516264585102307781479940544890822492399761191956639887168062400580710779196523229229763006574283115055266237046022388288172337538730904461731531669657018746303886833731785993284714759279049973641414823562096934917706399962382566398773041303315893060531204325514823326512180214263222389196584480

Alice's public key (YA): 1

Bob's private key (XB):
7521072966625250854259829444959247927806094514176506719176326096046099639835773145272945556450648562296851323775689270884558211458831592939917027315377157530956412806223833400056618157301730641882059997771478038257884046106083930929608468610428329516800497550316263600354203572423347621122850053379591904002

Bob's public key (YB): 1

Alice's computed shared secret: 1

Bob's computed shared secret: 1

Alice's derived key: 6b86b273ff34fce19d6b804eff5a3f57

Bob's derived key: 6b86b273ff34fce19d6b804eff5a3f57

Case: alpha = 1

Mallory knows the shared secret will always be 1

Mallory determines the possible shared secret(s): 1

Mallory's derived key: 6b86b273ff34fce19d6b804eff5a3f57

Alice's message (m0): Hi Bob!

Alice's ciphertext (c0): df92d4327d7bf4242f86c95d7a743a84

Bob's message (m1): Hi Alice!

Bob's ciphertext (c1): 59aacd9a1b2722c5cf38719944cc243f

Mallory successfully decrypts c0: Hi Bob!

Mallory successfully decrypts c1: Hi Alice!

MITM Generator Attack (alpha = 124325339146889384540494091085456630009856882741872806181731279018491820800119460022367403769795008250021191767583423221479185609066059226301250167164084041279837566626881119772675984258163062926954046545485368458404445166682380071370274810671501916789361956272226105723317679562001235501455748016154805420913)

--------------------------------------

Mallory tampers with the generator: alpha = 124325339146889384540494091085456630009856882741872806181731279018491820800119460022367403769795008250021191767583423221479185609066059226301250167164084041279837566626881119772675984258163062926954046545485368458404445166682380071370274810671501916789361956272226105723317679562001235501455748016154805420913

Alice's private key (XA):
3478232994266112863226409378550382061593547490735998702186935775457081719035856729165458277102508867381496139876375241134371160004836135835817104331215631284593141523659870528434130730497511582793560014717787905320357722283780863410551607621805359619537209959211647311387730567711330494458491890944900831271 8

Alice's public key (YA): 0

Bob's private key (XB):
898463005036306055539169696893930826002470953141209866981761057530925966337041705650237662959904436796093284377307419628752469412199488996376133107225252501182453660114734814338537150517207807647645061370171085116062237841233592999508738556434663759727671447764915936069137386068737557141165597578651766084 2

Bob's public key (YB): 0

Alice's computed shared secret: 0

Bob's computed shared secret: 0

Alice's derived key: 5feceb66ffc86f38d952786c6d696c79

Bob's derived key: 5feceb66ffc86f38d952786c6d696c79

Case: alpha = q

Mallory knows the shared secret will always be 0

Mallory determines the possible shared secret(s): 0

Mallory's derived key: 5feceb66ffc86f38d952786c6d696c79

Alice's message (m0): Hi Bob!

Alice's ciphertext (c0): 3b19ced240aa945a5791380fa7af8e84

Bob's message (m1): Hi Alice!

Bob's ciphertext (c1): 20b66b1a63f295b815a54e0dc4ea474a

Mallory successfully decrypts c0: Hi Bob!

Mallory successfully decrypts c1: Hi Alice!

MITM Generator Attack (alpha =
124325339146889384540494091085456630009856882741872806181731279018491820800119460022367
403769795008250021191767583423221479185609066059226301250167164084041279837566626881119
772675984258163062926954046545485368458404445166682380071370274810671501916789361956272
22610572331767956200123550145574801615480542 0912)

--------------------------------------

Mallory tampers with the generator: alpha =
124325339146889384540494091085456630009856882741872806181731279018491820800119460022367
403769795008250021191767583423221479185609066059226301250167164084041279837566626881119
772675984258163062926954046545485368458404445166682380071370274810671501916789361956272
226105723317679562001235501455748016154805420912

Alice's private key (XA):
103054195007019051272536774705593352108466419067810687511019094412247323718536370939550
543056182422948324954964571768683716759518247711668665609274417680818232952489778776787
185919997049281152535119591333284663415826297675486197366147334609884236380349174528036
549308891152511559142264277788463076914446454691

Alice's public key (YA):
124325339146889384540494091085456630009856882741872806181731279018491820800119460022367
403769795008250021191767583423221479185609066059226301250167164084041279837566626881119
772675984258163062926954046545485368458404445166682380071370274810671501916789361956272
226105723317679562001235501455748016154805420912

Bob's private key (XB):
342022085383535748945557228561146746101367792040843618602533334750907624520103704954740
954713781249442657091164276781346920951192683557114337076183894935247379066389791111963
231509434953849922776159815699544170403040009923219349641701285226825847914459732112522
568261459080679405781474355502937365200028 78064

Bob's public key (YB): 1

Alice's computed shared secret: 1

Bob's computed shared secret: 1

Alice's derived key: 6b86b273ff34fce19d6b804eff5a3f57

Bob's derived key: 6b86b273ff34fce19d6b804eff5a3f57

Case: alpha = q - 1

Mallory knows the shared secret will be either 1 or q-1

Mallory determines the possible shared secret(s): 1

Mallory's derived key: 6b86b273ff34fce19d6b804eff5a3f57

Alice's message (m0): Hi Bob!

Alice's ciphertext (c0): 5197330882501ad75a592bf1a18e8531

Bob's message (m1): Hi Alice!

Bob's ciphertext (c1): 2bb4ec0f1cd40c62234d077139370971

Mallory successfully decrypts c0: Hi Bob!

Mallory successfully decrypts c1: Hi Alice!

**Task 3: Implement "Textbook" RSA & MITM Key Fixing via Malleability**

**Task 3 Part 1: RSA Requirements**

- Implement RSA key generation supporting variable-length primes up to 2048 bits
- Use ‘e=65537’ as the public exponent
- Implement the calculation of the multiplicative inverse (for private key)
- Convert messages to integers in Z*n (less than n)
- Test encryption and decryption of messages

**Task 3 Part 1: Diagram**:

```
Key Generation                 Encryption                 Decryption
--------------                 ----------                 ----------

Generate p, q (primes)         Convert message to int
        |                      m = string_to_int(msg)
        v                               |
    n = p * q                           v
        |                      c = m^e mod n     c -----> c
        v                               |                 |
phi = (p-1) * (q-1)                     |                 v
        |                               |           m = c^d mod n
        v                               |                 |
    e = 65537                           |                 v
        |                               |        Decode m to string
        v                               |
d = mod_inverse(e, phi)                 |
        |                               |
        v                               |
Public key: (n, e)                      |
Private key: (n, d)                     |
        |                               |
        |                               |
        |              Encryption       |      Decryption
        |              ----------       |      ----------
        |                               |
        |              m <--- message   |      message <--- m
        |                   |           |         ^
        |                   |           |         |
        v                   v           v         |
     (n, e) -------------> m^e mod n = c ------> c^d mod n

Note: Ensure m < n (message must be in Zn*)
```

**Task 3 Part 1 Output**:

Testing RSA Implementation

-------------------------

Generated RSA key pair. n has 2048 bits.

Original message: Hello, World!

Encrypted (integer):
457954381165732758457571415526181644266169999268952136998238760493637727020639531972829
756267427491452763579854756553444716819629991216825198623113960524063048854008208306985
360120113304504126843642606641357933249467396334571238449314651631250945914131540673423
556064075820616381597952050200269216904101670543153351887164269917937055321427158865213
235815971953343999971763311468011478158248846090165526032589028615472057864912913782124
661325221859926681822784844056131201604717981340579664802176397383794166278109219599852
695345477177660112601026816762465427246412660214816394727256291384274382531594314249725
4673035

Decrypted: Hello, World!

Original message: RSA Encryption

Encrypted (integer):
906942585563223933032559921736933311420201010938925885712857328922732523273131129977453
138900360793435427234176051609352995865280246160820684633068672149985607091580997169966
970209196707129542168771336576091308414004426720652349966074871744193524995139919539734
132564275205821340144773332497607810649389313190269819812709382396929341374281120967493
999408978317097818605261698859768525105463065138558696552583659613605958430926004974642
999296046377370100603222873222152534396410562742519209409747866719280945272007688142425
611782771351946614652739893183561551888008797091149711427453589677506328903715292557173
8527974

Decrypted: RSA Encryption

Original message: Cryptography is fun!

Encrypted (integer):
108261045906402511331340368192134749951935955510268158510179941035400100693733230229755
663453373781848736655793476291877754274418051977030154601449526809386301019655560509386
624056289667727555739923574858413883862835423925945648814308899821669300171691724916716
633578924882291200521853603966644441257757675985591557924546108134515585387365883672802
620074214283342732497945986783090222960772160686084649140605081075002993619583024411256
954671852683872286043852166554587360984782256685808888140692237697259636836424394996617
719644091288995749362832444343251994694430932479823569056376971522673105755450760647490
72265510

Decrypted: Cryptography is fun!

Original message: Test message for RSA

Encrypted (integer):
1702850291357382889220908254375477122647472977808923152202814634038193425362487734281459801050353579603788049156978407894521820272760068757821938046198516377684362669503971037927241697810782843601947795838678102067993399952083031374400523093727505545411059542850957480379347423082106685183599945220213558522968264983127912972373025627295457535676643410264147758563010879779457722184646433262496109614510866790615704413864553660443487183591173784059139914394989525740348009153220931840398369785385671186197454740005537261807185025654707940373144310312113118868680356977513362218170074119402584830522815691898221654464

Decrypted: Test message for RSA

All valid messages were successfully encrypted and decrypted.

**Task 3 Part 2a: Malleability Key Exchange Attack Requirements**

- Alice encrypts a symmetric key s as c = s^e mod n
- Mallory modifies c → c' = c * (r^e mod n) (e.g., r = 2)
- Bob decrypts c', gets s' = s * r
- Mallory recovers s = s' * $r^{-1}$ mod n
- Use k = SHA256(s), decrypt AES-CBC encrypted m

**Task 3 Part 2a Diagram**:

```
Alice                         Mallory (Attacker)               Bob
  |                                 |                            |
  |   Generate RSA key pair         |                            |
  |   (n, e) - public key           |                            |
  |   (n, d) - private key          |                            |
  |                                 |                            |
  |   Generate symmetric key s      |                            |
  |   c = s^e mod n                 |                            |
  |                                 |                            |
  |        Send c --------------->|                            |
  |                                 |                            |
  |                                 | Intercept c                |
  |                                 |                            |
  |                                 | Attack Method 1:           |
  |                                 | Choose factor (e.g., 2)    |
  |                                 | c' = (c * 2^e mod n) mod n |
  |                                 |                            |
  |                                 | Attack Method 2:           |
  |                                 | Choose s' (e.g., 2s)       |
  |                                 | c'_alt = (s')^e mod n      |
  |                                 |                            |
  |                                 |    Send c' or c'_alt ----->|
  |                                 |                            |
  |                                 |                     Decrypt
  |                                 |                     s' = (c')^d mod n
  |                                 |                     (s' ≈ 2s mod n)
  |                                 |                            |
  |                                 | Deduce s from s'           |
  |                                 | (s ≈ s'/2 mod n)           |
  |                                 |                            |
  |   Use s to encrypt message m    |                            |
  |   c0 = AES_encrypt(m, s)        |                            |
```

```
  |        Send c0 -------------->|                            |
  |                               |                            |
  |                               | Decrypt c0 using s         |
  |                               | m = AES_decrypt(c0, s)     |
  |                               |                            |
  |                               | Mallory reads m            |

Key Points:
1. RSA is used to encrypt the symmetric key s
2. The attack exploits RSA's multiplicative property
3. Mallory modifies c without knowing s or d
4. Two methods shown for creating c':
   - Method 1: Multiply c by an encrypted factor
   - Method 2: Directly encrypt a chosen multiple of s
5. Bob unknowingly decrypts to a multiple of s
6. Mallory can deduce s from the modified value
7. This allows Mallory to decrypt subsequent AES messages

Security Implication:
This demonstrates why proper padding (e.g., OAEP) is crucial in RSA encryption.
```

**Task 3 Part 2a Output**:

Demonstrating RSA Encryption Malleability

----------------------------------------

Alice's original symmetric key (s): 2481247348473661360911194324834197018

Encrypted symmetric key (c):
1662381029442961653867631908925598860091307473564551287337217435178932982
8262462889661655750895331959387934850984792550926530660682106792717794992
8413773430908128725056704099068177227253710814923034126936175083879204814
3502525918242910241768885735762629757407771518827163932823951594430141832
7999423751721776

Mallory's modified ciphertext (c'):
3603093875214711715426922928995354149244037828941797861764731989761856174
6161473117708413155769940973353895085206908313495334746929290269282380376
6353815532703282054741587798904074822835730168128353149591538603033326561
8609124848966883994379072487690807895930779556302120098157402755220439472
9195543630569206

Alternative malleability attack approach:

Mallory's chosen s': 4962494696947322721822388649668394036

Mallory's computed c':
3603093875214711715426922928995354149244037828941797861764731989761856174
6161473117708413155769940973353895085206908313495334746929290269282380376
6353815532703282054741587798904074822835730168128353149591538603033326561
8609124848966883994379072487690807895930779556302120098157402755220439472
9195543630569206

Value Alice decrypts to: 4962494696947322721822388649668394036

Alternative attack successful: Alice decrypted to Mallory's chosen value!

Bob's decrypted value (s'): 4962494696947322721822388649668394036

Mallory's recovered symmetric key: 2481247348473661360911194324834197018

Attack successful: Mallory recovered the original symmetric key!

Bob's encrypted message (c0):
7035c1e859eb7cb5ea597f67d7173c2bf79b830eb3d1667ccdccbc8839ec5f0544f25a2bb8
576970d0dd75d6aa8777b1

Mallory's decrypted message: Secret message from Bob to Alice

Another example of RSA malleability exploitation:

In a digital signature system, an attacker could modify

a signed message to create a valid signature for a different

message, violating the integrity of the system.

**Task 3 Part 2b: RSA Signature Malleability Requirements**

1. Implement RSA key generation:
   - Generate two large primes p and q
   - Compute modulus n = p * q
   - Compute Euler's totient ϕ(n) = (p-1)*(q-1)
   - Use public exponent e = 65537
   - Compute private key $d = e^{-1}$ mod ϕ(n) using the Extended Euclidean Algorithm
2. Implement RSA signing function:
   - Input: message m (as integer), private key (n, d)
   - Output: signature s = m^d mod n
3. Implement RSA verification function:
   - Input: message m, signature s, public key (n, e)
   - Output: Boolean indicating whether s^e mod n == m
4. Sign two messages:
   - Choose two distinct integer messages m1 and m2 such that both are in $\mathbb{Z}_n^*$
   - Compute their signatures: s1 = m1^d mod n, s2 = m2^d mod n
5. Mallory creates a forged signature:
   - Compute m3 = m1 * m2 mod n
   - Compute s3 = s1 * s2 mod n
   - This works because: (s1 * s2)^e mod n == (m1 * m2) mod n
6. Verify the forged signature:
   - Confirm that verify(m3, s3, public_key) returns True
7. Explain why this demonstrates RSA signature malleability:
   - Show that a valid signature for m3 = m1 * m2 can be created without signing m3 directly

- Conclude that textbook RSA signatures are multiplicatively malleable

**Task 3 Part 2b Diagram:**

```
Alice                           Mallory (Attacker)                Verifier
  |                                    |                              |
  |   Generate RSA key pair            |                              |
  |   (n, e) - public key              |                              |
  |   (n, d) - private key             |                              |
  |                                    |                              |
  |   Create messages:                 |                              |
  |   m1 = 12345                       |                              |
  |   m2 = 67890                       |                              |
  |                                    |                              |
  |   Sign messages:                   |                              |
  |   s1 = m1^d mod n                  |                              |
  |   s2 = m2^d mod n                  |                              |
  |                                    |                              |
  |        Publish (m1, s1) ------>|                              |
  |        Publish (m2, s2) ------>|                              |
  |                                    |                              |
  |                                    | Intercept (m1, s1) and (m2, s2)
  |                                    |                              |
  |                                    | Create new message:          |
  |                                    | m3 = (m1 * m2) mod n         |
  |                                    |                              |
  |                                    | Forge new signature:         |
  |                                    | s3 = (s1 * s2) mod n         |
  |                                    |                              |
  |                                    |     Publish (m3, s3) ------->|
  |                                    |                              |
  |                                    |                       Verify:
  |                                    |                       m3 ?= s3^e mod n
  |                                    |                              |
  |                                    |                       Result: Valid!
```

```
Key Points:
1. RSA key pair: (n, e) public, (n, d) private
2. Signing: s = m^d mod n
3. Verification: m ?= s^e mod n
4. Attack exploits RSA's multiplicative property:
   (m1 * m2)^d ≡ m1^d * m2^d (mod n)
5. Mallory creates m3 and s3 without knowing d
6. The forged signature verifies as valid

Security Implications:
1. Demonstrates why hashing messages before signing is crucial
2. Shows the need for padding schemes like PSS in RSA signatures
3. Illustrates potential risks in systems using raw RSA signatures

Real-world Example:
In a financial system, this could potentially allow combining
two valid transactions to create a new, seemingly valid
transaction with a different amount.
```

**Task 3 Part 2b Output:**

Demonstrating RSA Signature Malleability

----------------------------------------

Original message 1 (m1): 12345

Original message 2 (m2): 67890

Signature for m1:
26463014065822811363285691449045903578551102558280267211910170438204934651067867168179114148641059345494491210956065381421064578661777773376917991313402184836891307930644487249979557668017665305935344074336647450075011985550270220700959614423862040862976836187117910429969035875143982025575719146472839607694

Signature for m2:
46013343437004384871577298474209080741697305983364222383674432373138699565087070159845038903071637977414083922974225373094305296764636659374996892032451216864238362357160047993923394830346401930264516875915402798956611752717439031584038129687211656600130724238704796831217098272457707445651613176840363398769

Verifying original signatures:

Signature 1 is valid: True

Signature 2 is valid: True

Mallory's new message (m3 = m1 * m2 mod n): 838102050

Mallory's forged signature for m3: 357418705263164239901519375996980330285956624345854497046495148241292994686341250803528080477036427227294732563451131096911686785288324534613311326678403083590657846784108912861916955333310008245602014017082268306265140876882925873227644464566098203775683960638801939776657199040484427749780578336400415931 3