Final Year B.Tech. (CSE) – VII [2024-25]

6CS451: Cryptography and Network Security Lab (C&NS Lab)

Date: 26/08/2024

Assignment 7

PRN: 21510017 **Name:** Onkar Anand Yemul

---

1. Implementation of RSA Algorithm

Ans:

The RSA algorithm is one of the first public-key cryptosystems and is widely used for secure data transmission. It is an asymmetric cryptographic algorithm, meaning it uses a pair of keys: a public key for encryption and a private key for decryption. It relies on the mathematical properties of prime numbers.

<u>How RSA Works:</u>

1. **Key Generation:**
    - Choose two large prime numbers p and q.
    - Compute n = p * q.
    - Compute the totient $\phi(n) = (p-1) * (q-1)$.
    - Choose an encryption key e such that $1 < e < \phi(n)$ and $\gcd(e, \phi(n)) = 1$. The integer e is the public key exponent.
    - Calculate the decryption key d such that $d * e \equiv 1 \pmod{\phi(n)}$. The integer d is the private key exponent.
2. **Encryption:**
    - The public key is (n, e).
    - Given a plaintext message M, the ciphertext C is computed as:

      $C = M^e \bmod n$.
3. **Decryption:**

- The private key is (n, d).
- Given a ciphertext C, the plaintext M is recovered as:

  M = C^d mod n

To implement the RSA algorithm **using large prime numbers with 2048 bits** and converting plaintext into numbers, we'll use the **Crypto library in Python**, which provides the **necessary tools to handle such large prime numbers and perform RSA encryption and decryption.**

The large primes and the strong key sizes make RSA secure against most attacks when implemented correctly.

Python Code:

```python
import random
from sympy import isprime, mod_inverse

def generate_prime_candidate(length):
    """Generate an odd integer randomly."""
    p = random.getrandbits(length)
    # Ensure p is odd
    p |= (1 << length - 1) | 1
    return p

def generate_prime_number(length):
    """Generate a prime number."""
    p = 4
    while not isprime(p):
        p = generate_prime_candidate(length)
    return p

def generate_keypair(keysize):
    """Generate RSA public and private keys."""
    # Generate two large primes p and q
    p = generate_prime_number(keysize)
```

```python
    q = generate_prime_number(keysize)

    print("\np: ", p)
    print("\nq: ", q)

    # Compute n = p * q
    n = p * q

    # Compute Euler's Totient ϕ(n) = (p-1)*(q-1)
    phi = (p - 1) * (q - 1)

    # Choose an integer e such that 1 < e < phi(n) and
gcd(e, phi(n)) = 1
    e = random.randrange(2, phi)
    g = gcd(e, phi)
    while g != 1:
        e = random.randrange(2, phi)
        g = gcd(e, phi)

    # Compute d, the modular inverse of e
    d = mod_inverse(e, phi)

    # Public key (e, n) and Private key (d, n)
    return ((e, n), (d, n))

def gcd(a, b):
    """Compute the greatest common divisor using Euclid's
algorithm."""
    while b != 0:
        a, b = b, a % b
    return a

def encrypt(public_key, plaintext):
    """Encrypt plaintext using the public key."""
    e, n = public_key
    cipher = [pow(ord(char), e, n) for char in plaintext]
    return cipher
```

```python
def decrypt(private_key, ciphertext):
    """Decrypt ciphertext using the private key."""
    d, n = private_key
    plain = [chr(pow(char, d, n)) for char in ciphertext]
    return ''.join(plain)

def main():
    """Run RSA algorithm with a menu-driven interface."""
    print("RSA Encryption/Decryption")

    # Take keysize as input from the user
    keysize = int(input("Enter key size (e.g., 1024, 2048): 
"))

    # Generate public and private keys
    public_key, private_key = generate_keypair(keysize)

    # Menu-driven system
    while True:
        print("\nMenu:")
        print("1. Show Public Key")
        print("2. Show Private Key")
        print("3. Enter a message to encrypt")
        print("4. Enter the encrypted text to decrypt")
        print("5. Exit")

        choice = input("Choose an option: ")

        if choice == '1':
            print(f"\nPublic key: {public_key}")

        elif choice == '2':
            print(f"\nPrivate key: {private_key}")

        elif choice == '3':
            plaintext = input("\nEnter a message to encrypt: 
")
            encrypted_msg = encrypt(public_key, plaintext)
```

```
            print(f"\nEncrypted message: {encrypted_msg}")

        elif choice == '4':
            encrypted_text = input("\nEnter the encrypted text as a list of integers (e.g., [123, 456, ...]): ")
            try:
                # Convert the input string to a list of integers
                encrypted_msg = [int(x) for x in encrypted_text.strip('[]').split(',')]
                decrypted_msg = decrypt(private_key, encrypted_msg)
                print(f"\nDecrypted message: {decrypted_msg}")
            except ValueError:
                print("Invalid input. Please provide the encrypted text in the correct format.")

        elif choice == '5':
            print("Exiting...")
            break

        else:
            print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main()
```

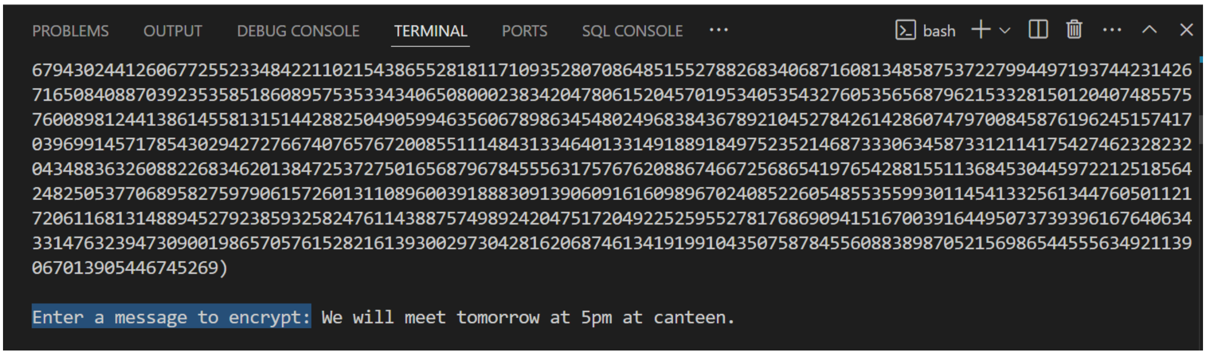
Output:

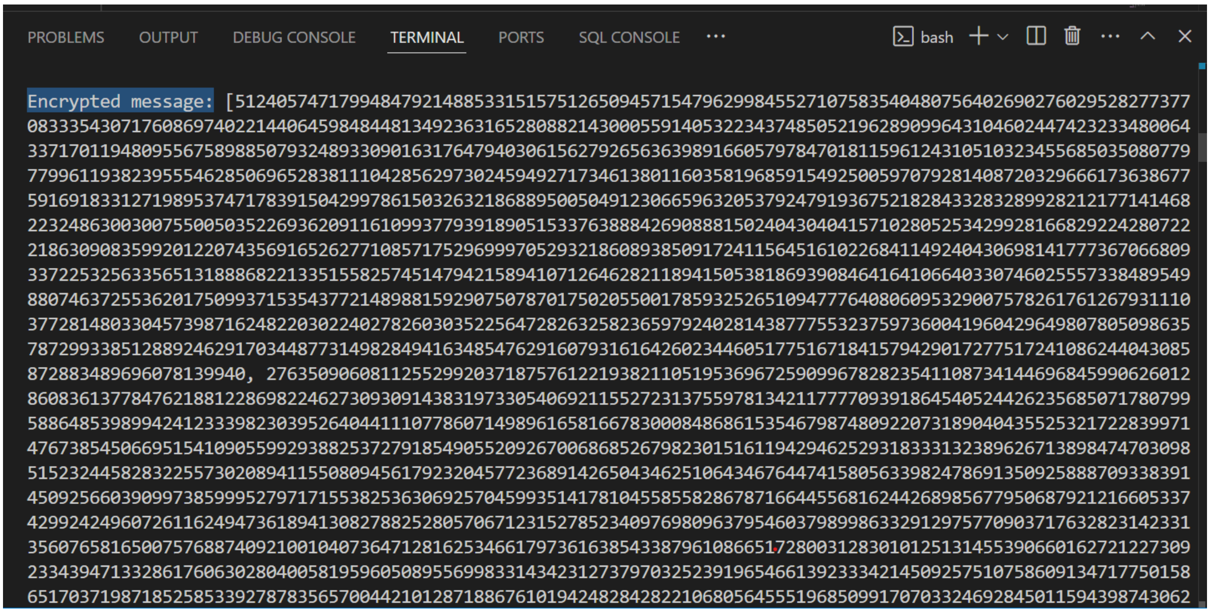
For Keysize = 2048:

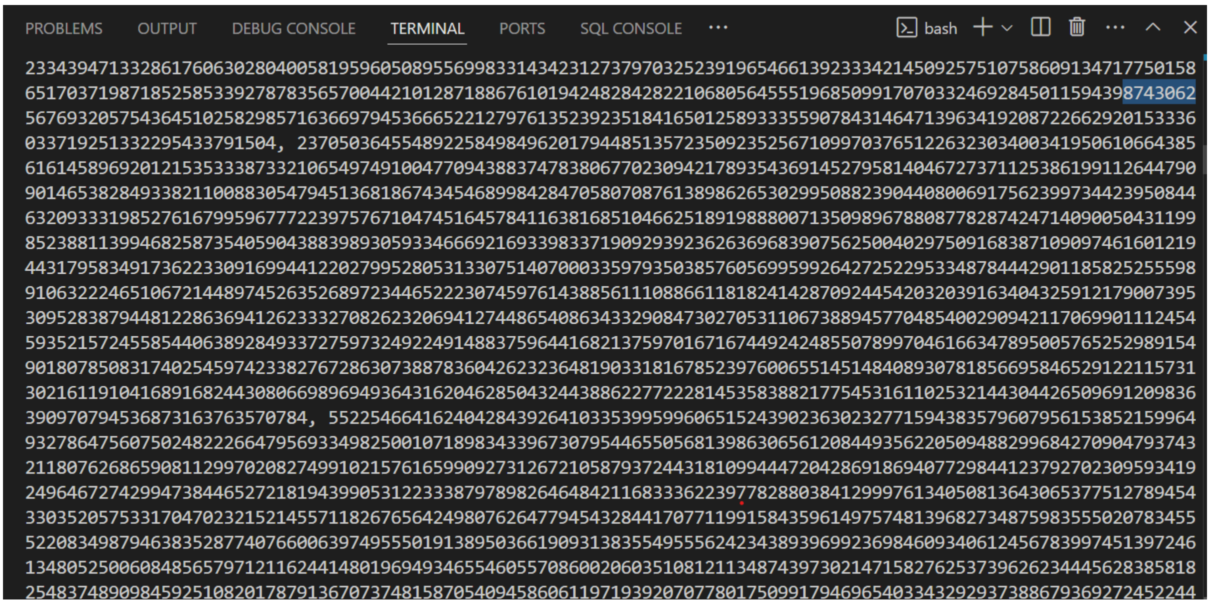
```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   SQL CONSOLE   ...        bash

Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 7
$ python rsa.py
RSA Encryption/Decryption

p:  2819061905820071920297650792763115653673975084276670485623196792830920165173526330834462031863421435498557836
0804688070137673469585309443923392261540725951035243814154348439868497038251696119009360230527777166174628375776
6132352430995436501637717893725833973140345180218247405783997917640788270403957816774578464372754871909951305517
2735070564624822681847412689817403467370787250068582096675565461822123928990359766376028900292375653951193530846
6617203431451149682221998706692104731477567319556070555640022183743152375868569593927655081401174079626310159436
50952725911516467120369892325136716622021260108227532253 4179

q:  2023038924913962339483431931348245731534773780158120341718533668242446350308903636473374542089991980242191319
1877863500524639964570370384321819750805515106511098116517376814598209170469726662726459518537879074520608361839
3191490655985630638872600113592188679611002155233484521561434270464501759651278628641175861431383894545221463270
4516727570600909362680292333088104420590941715812815703162328339548094700644405409813185873319394667572574471649
7725113633939564538951444462847317894125825801401373965254044014101586530778306329856410752048131322496228213468
618623795770314120612409499041935323061889732084110295536711
```

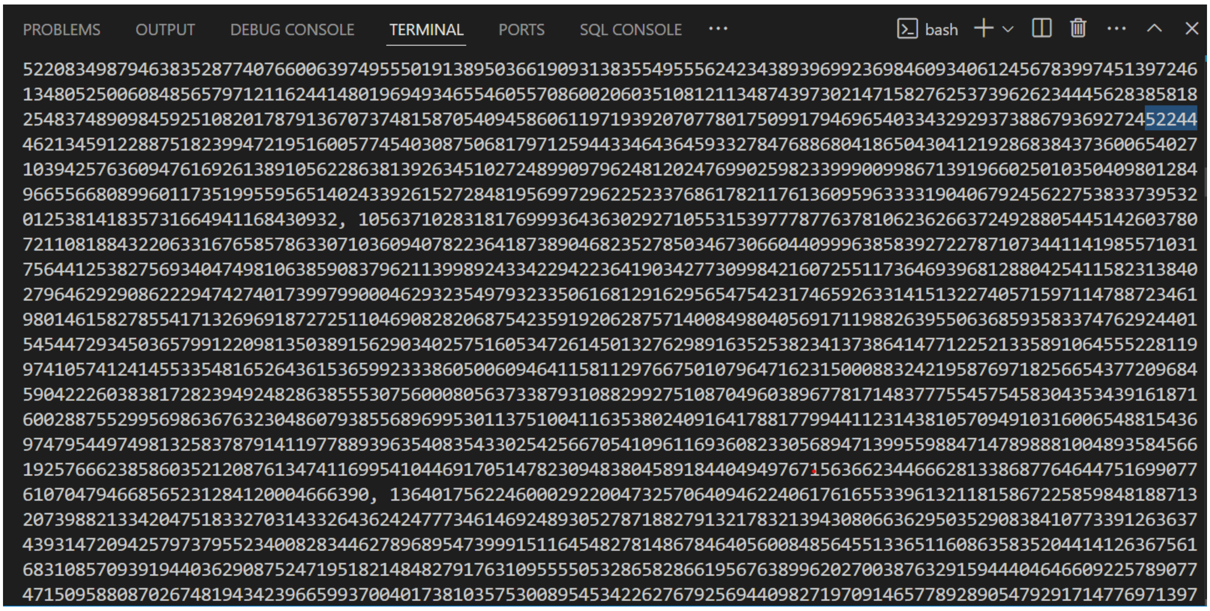
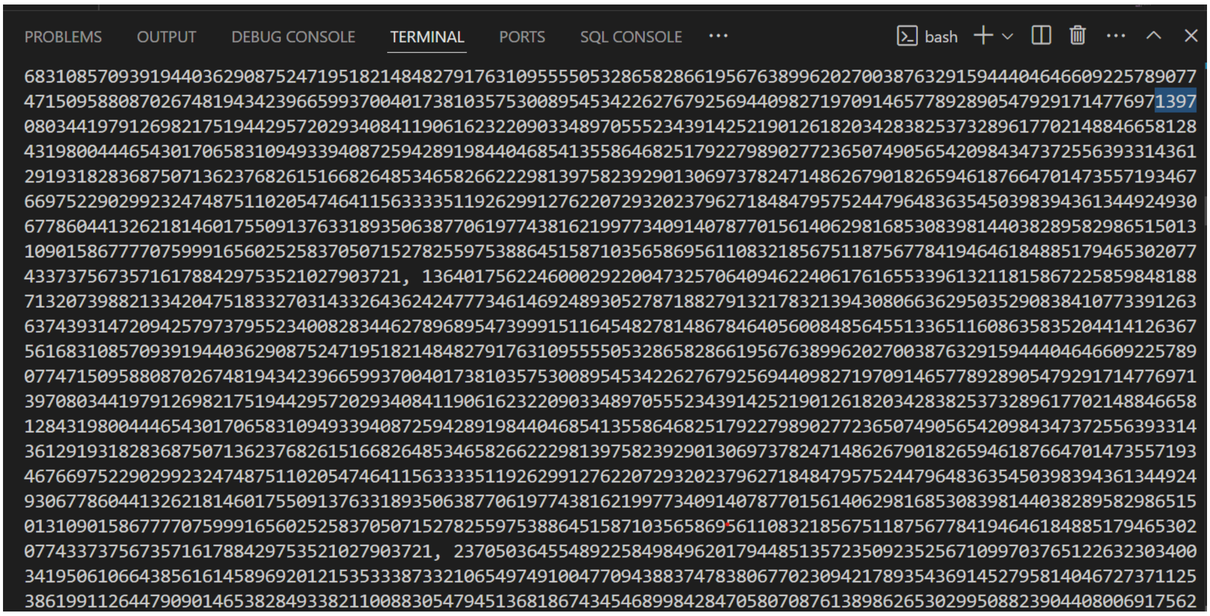
```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   SQL CONSOLE   ...        bash

Public key: (401255109074108980465921566640954685015789317040706324808294074793760149164137220756805659517308539
7563577034112088116199484745849904772683471202284494220027855166620075980082304443429479169622060679052470202693
0411787450330773602245241988236305999716273346537348737295969486379632376410752576096797368837081252281928811543
8499809719125842404252765062095777139821878609672956923385374696158715558204716377452238770185059655619694081228
8569422692664375052473888693495808605266702716229478849133929053928420993361232710271417821148142920046097622051
0134751115024305230586833872283163940971126347370493762167683249279350138621051572137038752639163879113599923330
9999697657072046833313723241862776246905259572370283003045826501387901640905965973037275023538843956560231048782
8759233095527964656322074227134589720583964198338472228019412720308801656813417372777422587951157806363092964073
7000204538193785072373882038138315473581186708555176224529929196915106965452203687517287537338945705812840967668
7332279970809149763585237442913932016321933691479760107609057028670610892258054931818693293885854694914264194271
0361221662850020461698721905478118067116265129741979759545727765323602166136205264690785479859636669826727164872
87962702766029, 5703071967216144049833508406663566756277632566798202845494799117909708761150084757462562502237768
1463715993218638830316814466917858399469951198400035430520383589258485781909494839040424985912321799704336737863
5894453069908989913116312291072460855745348453888281849398462799827369117592775225808313088909583848737859345567
9430244126067725523348422110215438655281811710935280708648515527882683406871608134858753722799449719374423142671
6508408870392353585186089575353343406508000238342047806152045701953405354327605356568796215332815012040748557576
0089812441386145581315144288250490599463560678986345480249683843678921045278426142860747970084587619624515741703
9699145717854302942727667407657672008551114843133464013314918891849752352146873330634587331211417542746232823204
3488363260882268346201384725372750165687967845556317576762088674667256865419765428815511368453044597221251856424
8250537706895827597906157260131108960039188830913906091616098967024085226054855355993011454133256134476050112172
061168131488945279238593258247611438875749892420475172049225259552781768690941516700391644950737393961676406343
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   SQL CONSOLE   ...        bash

4348836326088226834620138472537275016568796784555631757676208867466725686541976542881551136845304459722125185642
4825053770689582759790615726013110896003918883091390609161609896702408522605485535599301145413325613447605011217
2061168131488945279238593258247611438875749892420475172049225259552781768690941516700391644950737393961676406343
3147632394730900198657057615282161393002973042816206874613419199104350758784556088389870521569865445556349211390
670139054467452 69)
Private key: (1256840218386356995189688323038707685641653186029740327340647550364134509109290662185046689670174552
3776322230274710076848581032125390612951371992748543965824550081506023307600931842950553935626855491553293049060
1598553751287131755302512571609821768417825890509984246516408660420509930428864271770342242260551965363189134804
1903871117537519208915798890922202694238164791217504414315614971145144639439612409416349960410086308261935467940
5038165293343941583999320910033581072920219995270514583180165721414097582091899760877238579471036225174133156404
1542019034022727945682626851506648451435260014410312040772671335619168812562657693754769237993568041785389656729
6808329720421521262938163894578028331358218515959379835852071828871766391773345858160699181123153048607315247508
2730714463971944047942101413626292093091492118748673672621064191010845089440568239297473995754530596915564057232
9119521224815141543138560059591202356481629707731400497705103781745333639429193580602701212134166532835234803293
7651957078770457190505003791392322957660656695892507750877065847834698631302413783109016262241762497345561121088
8735225170050740349666551632094583018386681820774219090283279226146593891750533504443241200729080452298006411109
4651591526 6009, 5703071967216144049833508406663566756277632566798202845494799117909708761150084757462562502237768
1463715993218638830316814466917858399469951198400035430520383589258485781909494839040424985912321799704336737863
5894453069908989913116312291072460855745348453888281849398462799827369117592775225808313088909583848737859345567
9430244126067725523348422110215438655281811710935280708648515527882683406871608134858753722799449719374423142671
6508408870392353585186089575353343406508000238342047806152045701953405354327605356568796215332815012040748557576
0089812441386145581315144288250490599463560678986345480249683843678921045278426142860747970084587619624515741 7
```


PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL CONSOLE ···
bash
6794302441260677255233484221102154386552818117109352807086485155278826834068716081348587537227994497193744231426
7165084088703923535851860895753533434065080002383420478061520457019534053543276053565687962153328150120407485575
7600898124413861455813151442882504905994635606789863454802496838436789210452784261428607479700845876196245157417
0396991457178543029427276674076576720085511148431334640133149188918497523521468733306345873312114175427462328232
0434883632608822683462013847253727501656879678455563175767620886746672568654197654288155113684530445972212518564
2482505377068958275979061572601311089600391888309139060916160989670240852260548553559930114541332561344760501121
7206116813148894527923859325824761143887574989242047517204922525955278176869094151670039164495073739396167640634
3314763239473090019865705761528216139300297304281620687461341919910435075878455608838987052156986544555634921139
067013905446745269)
Enter a message to encrypt: We will meet tomorrow at 5pm at canteen.


PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL CONSOLE ···
bash
Encrypted message: [51240574717994847921488533151575126509457154796299845527107583540480756402690276029528277377
0833354307176086974022144064598484481349236316528088214300055914053223437485052196289099643104602447423233480064
3371701194809556758988507932489330901631764794030615627926563639891660579784701811596124310510323455685035080779
7799611938239555462850696528381110428562973024594927173461380116035819685915492500597079281408720329666173638677
5916918331271989537471783915042997861503263218688950050491230665963205379247919367521828433283289928212177141468
2232486300300755005035226936209116109937793918905153376388842690888150240430404157102805253429928166829224280722
2186309083599201220743569165262771085717529699970529321860893850917241156451610226841149240430698141777367066809
3372253256335651318886822133515582574514794215894107126462821189415053818693908464164106640330746025557338489549
8807463725536201750993715354377214898815929075078701750205500178593252651094777640806095329007578261761267931110
3772814803304573987162482203022402782603035225647282632582365979240281438777553237597360041960429649807805098635
7872993385128892462917034487731498284941634854762916079316164260234460517751671841579429017277517241086244043085
872883489696078139940, 27635090608112552992037187576122193821105195369672590996782823541108734144696845990626012
8608361377847621881228698224627309309143831973305406921155272313755978134211777709391864540524426235685071780799
5886485398994241233398230395264044111077860714989616581667830008486861535467987480922073189040435525321722839971
4767385450669515410905599293882537279185490552092670068685267982301516119429462529318333132389626713898474703098
5152324458283225573020894115508094561792320457723689142650434625106434676447415805633982478691350925888709338391
4509256603909973859995279717155382536306925704599351417810455855828678716644556816244268985677950687921216605337
4299242496072611624947361894130827882528057067123152785234097698096379546037989986332912975770903717632823142331
3560765816500757688740921001040736471281625346617973616385433879610866517280031283010125131455390660162721227309
2334394713328617606302804005819596050895569983314342312737970325239196546613923334214509257510758609134717750158
6517037198718525853392787835657004421012871886761019424828428221068056455519685099170703324692845011594398743062


PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL CONSOLE ···
bash
2334394713328617606302804005819596050895569983314342312737970325239196546613923334214509257510758609134717750158
6517037198718525853392787835657004421012871886761019424828428221068056455519685099170703324692845011594398743062
5676932057543645102582985716366979453666522127976135239235184165012589333559078431464713963419208722662920153336
033719251332295433791504, 23705036455489225849849620179448513572350923525671099703765122632303400341950610664385
6161458969201215353338733210654974910047709438837478380677023094217893543691452795814046727371125386199112644790
9014653828493382110088305479451368186743454689984284705807087613898626530299508823904408006917562399734423950844
6320933319852761679959677722397576710474516457841163816851046625189198880071350989678808778287424714090050431199
8523881139946825873540590438839893059334666921693398337190929392362636968390756250040297509168387109097461601219
4431795834917362233091699441220279952805313307514070003359793503857605699599264272522953348784442901185825255598
9106322246510672144897452635268972344652223074597614388561110886611818241428709244542032039163404325912179007395
3095283879448122863694126233327082623206941274486540863433290847302705311067388945770485400290942117069901112454
5935215724558544063892849337275973249224914883759644168213759701671674492424855078997046166347895005765252989154
9018078508317402545974233827672863073887836042623236481903318167852397600655145148408930781856695846529122115731
3021611910416891682443080669896949364316204628504324438862277222814535838821775453161102532144304426509691209836
390970794536873163763570784, 55225466416240428439264103353995996065152439023630232771594383579607956153852159964
9327864756075024822266479569334982500107189834339673079544655056813986306561208449356220509488299684270904793743
2118076268659081129970208274991021576165990927312672105879372443181099444720428691869407729844123792702309593419
2496467274299473844652721819439905312233387978982646484211683336223977828803841299976134050813643065377512789454
3303520575331704702321521455711826765642498076264779454328441707711991584359614975748139682734875983555020783455
5220834987946383528774076600639749555019138950366190931383554955562423438939699236984609340612456783997451397246
1348052500608485657971211624414801969493465546055708600206035108121134874397302147158276253739626234445628385818
2548374890984592510820178791367073748158705409458606119719392070778017509917946965403343292937388679369272452244


PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL CONSOLE ···
bash
5220834987946383528774076600639749555019138950366190931383554955562423438939699236984609340612456783997451397246
1348052500608485657971211624414801969493465546055708600206035108121134874397302147158276253739626234445628385818
2548374890984592510820178791367073748158705409458606119719392070778017509917946965403343292937388679369272452244
4621345912288751823994721951600577454030875068179712594433464364593327847688680418650430412192868384373600654027
1039425763609476169261389105622863813926345102724899097962481202476990259823399900998671391966025010350409801284
9665566808996011735199559565140243392615272848195699729622523376861782117613609596333319040679245622753833739532
0125381418357316649411684309 32, 1056371028318176999364363029271055315397778776378106236266372492880544514260378 0
7211081884322063316765857863307103609407822364187389046823527850346730660440999638583927227871073441141985571031
7564412538275693404749810638590837962113998924334229422364190342773099842160725511736469396812880425411582313840
2796462929086222947427401739979900046293235497932335061681291629565475423174659263314151322740571597114788723461
9801461582785541713269691872725110469082820687542359192062875714008498040569171198826395506368593583374762924401
5454472934503657991220981350389156290340257516053472614501327629891635253823413738641477122521335891064555228119
9741057412414553354816526436153659923338605006094641158112976675010796471623150008832421958769718256654377209684
5904222603838172823949248286385553075600080563733879310882992751087049603896778171483777554575458304353439161871
6002887552995698636763230486079385568969953011375100411635380240916417881779944112314381057094910316006548815436
9747954497498132583787914119778893963540835433025425667054109611693608233056894713995598847147898881004893584566
1925766623858603521208761347411699541044691705147823094838045891844049497671563662344666281338687764644751699077
6107047946685652312841200046663 90, 1364017562246000292200473257064094622406176165533961321181586722585984818871 3
2073988213342047518332703143326436242477734614692489305278718827913217832139430806636295035290838410773391263637
4393147209425797379552340082834462789689547399915116454827814867846405600848564551336511608635835204414126367561
6831085709391944036290875247195182148482791763109555505328658286619567638996202700387632915944404646609225789077
4715095880870267481943423966599370040173810357530089545342262767925694409827197091465778928905479291714776971397

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL CONSOLE ···
bash
6831085709391944036290875247195182148482791763109555505328658286619567638996202700387632915944404646609225789077
4715095880870267481943423966599370040173810357530089545342262767925694409827197091465778928905479291714776971397
0803441979126982175194429572029340841190616232209033489705552343914252190126182034283825373289617702148846658128
4319800444654301706583109493394087259428919844046854135586468251792279890277236507490565420984347372556393314361
2919318283687507136237682615166826485346582662229813975823929013069737824714862679018265946187664701473557193467
6697522902992324748751102054746411563333511926299127622072932023796271848479575244796483635450398394361344924930
6778604413262181460175509137633189350638770619774381621997734091407877015614062981685308398144038289582986515013
1090158677770759991656025258370507152782559753886451587103565869561108321856751187567784194646184885179465302077
4337375673571617884297535210279037 21, 1364017562246000292200473257064094622406176165533961321181586722585984818 8
7132073988213342047518332703143326436242477734614692489305278718827913217832139430806636295035290838410773391263
6374393147209425797379552340082834462789689547399915116454827814867846405600848564551336511608635835204414126367
5616831085709391944036290875247195182148482791763109555505328658286619567638996202700387632915944404646609225789
0774715095880870267481943423966599370040173810357530089545342262767925694409827197091465778928905479291714776971
3970803441979126982175194429572029340841190616232209033489705552343914252190126182034283825373289617702148846658
1284319800444654301706583109493394087259428919844046854135586468251792279890277236507490565420984347372556393314
3612919318283687507136237682615166826485346582662229813975823929013069737824714862679018265946187664701473557193
4676697522902992324748751102054746411563333511926299127622072932023796271848479575244796483635450398394361344924
9306778604413262181460175509137633189350638770619774381621997734091407877015614062981685308398144038289582986515
0131090158677770759991656025258370507152782559753886451587103565869561108321856751187567784194646184885179465302
0774337375673571617884297535210279037 21, 2370503645548922584984962017944851357235092352567109970376512263230340 0
3419506106643856161458969201215353338733210654974910047709438837478380677023094217893543691452795814046727371125
3861991126447909014653828493382110088305479451368186743454689984284705807087613898626530299508823904408006917562

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    SQL CONSOLE    ...        bash
910048033052601280036859224632031916928387685811283156070875895440790989065405111489335386808838014341786685990З
4058533708874771037003139795862055837144163552918029834505968755720570121631767834144528464251239783594207362368
9739678726727181784852025386797980699598838700150479223062044480082299967033903344268816170333771533808283622333
4220171600398779468870869247527604748330305662065811275483486463637325210585904229629085207895144378177796771996
2425681943056619355123950535714886258097111836264993523397107647991586316535845384158316049800058885316786044794
82932475958713645, 4213687425340908524073391105252099760204772305727938085616392189839735577744426642995707498 41
4253895096616010890308139645537599715812445641868854145252180613355701538703539444922063827343960277052904371846
0491867484886849836788968759537678085921026139389766640479525582574837172267224995272121051023499021928719552991
2586339949443762522292413438952476944567796214439352781571895241660888115982829717615997546705540902205499871333
4046621079250030283860627197173640409982723145353969557949405488282277944370641863297926346329624120033218319429
0315493607445576677285567346785502966905855166056181144098717895173066936235399761224454727915898391708560654922
8564045796700886836329236624440385177391838863797023274001517601323612331632938503252762608507021610653549535277
1232837102945771269467755521477296778342707489179342120183278791219797857109577296820847891434938513040540067885
7494218680264520957667919823153698236202754945743846157427829404222247071608819909996281603336356237857331614109
7282470056218721417206344725452407424299201360274032752501957588782381754098710689234861287999842680624781774336
9997148442417947615430392067876952103958885760576235395324401519068218885378341552727660067403478750838369716247
910857560182899112 92]

Decrypted message: We will meet tomorrow at 5pm at canteen.

Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 7
$
```

For Keysize = 16:

```
Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 7 (main)
$ python rsa.py
RSA Encryption/Decryption
Enter key size (e.g., 1024, 2048): 16

p:  42899

q:  63367

Menu:
1. Show Public Key
2. Show Private Key
3. Enter a message to encrypt
4. Enter the encrypted text to decrypt
5. Exit
Choose an option: 1

Public key: (899822465, 2718380933)
```

```
Menu:
1. Show Public Key
2. Show Private Key
3. Enter a message to encrypt
4. Enter the encrypted text to decrypt
5. Exit
Choose an option: 2

Private key: (2580611021, 2718380933)

Menu:
1. Show Public Key
2. Show Private Key
3. Enter a message to encrypt
4. Enter the encrypted text to decrypt
5. Exit
Choose an option: 3

Enter a message to encrypt: We will cancel their order.

Encrypted message: [580383014, 1674094199, 1608554978, 1548141833, 2147988317, 2113262506, 21132625
06, 1608554978, 539500294, 324068690, 1171192377, 539500294, 1674094199, 2113262506, 1608554978, 12
31643643, 606790713, 1674094199, 2147988317, 1675740237, 1608554978, 2009642701, 1675740237, 191341
3955, 1674094199, 1675740237, 2518779932]
```

```
Menu:
1. Show Public Key
2. Show Private Key
3. Enter a message to encrypt
4. Enter the encrypted text to decrypt
5. Exit
Choose an option: 4

Enter the encrypted text as a list of integers (e.g., [123, 456, ...]): [580383014, 1674094199, 160
8554978, 1548141833, 2147988317, 2113262506, 2113262506, 1608554978, 539500294, 324068690, 11711923
77, 539500294, 1674094199, 2113262506, 1608554978, 1231643643, 606790713, 1674094199, 2147988317, 1
675740237, 1608554978, 2009642701, 1675740237, 1913413955, 1674094199, 1675740237, 2518779932]

Decrypted message: We will cancel their order.

Menu:
1. Show Public Key
2. Show Private Key
3. Enter a message to encrypt
4. Enter the encrypted text to decrypt
5. Exit
Choose an option: 5
Exiting...

Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 7 (main)
$
```

Practical Applications of RSA

- **Secure Communication:** Encrypting emails and messages.

- **Digital Signatures**: Verifying the authenticity of a message or document.
- **Key Exchange**: Securely exchanging keys for symmetric encryption algorithms.

RSA is widely used in various security protocols, including SSL/TLS for secure internet communications.

RSA ensures security through the difficulty of factoring large numbers. It is commonly used for securing sensitive data, digital signatures, and in SSL/TLS protocols.

Final Year B.Tech. (CSE) – VII [2024-25]

6CS451: Cryptography and Network Security Lab (C&NS Lab)

Date: 26/08/2024

Assignment 8

PRN: 21510017 **Name:** Onkar Anand Yemul

---

1. **Implement the Diffie-Hellman Key Exchange algorithm for a given problem**

Ans:

The Diffie-Hellman Key Exchange is a cryptographic algorithm that allows two parties to securely share a secret key over a public channel. This shared key can then be used for encrypted communication. The algorithm allows two parties to generate a shared secret key that can be used for subsequent encryption and decryption, even if the exchange itself is observed by an eavesdropper.

<u>How Diffie-Hellman Works:</u>

1. **Public Parameters**:
   - Both parties agree on a large prime number p and a base g (a primitive root modulo p).
2. **Key Exchange Process**:
   - **Party A** selects a private key 'a' and computes A = g^ a mod p, then sends A to Party B.
   - **Party B** selects a private key 'b' and computes B = g^ b mod p, then sends B to Party A.
3. **Shared Secret**:
   - **Party A** computes the shared secret as S = B^ a mod p.
   - **Party B** computes the shared secret as S = A^ b mod p.

Since both calculations result in the same value, S becomes the shared secret key, even though an eavesdropper only knows p, g, A, and B.

The Diffie-Hellman algorithm securely establishes a shared secret key without transmitting it directly, making it fundamental for secure communications in protocols like SSL/TLS.

To implement the Diffie-Hellman Key Exchange algorithm for client-server communication across two different machines, we will create two Python programs: one for the client and one for the server. The server will generate its public key and share it with the client, and vice versa. Both will then calculate the shared secret key independently.

Python Code:

Client-side program:

```python
import socket
import random

def generate_private_key(p):
    """Generate a private key."""
    private_key = random.randint(2, p - 2)
    print(f"\nGenerated Private Key: {private_key}")
    return private_key

def calculate_public_key(g, private_key, p):
    """Calculate the public key."""
    public_key = pow(g, private_key, p)
    print(f"\nCalculated Public Key: {public_key}")
    return public_key

def calculate_shared_secret(public_key, private_key, p):
    """Calculate the shared secret."""
    shared_secret = pow(public_key, private_key, p)
```

```python
    return shared_secret

def start_client(server_host='localhost', server_port=5000):
    # Take p and g as user input
    p = int(input("\nEnter a prime number (p): "))
    g = int(input("\nEnter a primitive root (g): "))

    # Generate client's private and public keys
    private_key = generate_private_key(p)
    public_key = calculate_public_key(g, private_key, p)

    # Create client socket
    client_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
    client_socket.connect((server_host, server_port))

    # Receive the server's public key
    server_public_key =
int(client_socket.recv(1024).decode())
    print(f"\nReceived Server's Public Key:
{server_public_key}")

    # Send the client's public key to the server
    client_socket.sendall(str(public_key).encode())

    # Calculate the shared secret
    shared_secret =
calculate_shared_secret(server_public_key, private_key, p)
    print(f"\nShared Secret (Client): {shared_secret}")

    client_socket.close()

if __name__ == "__main__":
    start_client()
```

Server-side program:

```python
import socket
import random

def generate_private_key(p):
    """Generate a private key."""
    private_key = random.randint(2, p - 2)
    print(f"\nGenerated Server's Private Key:
{private_key}")
    return private_key

def calculate_public_key(g, private_key, p):
    """Calculate the public key."""
    public_key = pow(g, private_key, p)
    print(f"\nCalculated Server's Public Key: {public_key}")
    return public_key

def calculate_shared_secret(public_key, private_key, p):
    """Calculate the shared secret."""
    shared_secret = pow(public_key, private_key, p)
    return shared_secret

def start_server(host='localhost', port=5000):
    # Take p and g as user input
    p = int(input("\nEnter a prime number (p): "))
    g = int(input("\nEnter a primitive root (g): "))

    # Generate server's private and public keys
    private_key = generate_private_key(p)
    public_key = calculate_public_key(g, private_key, p)

    # Create server socket
    server_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
    server_socket.bind((host, port))
    server_socket.listen(1)
    print(f"\nServer started. Listening on {host}:{port}")

    conn, addr = server_socket.accept()
```

```python
    print(f"\nConnected by {addr}")

    # Send the server's public key to the client
    conn.sendall(str(public_key).encode())

    # Receive the client's public key
    client_public_key = int(conn.recv(1024).decode())
    print(f"\nReceived Client's Public Key:
{client_public_key}")

    # Calculate the shared secret
    shared_secret =
calculate_shared_secret(client_public_key, private_key, p)
    print(f"\nShared Secret (Server): {shared_secret}")

    conn.close()
    server_socket.close()

if __name__ == "__main__":
    start_server()

# For
# p: 1014273607262027361
# For
# g: 2
```

Output:

Server output–

```
MINGW64:/c/Users/Onkar/Downloads/CNS Lab/Assignment 8

Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 8 (main)
$ python server_diffie_hellman.py

Enter a prime number (p): 1014273607262027361

Enter a primitive root (g): 2

Generated Server's Private Key: 710999113647365409

Calculated Server's Public Key: 261836127913730729

Server started. Listening on localhost:5000
```

```
MINGW64:/c/Users/Onkar/Downloads/CNS Lab/Assignment 8

Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 8 (main)
$ python server_diffie_hellman.py

Enter a prime number (p): 1014273607262027361

Enter a primitive root (g): 2

Generated Server's Private Key: 710999113647365409

Calculated Server's Public Key: 261836127913730729

Server started. Listening on localhost:5000

Connected by ('127.0.0.1', 52965)

Received Client's Public Key: 384265924014607322

Shared Secret (Server): 194219320603336 79

Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 8 (main)
$
```

Client output-

```
MINGW64:/c/Users/Onkar/Downloads/CNS Lab/Assignment 8

Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 8 (main)
$ python client_diffie_hellman.py

Enter a prime number (p): 1014273607262027361

Enter a primitive root (g): 2

Generated Private Key: 116258217018027939

Calculated Public Key: 384265924014607322

Received Server's Public Key: 261836127913730729

Shared Secret (Client): 194219320603336797

Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 8 (main)
$
```

Practical Applications of Diffie-Hellman:

- **Secure Communication**: Establishing a shared secret for symmetric encryption over an insecure channel.
- **VPNs**: Secure key exchange for Virtual Private Networks.
- **TLS/SSL**: Part of the key exchange process in securing internet communications.

The Diffie-Hellman algorithm forms the basis of many modern cryptographic protocols and is crucial for secure communication in distributed systems.

Final Year B.Tech. (CSE) – VII [2024-25]

6CS451: Cryptography and Network Security Lab (C&NS Lab)

Date: 01/10/2024

Assignment 9

PRN: 21510017 **Name:** Onkar Anand Yemul

---

9. Calculate the message digest of a text using the SHA-1 algorithm

Ans:

SHA-1 Algorithm:

SHA-1 (Secure Hash Algorithm 1) is a cryptographic hash function that produces a 160-bit hash value (20 bytes), often referred to as a **message digest**. It takes an input message of any size and outputs a fixed-size hash, which is commonly represented as a 40-character hexadecimal number. It was developed by the National Security Agency (NSA) and published by NIST in 1993.

Message Digest of a Text:

A **message digest** is a fixed-size numerical representation of the contents of a message. For SHA-1, this digest is 160 bits long, and any change in the input message, even a single bit, will result in a drastically different digest (this is known as the **avalanche effect**). The message digest ensures data integrity by allowing anyone to verify that the message hasn't been altered.

To implement the SHA-1 algorithm without using Python's *hashlib* library, we need to follow the algorithm's steps manually, which involves bitwise operations, padding the input message, and processing it in blocks.

Overview of SHA-1 Algorithm:

1. **Padding the message**: The message is padded so that its length becomes a multiple of 512 bits.

2. **Initialize hash values**: There are five constants (H0, H1, H2, H3, H4) initialized to specific values.
3. **Processing the message in blocks**: The message is processed in chunks of 512 bits, and the hash is updated after each chunk.
4. **Final output**: After all blocks are processed, the hash digest is formed by concatenating the values of H0, H1, H2, H3, and H4.

<u>Python Code for SHA-1 Implementation without using Python's built-in hashlib library:</u>

```
import struct

# Helper functions for bitwise operations
def left_rotate(n, b):
    """Left rotate a 32-bit integer n by b bits."""
    return ((n << b) | (n >> (32 - b))) & 0xFFFFFFFF


def sha1_padding(message):
    """Pad the message to ensure the length is a multiple of 
512 bits."""
    original_byte_len = len(message)
    original_bit_len = original_byte_len * 8

    # Padding with a '1' bit followed by '0's, and add the 
original message length in bits at the end
    message += b'\x80'  # append the bit '1' (10000000 in 
binary)

    # Pad with 0s so that the message length is 64 bits 
short of a multiple of 512
    while (len(message) * 8) % 512 != 448:
        message += b'\x00'

    # Append the length of the original message in bits (64-
bit big-endian integer)
```

```
    message += struct.pack('>Q', original_bit_len)

    return message


def sha1(message):
    """Calculate the SHA-1 hash of a message."""
    # Initial hash values (first 32 bits of the fractional
parts of the square roots of the first 5 primes)
    h0 = 0x67452301
    h1 = 0xEFCDAB89
    h2 = 0x98BADCFE
    h3 = 0x10325476
    h4 = 0xC3D2E1F0

    # Preprocessing: padding the message
    message = sha1_padding(message)

    # Process the message in successive 512-bit chunks (64
bytes each)
    for i in range(0, len(message), 64):
        chunk = message[i:i + 64]

        # Break chunk into sixteen 32-bit big-endian words
        w = [0] * 80
        for j in range(16):
            w[j] = struct.unpack('>I',
chunk[j*4:(j*4)+4])[0]

        # Extend the sixteen 32-bit words into eighty 32-bit
words
        for j in range(16, 80):
            w[j] = left_rotate((w[j-3] ^ w[j-8] ^ w[j-14] ^
w[j-16]), 1)

        # Initialize hash value for this chunk
        a, b, c, d, e = h0, h1, h2, h3, h4

        # Main loop
```

```python
        for j in range(80):
            if 0 <= j <= 19:
                f = (b & c) | ((~b) & d)
                k = 0x5A827999
            elif 20 <= j <= 39:
                f = b ^ c ^ d
                k = 0x6ED9EBA1
            elif 40 <= j <= 59:
                f = (b & c) | (b & d) | (c & d)
                k = 0x8F1BBCDC
            elif 60 <= j <= 79:
                f = b ^ c ^ d
                k = 0xCA62C1D6

            temp = (left_rotate(a, 5) + f + e + k + w[j]) &
0xFFFFFFFF
            e = d
            d = c
            c = left_rotate(b, 30)
            b = a
            a = temp

        # Add this chunk's hash to the result so far
        h0 = (h0 + a) & 0xFFFFFFFF
        h1 = (h1 + b) & 0xFFFFFFFF
        h2 = (h2 + c) & 0xFFFFFFFF
        h3 = (h3 + d) & 0xFFFFFFFF
        h4 = (h4 + e) & 0xFFFFFFFF

    # Produce the final hash value (big-endian)
    return '{:08x}{:08x}{:08x}{:08x}{:08x}'.format(h0, h1,
h2, h3, h4)


if __name__ == "__main__":
    # Input text
    text = input("Enter the text to calculate SHA-1 hash: ")

    # Convert the text to bytes and compute SHA-1 hash
```

```python
    sha1_digest = sha1(text.encode('utf-8'))

    # Output the SHA-1 hash
    print(f"SHA-1 Digest: {sha1_digest}")
```

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    SQL CONSOLE    GITLENS    ...

Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 9 (main)
$ python sha1_without_using_hashlib.py
Enter the text to calculate SHA-1 hash: We will have meeting at 5pm tomorrow.
SHA-1 Digest: 1d9744c575c698f088f2e0fa90c29311abaa1ffb

Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 9 (main)
$ python sha1_without_using_hashlib.py
Enter the text to calculate SHA-1 hash: We will meet at CCF.
SHA-1 Digest: f36ccffcd2c4ef953b0ea93914c6e40f9c8b5afb

Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 9 (main)
$ 
```

To calculate the message digest of a text using the SHA-1 algorithm in Python, you can use the hashlib library, which provides easy access to various hash algorithms, including SHA-1.

1. **hashlib library**:
    - The hashlib library provides various cryptographic hashing algorithms including SHA-1, SHA-256, MD5, etc.
2. **SHA-1 Hash Object**:
    - hashlib.sha1() creates a new SHA-1 hash object.
3. **Updating the Hash**:

- The update() method takes the input text (which is first encoded into bytes) and updates the hash object with that data.

4. **Getting the Digest**:

- The hexdigest() method returns the hash value as a hexadecimal string.

<u>Python Code for SHA-1 Message Digest Calculation using hashlib library:</u>

```python
import hashlib


def calculate_sha1(text):
    # Create a new SHA-1 hash object
    sha1_hash = hashlib.sha1()

    # Encode the input text to bytes and update the hash
object
    sha1_hash.update(text.encode('utf-8'))

    # Get the hexadecimal representation of the digest
    digest = sha1_hash.hexdigest()

    return digest


if __name__ == "__main__":
    # Input text
    text = input("Enter the text to calculate SHA-1 hash: ")

    # Calculate SHA-1 message digest
    sha1_digest = calculate_sha1(text)

    # Output the result
    print(f"SHA-1 Digest: {sha1_digest}")
```

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    SQL CONSOLE    GITLENS    ...

Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 9 (main)
$ python sha1_using_hashlib.py
Enter the text to calculate SHA-1 hash: We will have meeting at 5pm tomorrow.
SHA-1 Digest: 1d9744c575c698f088f2e0fa90c29311abaa1ffb

Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 9 (main)
$ python sha1_using_hashlib.py
Enter the text to calculate SHA-1 hash: We will meet at the CCF.
SHA-1 Digest: faee421202ee56ead28654f04bbcff22add29fca

Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 9 (main)
$ 
```

Advantages of SHA-1:

- **Speed and Efficiency**: SHA-1 was designed to be computationally efficient and can process large amounts of data quickly.
- **Widespread Use**: It has been widely adopted and used for various cryptographic applications, including digital signatures, certificates, and integrity checks.
- **Fixed-Length Output**: Regardless of the input size, the output is always 160 bits, making it convenient to use in various security protocols.

Disadvantages of SHA-1:

- **Weakness to Collisions**: SHA-1 is vulnerable to **collision attacks**, where two different inputs produce the same hash output. This reduces its effectiveness in ensuring data integrity and security.
- **Security Deprecation**: Due to these vulnerabilities, SHA-1 is no longer considered secure for cryptographic purposes. Most modern systems and protocols, including browsers and SSL certificates, have moved to stronger hash functions like SHA-256 or SHA-3.

Importance of SHA-1:

- **Legacy Systems**: Despite its vulnerabilities, SHA-1 was used for many years in security applications such as digital signatures and certificates.
- **Data Integrity**: SHA-1 can still be used to check the integrity of data, ensuring that files have not been altered during transmission.

Security Risks and Vulnerabilities of SHA-1:

- **Collision Attacks**: The primary vulnerability is the possibility of collision attacks. This means that an attacker could potentially create two different messages with the same hash, compromising the authenticity of the data.
- **Birthday Attack**: A specific type of attack known as a **birthday attack** makes it easier to find collisions in SHA-1 due to its 160-bit length, reducing the security level.
- **Deprecation in Modern Systems**: Due to these weaknesses, SHA-1 has been deprecated in most cryptographic protocols like TLS (Transport Layer Security) and digital certificates, where stronger algorithms like SHA-256 are preferred.

While SHA-1 played a significant role in the development of cryptographic standards, its vulnerabilities, especially to collision attacks, have made it unsuitable for modern security applications. Understanding SHA-1's purpose and limitations is important, especially when dealing with legacy systems or understanding the evolution of cryptographic hash functions.

Practical Applications of SHA-1:

1. **Digital Signatures**: SHA-1 was commonly used in creating digital signatures to ensure the authenticity and integrity of documents. It

would generate a hash of the message, which is then signed by a private key.

2. **File Integrity Verification**: SHA-1 was used to generate checksums for files to verify that files were not altered during transfer or storage. The recipient could compare the hash of the received file with the original hash to ensure integrity.
3. **Version Control Systems**: In systems like Git, SHA-1 hashes were used to identify commits, ensuring the integrity and tracking of changes in code repositories.
4. **SSL Certificates**: Until 2017, SHA-1 was used in SSL/TLS certificates for secure web communications. The hash was part of the process to ensure a website's identity and secure data transmission.
5. **Password Hashing**: SHA-1 was once used for hashing passwords in databases, providing a layer of security by storing a hashed version of the password instead of the plaintext.

Despite these applications, most systems have transitioned to more secure alternatives due to SHA-1's vulnerabilities.

SHA-512 Algorithm:

SHA-512 (Secure Hash Algorithm 512-bit) is part of the SHA-2 family and produces a 512-bit message digest. It's a cryptographic hash function designed to provide higher security by generating a unique, fixed-length hash value from input data. SHA-512 is widely used for its robust security features.

Overview of SHA-512 Algorithm

SHA-512 is a cryptographic hash function that converts any input (such as a message or file) into a fixed 512-bit hash. The algorithm processes the input data in blocks and applies multiple rounds of complex operations to produce a unique hash value.

Key Steps in the SHA-512 Algorithm:

1. **Padding the Message**:
   The input message is padded to ensure its length is congruent to 896 bits modulo 1024. Padding involves adding a single '1' bit followed by enough '0' bits, so that the message length becomes 1024 bits less than a multiple of 1024. The last 128 bits are used to store the original length of the message.
2. **Breaking the Message into Blocks**:
   The padded message is divided into 1024-bit blocks for processing. Each block will undergo the hashing process individually.
3. **Initialize Hash Values**:
   The algorithm uses eight 64-bit initial hash values, which are constant and specified by the SHA-512 standard. These values form the basis of the hash computation.
4. **Processing Each Block**:
   For each 1024-bit block:
   - **Message Expansion**: The 1024-bit block is expanded into 80 64-bit words. These words are used in the main hashing loop.

- **Compression Function**: The main loop processes the block with bitwise operations, modular additions, and logical functions (such as AND, XOR, OR). A set of 80 constant values is used along with the expanded message words.
- The hash values are updated after each round using these operations.

5. **Update Hash Values**:
   After processing each block, the intermediate hash values are updated. These values accumulate the results of each block's computations.

6. **Concatenate Final Hash**:
   After all the blocks have been processed, the final 512-bit hash is produced by concatenating the updated hash values from all rounds.

Python Code for SHA-512 Implementation using Python's built-in hashlib library:

```
import hashlib

# Function to hash a message using SHA-512
def sha512_encrypt(message):
    sha512_hash = hashlib.sha512()
    sha512_hash.update(message.encode('utf-8'))  # Convert
the message to bytes
    return sha512_hash.hexdigest()

# Function to verify the hash (like a decryption process)
def verify_hash(original_message, provided_hash):
    original_hash = sha512_encrypt(original_message)
    return original_hash == provided_hash

# Menu-driven system
def menu():
    while True:
        print("\n===== SHA-512 Hashing System =====")
```

```python
        print("1. Encrypt a message using SHA-512")
        print("2. Verify a message against a given hash")
        print("3. Exit")

        choice = input("Enter your choice (1/2/3): ")

        if choice == '1':
            message = input("Enter the message to hash: ")
            hashed_message = sha512_encrypt(message)
            print(f"\nSHA-512 Hash: {hashed_message}")

        elif choice == '2':
            original_message = input("Enter the original 
message: ")
            provided_hash = input("Enter the hash to verify 
against: ")

            if verify_hash(original_message, provided_hash):
                print("\nVerification successful! The 
message matches the provided hash.")
            else:
                print("\nVerification failed! The message 
does not match the provided hash.")

        elif choice == '3':
            print("Exiting the program...")
            break

        else:
            print("Invalid choice. Please choose a valid 
option.")

if __name__ == "__main__":
    menu()
```

<u>Output:</u>

```
Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 9 (main)
$ python sha512.py

===== SHA-512 Hashing System =====
1. Encrypt a message using SHA-512
2. Verify a message against a given hash
3. Exit
Enter your choice (1/2/3): 1
Enter the message to hash: We will attack on Pakistan on 11 November, 2019.

SHA-512 Hash: 6d078e5048722b22c86e92e7390130d3a3b26ed6f36fee69e61eef3e6bccf4b59e42cd4645e142648396a83bea77b55317
e50607742febfaba116c59eeee88be

===== SHA-512 Hashing System =====
1. Encrypt a message using SHA-512
2. Verify a message against a given hash
3. Exit
Enter your choice (1/2/3): 2
Enter the original message: We will attack on Pakistan on 11 November, 2019.
Enter the hash to verify against: 6d078e5048722b22c86e92e7390130d3a3b26ed6f36fee69e61eef3e6bccf4b59e42cd4645e142
648396a83bea77b55317e50607742febfaba116c59eeee88be

Verification successful! The message matches the provided hash.
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SQL CONSOLE  GITLENS  ...  bash

Enter your choice (1/2/3): 2
Enter the original message: We will attack on Pakistan on 11 November, 2019.
Enter the hash to verify against: 6d078e5048722b22c86e92e7390130d3a3b26ed6f36fee69e61eef3e6bccf4b59e42cd4645e142
648396a83bea77b55317e50607742febfaba116c59eeee88be

Verification successful! The message matches the provided hash.

===== SHA-512 Hashing System =====
1. Encrypt a message using SHA-512
2. Verify a message against a given hash
3. Exit
Enter your choice (1/2/3): 2
Enter the original message: We will attack on Pakistan on 11th November, 2019.
Enter the hash to verify against: 6d078e5048722b22c86e92e7390130d3a3b26ed6f36fee69e61eef3e6bccf4b59e42cd4645e142
648396a83bea77b55317e50607742febfaba116c59eeee88be

Verification failed! The message does not match the provided hash.

===== SHA-512 Hashing System =====
1. Encrypt a message using SHA-512
2. Verify a message against a given hash
3. Exit
Enter your choice (1/2/3): 3
Exiting the program...

Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 9 (main)
```

Advantages of SHA-512:

1. **High Security**: SHA-512 produces a 512-bit hash, making it much harder to break with brute force attacks compared to smaller hash sizes like SHA-1.

2. **Collision Resistance**: It offers strong resistance to collision attacks, meaning it's very unlikely two different inputs will produce the same hash.

3. **Efficiency**: Despite its large output size, SHA-512 is designed to be computationally efficient on modern hardware.
4. **Compatibility with SHA-2 Family**: SHA-512 shares its core design with other SHA-2 algorithms, making it easier to switch between different levels of security.

<u>Disadvantages of SHA-512:</u>

1. **Higher Computational Cost**: Because of its larger size, SHA-512 may require more processing power and memory, making it slower on less powerful devices.
2. **Large Hash Size**: The 512-bit hash is larger, which may not be necessary for all applications, particularly when storage or bandwidth is a concern.
3. **Overkill for Small Applications**: In some use cases, the high security provided by SHA-512 might be unnecessary, and a smaller hash size like SHA-256 might suffice.

<u>Importance of SHA-512:</u>

SHA-512 is critical in contexts where strong security is essential, particularly in environments that demand protection against sophisticated attacks. Its high bit-length and resistance to common cryptographic attacks make it crucial for protecting sensitive data.

<u>Practical Applications of SHA-512:</u>

1. **Digital Signatures and Certificates**: SHA-512 is often used in creating digital signatures and securing SSL/TLS certificates to verify the authenticity and integrity of data.
2. **File Integrity**: It's used in verifying file integrity by generating checksums to ensure that files have not been tampered with during transfer or storage.

3. **Cryptocurrency**: SHA-512 is used in blockchain technology to secure transactions and validate blocks.
4. **Password Hashing**: It's commonly used in securely hashing passwords in databases, making stored passwords difficult to reverse-engineer.
5. **Secure Communication**: SHA-512 plays a role in securing communications over networks by being part of cryptographic protocols such as TLS.

Security Risks and Vulnerabilities:

- **Larger Hash Size Overhead**: While SHA-512 is more secure than smaller hashes, the added size may introduce performance issues for systems that don't need this level of security.
- **Quantum Computing Threat**: In the future, quantum computing might pose a threat to even robust algorithms like SHA-512, necessitating the development of quantum-resistant algorithms.

Despite its high security, SHA-512 is still vulnerable to advances in technology, but for now, it remains one of the strongest cryptographic hash functions available.

Final Year B.Tech. (CSE) – VII [2024-25]

6CS451: Cryptography and Network Security Lab (C&NS Lab)

Date: 08/10/2024

Assignment 10

PRN: 21510017 **Name:** Onkar Anand Yemul

---

10. Implement the SIGNATURE SCHEME - Digital Signature Standard

Ans:

To implement the **Digital Signature Standard (DSS)**, we need to understand its process, which involves the **Digital Signature Algorithm (DSA)**. The DSS is a Federal Information Processing Standard (FIPS) for digital signatures, and it involves three main stages:

1. **Key Generation**: Generate a public and private key pair.
2. **Signature Generation**: Use the private key to sign a message.
3. **Signature Verification**: Use the public key to verify the authenticity of the message.

<u>Overview of the DSA Algorithm:</u>

- DSA involves a pair of keys: **private key** (used for signing) and **public key** (used for verification).
- The signature is generated by applying a **hashing algorithm** (such as SHA-1 or SHA-256) to the message, which is then signed using the private key.
- The signature is verified using the public key.

<u>Python Implementation of the DSA (Digital Signature Algorithm) using Python's *cryptography* library for cryptographic operations:</u>

- Using a library here simplifies the task

Steps:

1. **Key Generation**: Generate the DSA keys.

   - A DSA private key is generated using dsa.generate_private_key(key_size=2048). The key size can be adjusted (1024, 2048, or 3072 bits), but 2048 is commonly used.
   - The corresponding public key is derived from the private key and returned.

2. **Signing**: Use the private key to sign a message.

   - The private_key.sign function is used to sign the message using a specified hash function (SHA-256 here).
   - This produces a signature that can later be verified.

3. **Verification**: Use the public key to verify the signature.

   - The public_key.verify function checks whether the signature matches the original message using the public key.
   - If the verification fails, an InvalidSignature exception is raised, indicating that the signature is not valid.

Python Code: DSA-based Digital Signature Implementation:

```
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.primitives import hashes, serialization
from cryptography.exceptions import InvalidSignature
import hashlib


# Function to generate RSA private and public keys
def generate_keys():
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048,
        backend=default_backend()
    )
    public_key = private_key.public_key()
```

```python
    return private_key, public_key


# Function to sign a message and print its hash
def sign_message(private_key, message):
    # Hash the message using SHA-256
    message_hash = hashlib.sha256(message.encode()).hexdigest()

    # Sign the hashed message using RSA private key
    signature = private_key.sign(
        bytes.fromhex(message_hash),  # Convert the hex hash to
bytes
        padding.PSS(
            mgf=padding.MGF1(hashes.SHA256()),
            salt_length=padding.PSS.MAX_LENGTH
        ),
        hashes.SHA256()
    )

    print(f"Hash of the message: {message_hash}")  # Print the
message hash
    return signature, message_hash


# Function to verify the signature using the provided hash
def verify_signature(public_key, message_hash, signature):
    try:
        # Verify the signature using RSA public key
        public_key.verify(
            signature,
            bytes.fromhex(message_hash),  # Convert the hex hash
back to bytes
            padding.PSS(
                mgf=padding.MGF1(hashes.SHA256()),
                salt_length=padding.PSS.MAX_LENGTH
            ),
            hashes.SHA256()
        )
        return True
    except InvalidSignature:
        return False


# Function to save keys to files
def save_keys_to_file(private_key, public_key):
```

```
    # Save private key
    with open("private_key.pem", "wb") as private_file:
        private_file.write(
            private_key.private_bytes(
                encoding=serialization.Encoding.PEM,
                format=serialization.PrivateFormat.PKCS8,
                encryption_algorithm=serialization.NoEncryption()
            )
        )
    # Save public key
    with open("public_key.pem", "wb") as public_file:
        public_file.write(
            public_key.public_bytes(
                encoding=serialization.Encoding.PEM,
                format=serialization.PublicFormat.SubjectPublicKeyIn
fo
            )
        )
    print("Keys saved to files: private_key.pem and public_key.pem")


# Menu for the digital signature system
def menu():
    private_key, public_key = None, None
    signature = None
    message_hash = None

    while True:
        print("\n===== Digital Signature System =====")
        print("1. Generate RSA Keys")
        print("2. Sign a Message")
        print("3. Verify Signature")
        print("4. Save Keys to Files")
        print("5. Exit")

        choice = input("Enter your choice (1/2/3/4/5): ")

        if choice == '1':
            # Generate RSA private and public keys
            private_key, public_key = generate_keys()
            print("\nRSA Keys Generated!")

        elif choice == '2':
            # Sign a message
```

```
            if private_key is None:
                print("You need to generate RSA keys first.")
            else:
                message = input("Enter the message to sign: ")
                signature, message_hash = sign_message(private_key,
message)
                print("\nMessage signed successfully!")

        elif choice == '3':
            # Verify the signature
            if public_key is None or message_hash is None or
signature is None:
                print("You need to sign a message first.")
            else:
                input_hash = input("Enter the hash of the message to
verify: ")
                verification_result = verify_signature(public_key,
input_hash, signature)
                if verification_result:
                    print("\nSignature verified successfully! The
message is authentic.")
                else:
                    print("\nSignature verification failed! The
message is not authentic.")

        elif choice == '4':
            # Save RSA keys to files
            if private_key is None or public_key is None:
                print("You need to generate RSA keys first.")
            else:
                save_keys_to_file(private_key, public_key)

        elif choice == '5':
            print("Exiting the program...")
            break

        else:
            print("Invalid choice. Please try again.")


if __name__ == "__main__":
    menu()
```

Output:

```
MINGW64:/c/Users/Onkar/Downloads

Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 10 (main)
$ python dss.py

===== Digital Signature System =====
1. Generate RSA Keys
2. Sign a Message
3. Verify Signature
4. Save Keys to Files
5. Exit
Enter your choice (1/2/3/4/5): 1

RSA Keys Generated!

===== Digital Signature System =====
1. Generate RSA Keys
2. Sign a Message
3. Verify Signature
4. Save Keys to Files
5. Exit
Enter your choice (1/2/3/4/5): 2
Enter the message to sign: Tomorrow's test is resheduled on 25th october, 2024.
Hash of the message: a17d3a1a76be18c39063139bc4d9df98aa9d707d4a1eae5c4cc2a512f53bc371

Message signed successfully!
```

```
===== Digital Signature System =====
1. Generate RSA Keys
2. Sign a Message
3. Verify Signature
4. Save Keys to Files
5. Exit
Enter your choice (1/2/3/4/5): 3
Enter the hash of the message to verify: a17d3a1a76be18c39063139bc4d9df98aa9d707d4a1eae5c4cc2a512f53bc371

Signature verified successfully! The message is authentic.

===== Digital Signature System =====
1. Generate RSA Keys
2. Sign a Message
3. Verify Signature
4. Save Keys to Files
5. Exit
Enter your choice (1/2/3/4/5): 3
Enter the hash of the message to verify: a17d3a1a76be18c39063139bc4d9df98aa9d707d4a1eae5c4cc2a512f53bc361

Signature verification failed! The message is not authentic.

===== Digital Signature System =====
1. Generate RSA Keys
2. Sign a Message
3. Verify Signature
4. Save Keys to Files
5. Exit
Enter your choice (1/2/3/4/5): 4
Keys saved to files: private_key.pem and public_key.pem

===== Digital Signature System =====
1. Generate RSA Keys
2. Sign a Message
3. Verify Signature
4. Save Keys to Files
5. Exit
Enter your choice (1/2/3/4/5): 5
Exiting the program...
```

dss.py | public_key.pem | private_key.pem

public_key.pem


```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAmtOyELOYrep3PXbsMBl4
qmho4cG3d5KbiB+KHuD24fON+pN1tXKAJgQ4Au80wmWfyhwe61jEWv+8fbdIdGMt
qeY0xwLa4LAhej1UUpP5/D6/2891jXuIJkb/yGh6sVSnvfqITGG5j7dG4Vepynes
fdkX3rskTgvPhWTxChY3CQsmRIaWBu05kzkMW7zajb6rF5C59SNRI0xaE3DzmMC6
Nas8RQPTIFw0Rhzp7H5ydtEvziwV3i8X1PbDc0m4jhsXP/hfgIXBSFVjZHZIHHmQ
1V8WY2xbAqC46p/oJ7r+29P7GyiIhPdoUfEr2vZ+1qvkVbe+RoRBeOxlcCzm0brE
DwIDAQAB
-----END PUBLIC KEY-----
```

dss.py | public_key.pem | private_key.pem

private_key.pem


```
-----BEGIN PRIVATE KEY-----
MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQCa07IQs5it6nc9
duwwGXiqaGjhwbd3kpuIH4oe4Pbh8436k3W1coAmBDgC7zTCZZ/KHB7rWMRa/7x9
t0h0Yy2p5jTHAtrgsCF6PVRSk/n8Pr/bz3WNe4gmRv/IaHqxVKe9+ohMYbmPt0bh
V6nKd6x92RfeuyROC8+FZPEKFjcJCyZEhpYG7TmTOQxbvNqNvqsXkLn1I1EjTFoT
cPOYwLo1qzxFA9MgXDRGHOnsfnJ20S/OLBXeLxfU9sNzSbiOGxc/+F+AhcFIVWNk
dkgceZDVXxZjbFsCoLjqn+gnuv7b0/sbKIiE92hR8Sva9n7Wq+RVt75GhEF47GVw
LObRusQPAgMBAAECggEAIXoVnfPfVveECde9cRcI1rvVtb63p1SdaZz8NfoJl8ic
UepxgKa927p8y/Gh9EiZ+M8Exnl+l3iFbp9jPLs3I+3hSois//lFe9Nn+KUDYA3H
WUVt0wY7ZZ9jbhddNBusW3Eg6HcW8uK7ojzcuykONHXs6d2IF5IJMHwnfDladAtN
f8V06Id7LyJeKtuSEOHgOuQ4Hb93bGNiO3dx/NdTMvoDpKf4xKnUXDyFoYvihB7u
MXgdP/QiVTt3rsjq2v0imaQn5ve8hTEnwvsuA4O5G8s9rfodkbAheNGpEh8rVvXF
mgobxfKz+8a1CJL9FLEgBlk9LCTU3AnuOTqlqTTXzQKBgQDL6mJ4B/MYkEjyEdPl
WFbxCuvdHM94/dZrP+Tt8RnnByWtC2cvBcK1SU7MtDhV/ChwhPP2ueM8AbJFf2cI
eKZHDfbHWNhVKHnwKviUwGzfArlt6EfsiPYyeZpiSWCeWdNOS5fm5kUboQmNU9+X
jtE0nhCyJfdh5w8dDYMVNvs0rQKBgQDCX4Hkmf3mnK0S0IESigRWFkhwPBEWpbHK
m3oTG9FHUhbXd3nNfW9t/IuZP6dGx6WpTUdIqiEOTaFQqpv1RKKSc2iRbFxX/dPK
g/xp3WL+A75M210qgbFVQi8RJmdbnmQP98HhvjsH85BR+B37fjEwqWvIT4VikQcz
EKNs7U73KwKBgQDJaSzxmgiHDpy+XNWLBoJ5e2wU7kwPGcocDPQ2AZqwEuMn1MeK
LopvPYVTs/6hD3tyCCBgZqMHtjU7Z+eA+opiTGyf4iVrl5s5mXgGlTnz3GEDKhdA
jtd+a5YN1qRURCzufMQBERjZfWpN6bZDoJELA8VB8TYzwWGwNJRw8+mT0QKBgQCR
xYrm5M5UK2BpYdHLE5MY9PEyqvmt4GosJtowhoY3VMsooMPKZq4w8FvJfMF5BbWp
+Pji+CR+U54Nn0YDdYvvBXqAxaGjpAs4MDAXPR9GnYwUm9eNT2KtLEucMw8E2Q72
IrtXD5G57uJndwsVV5iQhB7/RRQlzdp2Y54MzRSRrQKBgApuWUInk3T6u5GBrfuJ
08MFrDTQt48EUldhL/h+i3oya2Z/nuC9R0m6ubyjeRxMwZ12Fk8GESEIaBqjbWTY
JJQmQu/fHZHRakpLSOYkHlMqvJLY5Fn0f8nXAJqlp5R8MivzclsLG5ooxfpoCw5U
EsD+eMo57vITF1H5cNYVaWRi
-----END PRIVATE KEY-----
```


Virtual labs:

Virtual Labs

An MoE Govt of India Initiative

Digitally sign the plaintext with Hashed RSA.

Plaintext (string):

Ex. Minister of Maharashtra got assassinated yest | SHA-1

Hash output(hex):

041d610065f6a89ecd2059032221ec4e6e2e42f0

Input to RSA(hex):

041d610065f6a89ecd2059032221ec4e6e2e42f0 Apply RSA

Digital Signature(hex):

Digital Signature(base64):

Status:

RSA public key

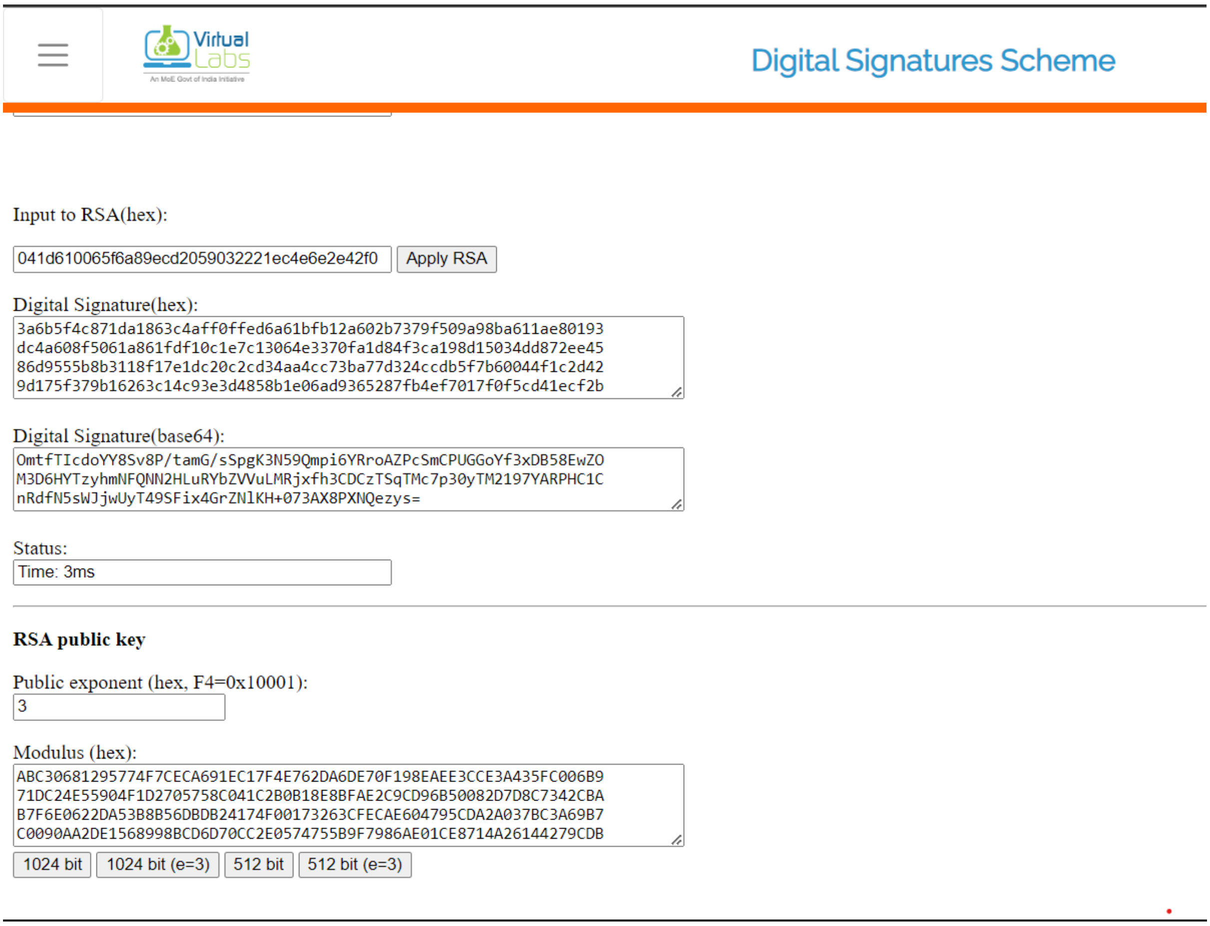
Public exponent (hex, F4=0x10001):

3

Modulus (hex):

```
ABC30681295774F7CECA691EC17F4E762DA6DE70F198EAEE3CCE3A435FC006B9
71DC24E55904F1D2705758C041C2B0B18E8BFAE2C9CD96B50082D7D8C7342CBA
B7F6E0622DA53B8B56DBDB24174F00173263CFECAE604795CDA2A037BC3A69B7
C0090AA2DE1568998BCD6D70CC2E0574755B9F7986AE01CE8714A26144279CDB
```

1024 bit | 1024 bit (e=3) | 512 bit | 512 bit (e=3)



How It Works:

- The **private key** signs the message, creating a unique signature.
- The **public key** is used by anyone to verify that the message has not been tampered with and that it was signed by the owner of the private key.

Practical Applications:

- **Digital Signatures** ensure the authenticity and integrity of a message or document.
- **Message Authentication**: In secure communication, the sender can sign the message, and the receiver can verify the signature to ensure the message is authentic.

Advantages of DSS (Digital Signature Standard, based on DSA):

- **Authentication**: Ensures the identity of the sender, as only the sender's private key can create a valid signature.
- **Integrity**: Guarantees that the message hasn't been altered, as any change would invalidate the signature.
- **Non-repudiation**: The sender cannot deny having signed the message, as the signature is unique to the private key.
- **Efficiency**: DSA is optimized for creating signatures, making it relatively fast for signing compared to some other algorithms.
- **Security**: Provides a high level of security, especially with modern key sizes (2048 bits or more).

<u>Disadvantages of DSS:</u>

- **Slower Verification**: DSA is slower at verifying signatures compared to some alternatives like RSA, making it less suitable for environments that require frequent verification.
- **Key Management**: Requires secure management of private keys; if the private key is compromised, the entire security system is at risk.
- **Message Size Limitations**: DSA only signs the hash of a message, so very large messages require hashing before signature generation.
- **Limited Use Cases**: DSA is primarily designed for digital signatures and not for encryption, unlike algorithms like RSA.

<u>Importance of DSS:</u>

- **Government Standard**: DSS (and DSA) is an official standard used by governments and organizations worldwide for secure digital signatures.
- **Legal Recognition**: Digital signatures created using DSS are often legally recognized, making them suitable for contracts and other legal documents.

- **Widely Used in Secure Communications**: DSS is crucial in protocols like SSL/TLS, ensuring secure web communications, digital certificates, and more.

In summary, DSS is important for ensuring the authenticity, integrity, and non-repudiation of digital communications, despite some performance-related limitations.

Final Year B.Tech. (CSE) – VII [2024-25]

6CS451: Cryptography and Network Security Lab (C&NS Lab)

Date: 15/10/2024

Assignment 11

PRN: 21510017 **Name:** Onkar Anand Yemul

---

11. Demonstration of SSL using Wireshark.

Ans:

SSL (Secure Sockets Layer) and **TLS (Transport Layer Security)** are cryptographic protocols designed to secure communication over a network, especially the internet. They ensure that data transmitted between a client (e.g., web browser) and a server (e.g., website) is encrypted, authenticated, and tamper-proof.

TLS is the more recent and secure version of SSL. SSL is now considered obsolete, and TLS is widely used today (referred to as "SSL/TLS" in many contexts).

<u>SSL vs TLS:</u>

- **SSL**: Older protocol, now considered insecure due to vulnerabilities like **POODLE** and **BEAST**.
- **TLS**: Successor to SSL, currently in use with versions like **TLS 1.2** and **TLS 1.3**. TLS 1.3 is the most secure and efficient version available today.

<u>Goals of SSL/TLS:</u>

1. **Confidentiality**: Encrypts data to prevent unauthorized access.
2. **Integrity**: Ensures data has not been altered during transmission.

3. **Authentication**: Verifies the identity of the server, and optionally the client.

How SSL/TLS Works:

SSL/TLS secures communication through a series of steps that establish a **secure connection** before data exchange. These steps are known as the **SSL/TLS handshake**. The handshake is a process where the client and server agree on encryption methods and share keys for secure communication.

Steps Involved in SSL/TLS Handshake:

1. **Client Hello**:
    - The client (e.g., a web browser) sends a **Client Hello** message to the server.
    - The message includes:
        - SSL/TLS version the client supports.
        - A list of supported **cipher suites** (encryption algorithms).
        - A randomly generated value called the **Client Random.**
        - Optional information like session resumption data.
2. **Server Hello**:
    - The server responds with a **Server Hello** message.
    - It selects the SSL/TLS version and cipher suite from the list provided by the client.
    - It also generates and sends a **Server Random** value.
3. **Server Certificate**:
    - The server sends its **SSL/TLS certificate** to the client.
    - This certificate contains the server's **public key** and is signed by a trusted **Certificate Authority (CA)**.
    - The client uses this certificate to verify the server’s identity.

4. **Key Exchange**:
    - The client and server exchange information that allows both parties to generate a shared **session key.**
    - The session key is used to encrypt data exchanged during the session.
    - The key exchange can use algorithms like **RSA** (Rivest-Shamir-Adleman) or **Diffie-Hellman.**
5. **Client Key Exchange**:
    - The client generates a **Pre-master Secret** (a random number) and encrypts it using the server's public key from the certificate.
    - This pre-master secret is sent to the server.
6. **Session Key Generation**:
    - Both the client and the server use the pre-master secret, along with the Client Random and Server Random values, to independently generate the same **session key.**
    - The session key is a symmetric key used for encrypting and decrypting the communication during the session.
7. **Change Cipher Spec**:
    - The client and server send a **Change Cipher Spec** message to inform each other that future messages will be encrypted using the session key.
8. **Finished**:
    - Both the client and the server send a **Finished** message, encrypted with the session key, to indicate that the handshake is complete.
    - If both messages are successfully decrypted and verified, the SSL/TLS connection is established.
9. **Encrypted Communication**:

- After the handshake, the client and server use the session key to encrypt and decrypt all subsequent data exchanged during the session.
- This ensures that sensitive data, such as login credentials, is transmitted securely.

<u>Summary of the SSL/TLS Handshake:</u>

1. **Client Hello** (Client proposes SSL/TLS version, cipher suite, and sends a random number)
2. **Server Hello** (Server selects version, cipher suite, and sends a random number)
3. **Server Certificate** (Server sends its certificate for client to verify)
4. **Key Exchange** (Client and server exchange information to generate a shared session key)
5. **Change Cipher Spec** (Both agree to switch to encrypted communication)
    1. **Finished** (Handshake is complete, and encrypted communication begins)

Step	Client	Direction	Message	Direction	Server
1			Client Hello	→	
2		←	Server Hello		
3		←	Certificate		
4		←	Server Key Exchange		
5		←	Server Hello Done		
6			Client Key Exchange	→	
7			Change Cipher Spec	→	
8			Finished	→	
9		←	Change Cipher Spec		
10		←	Finished		

<u>Importance of SSL/TLS:</u>

- **Web Security**: SSL/TLS ensures that sensitive information (such as passwords, credit card details) is transmitted securely over the internet.
- **Authentication**: Verifies the identity of the website to prevent man-in-the-middle (MITM) attacks.
- **Data Privacy**: Prevents eavesdropping and tampering during communication.

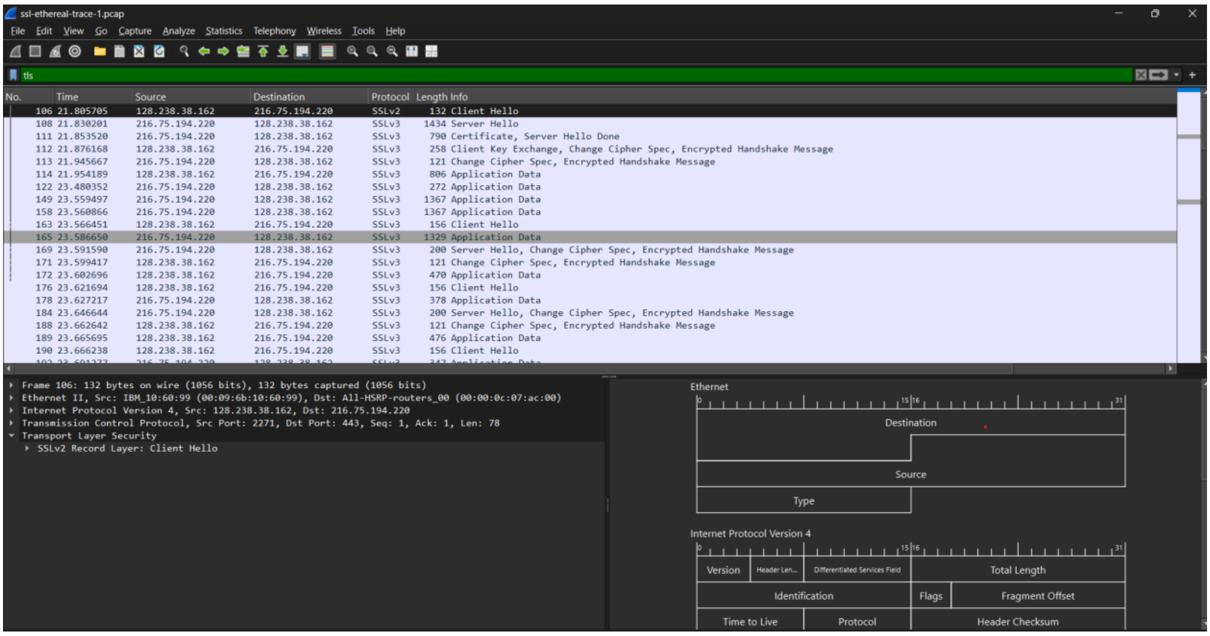
In conclusion, SSL/TLS is crucial for ensuring secure and trusted communication over networks, particularly for websites and online services.

Wireshark Lab: SSL v8.0

Download the zip file http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip and extract the ***ssl-ethereal trace-1*** packet trace.

1. Capturing packets in an SSL session

It displays only the Ethernet frames that contain SSL records sent from and received by your host



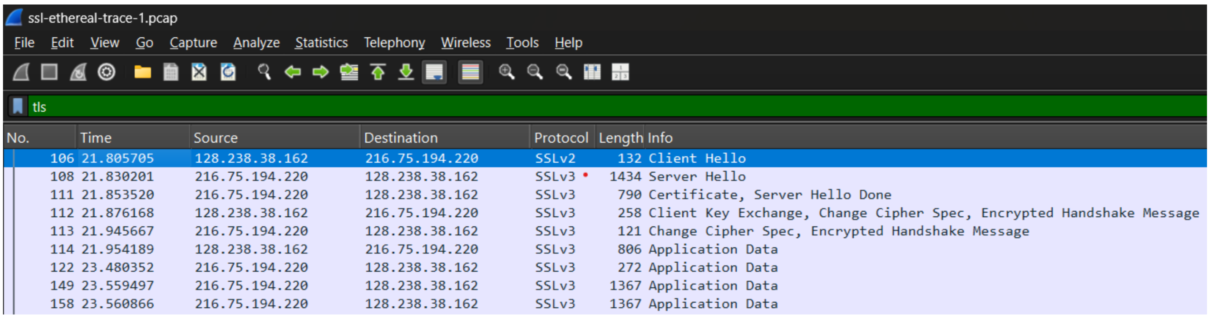
An SSL record is the same thing as an SSL message.

2. A look at the captured trace

An Ethernet frame may contain one or more SSL records. (This is very different from HTTP, for which each frame contains either one complete HTTP message or a portion of a HTTP message.) Also, an SSL record may not completely fit into an Ethernet frame, in which case multiple frames will be needed to carry the record.

1. For each of the first 8 Ethernet frames, specify the source of the frame (client or server), determine the number of SSL records that are included in the frame, and list the SSL record types that are included in the frame. Draw a timing diagram between client and server, with one arrow for each SSL record.

<u>Ans:</u>



No.	Time	Source	Destination	Protocol	Length	Info
106	21.805705	128.238.38.162	216.75.194.220	SSLv2	132	Client Hello
108	21.830201	216.75.194.220	128.238.38.162	SSLv3	1434	Server Hello
111	21.853520	216.75.194.220	128.238.38.162	SSLv3	790	Certificate, Server Hello Done
112	21.876168	128.238.38.162	216.75.194.220	SSLv3	258	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
113	21.945667	216.75.194.220	128.238.38.162	SSLv3	121	Change Cipher Spec, Encrypted Handshake Message
114	21.954189	128.238.38.162	216.75.194.220	SSLv3	806	Application Data
122	23.480352	216.75.194.220	128.238.38.162	SSLv3	272	Application Data
149	23.559497	216.75.194.220	128.238.38.162	SSLv3	1367	Application Data
158	23.560866	216.75.194.220	128.238.38.162	SSLv3	1367	Application Data

Frame 1: (Frame 106)

- Source: Client (128.238.38.162)
- Number of SSL Records: 1
- SSL Record Type: Client Hello (SSLv2)

Frame 2: (Frame 108)

- Source: Server (216.75.194.220)
- Number of SSL Records: 1
- SSL Record Type: Server Hello (SSLv3)

Frame 3: (Frame 111)

- Source: Server (216.75.194.220)
- Number of SSL Records: 2
- SSL Record Types:
    1. Certificate (SSLv3)
    2. Server Hello Done (SSLv3)

Frame 4: (Frame 112)

- Source: Client (128.238.38.162)
- Number of SSL Records: 3
- SSL Record Types:
    1. Client Key Exchange (SSLv3)
    2. Change Cipher Spec (SSLv3)
    3. Encrypted Handshake Message (SSLv3)

Frame 5: (Frame 113)

- Source: Server (216.75.194.220)
- Number of SSL Records: 2
- SSL Record Types:
    1. Change Cipher Spec (SSLv3)
    2. Encrypted Handshake Message (SSLv3)

Frame 6: (Frame 114)

- Source: Client (128.238.38.162)
- Number of SSL Records: 1
- SSL Record Type: Application Data (SSLv3)

Frame 7: (Frame 122)

- Source: Server (216.75.194.220)
- Number of SSL Records: 1
- SSL Record Type: Application Data (SSLv3)

Frame 8: (Frame 149)

- Source: Server (216.75.194.220)
- Number of SSL Records: 1
- SSL Record Type: Application Data (SSLv3)

2. Each of the SSL records begins with the same three fields (with possibly different values). One of these fields is “content type” and has length of one byte. List all three fields and their lengths.

Ans:

Each SSL record starts with the following three fields:

- **Content Type**: 1 byte
- **Version**: 2 bytes
- **Length**: 2 bytes

How to Find These Fields:

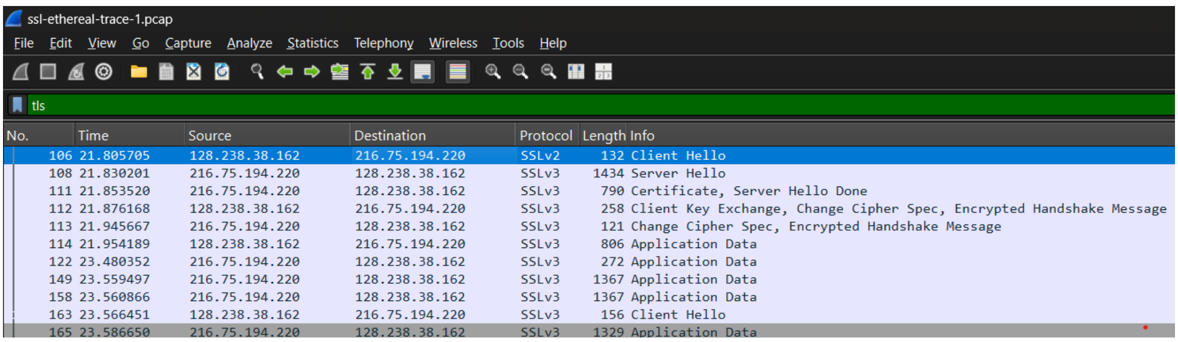
If you are using packet capture software like **Wireshark**, you can find these fields in the packet capture by:

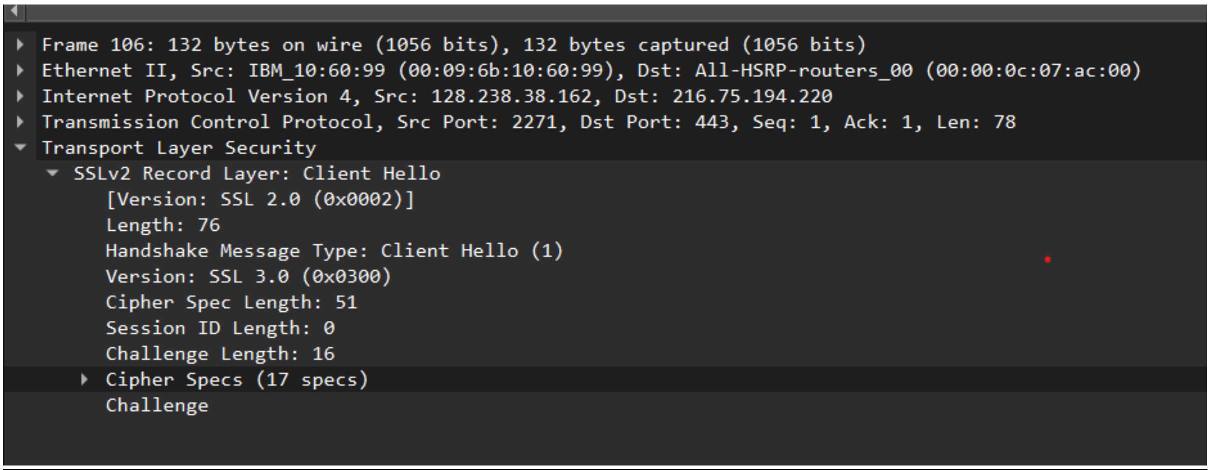
1. **Open Wireshark** and load the captured SSL/TLS packet data (the one you listed).

2. **Select an SSL/TLS packet** from the list and expand the **"Secure Sockets Layer"** or **"Transport Layer Security"** section in the detailed packet view.

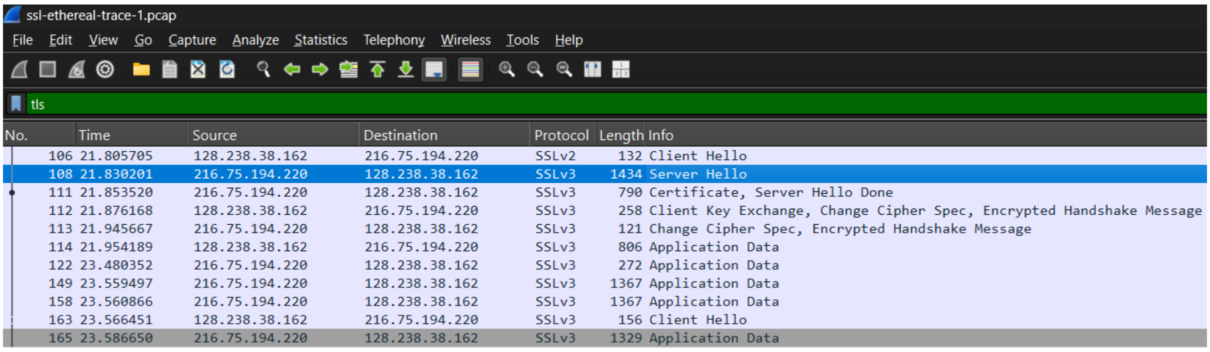
3. You will see the **Record Layer** header information, where these fields will be listed:
    - **Content Type**: Displays the type of SSL/TLS record (Handshake, Application Data, etc.)
    - **Version**: The protocol version (e.g., TLS 1.2)
    - **Length**: The size of the encrypted data.

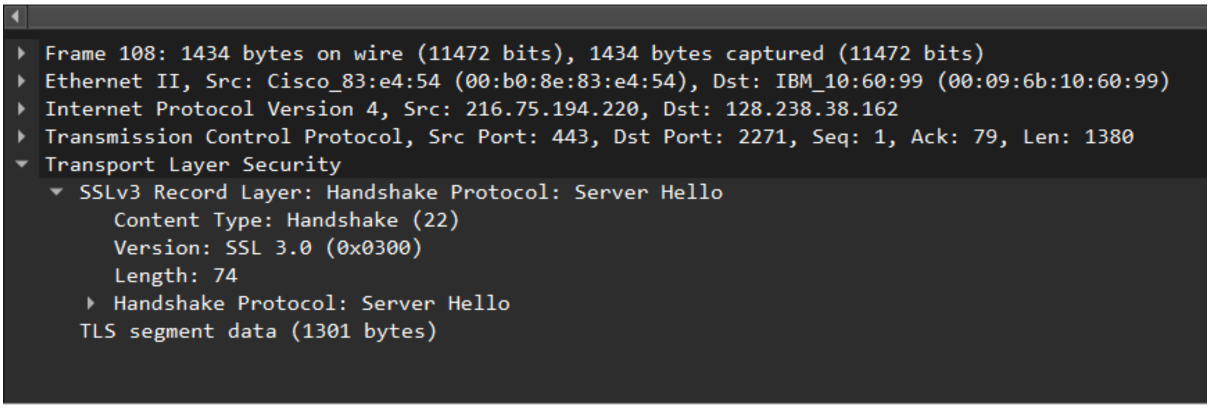
Selecting Client Hello packet:





Selecting Server Hello packet:



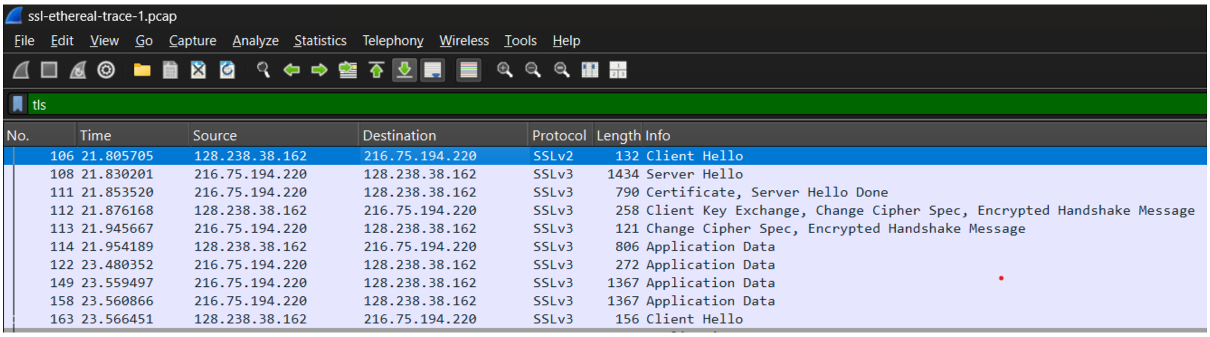


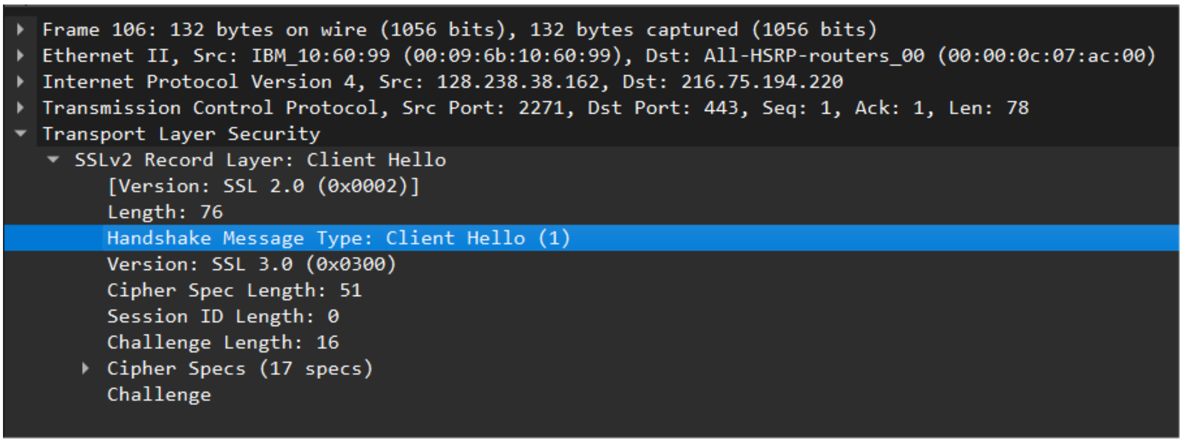
ClientHello Record:

3. Expand the ClientHello record. (If your trace contains multiple ClientHello records, expand the frame that contains the first one.) What is the value of the content type?

Ans:

The **ClientHello** record in **Frame 106** is an SSLv2 message with a handshake message type of **Client Hello (1)**.

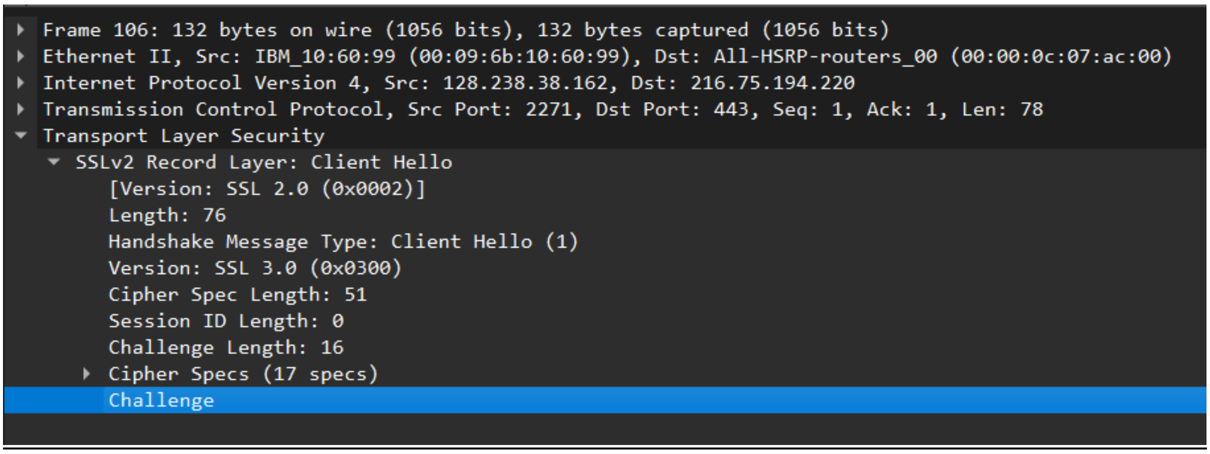


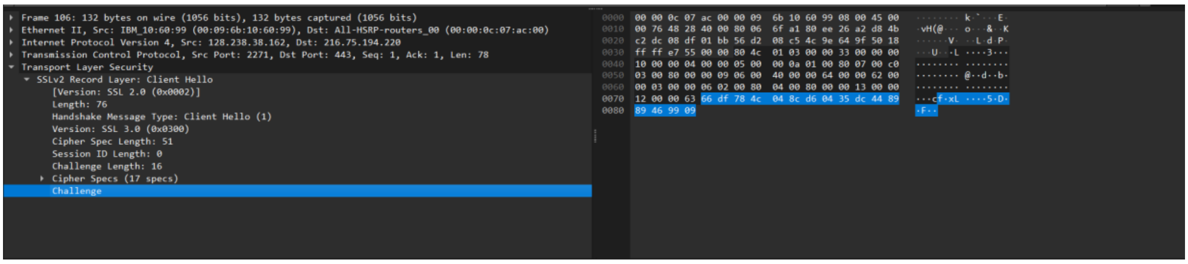


4. Does the ClientHello record contain a nonce (also known as a "challenge")? If so, what is the value of the challenge in hexadecimal notation?

Ans:

Yes, the ClientHello record contains a nonce (also known as a "challenge").

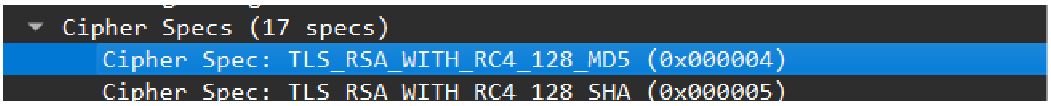




5. Does the ClientHello record advertise the cyber suites it supports? If so, in the first listed suite, what are the public-key algorithm, the symmetric-key algorithm, and the hash algorithm?

Ans:

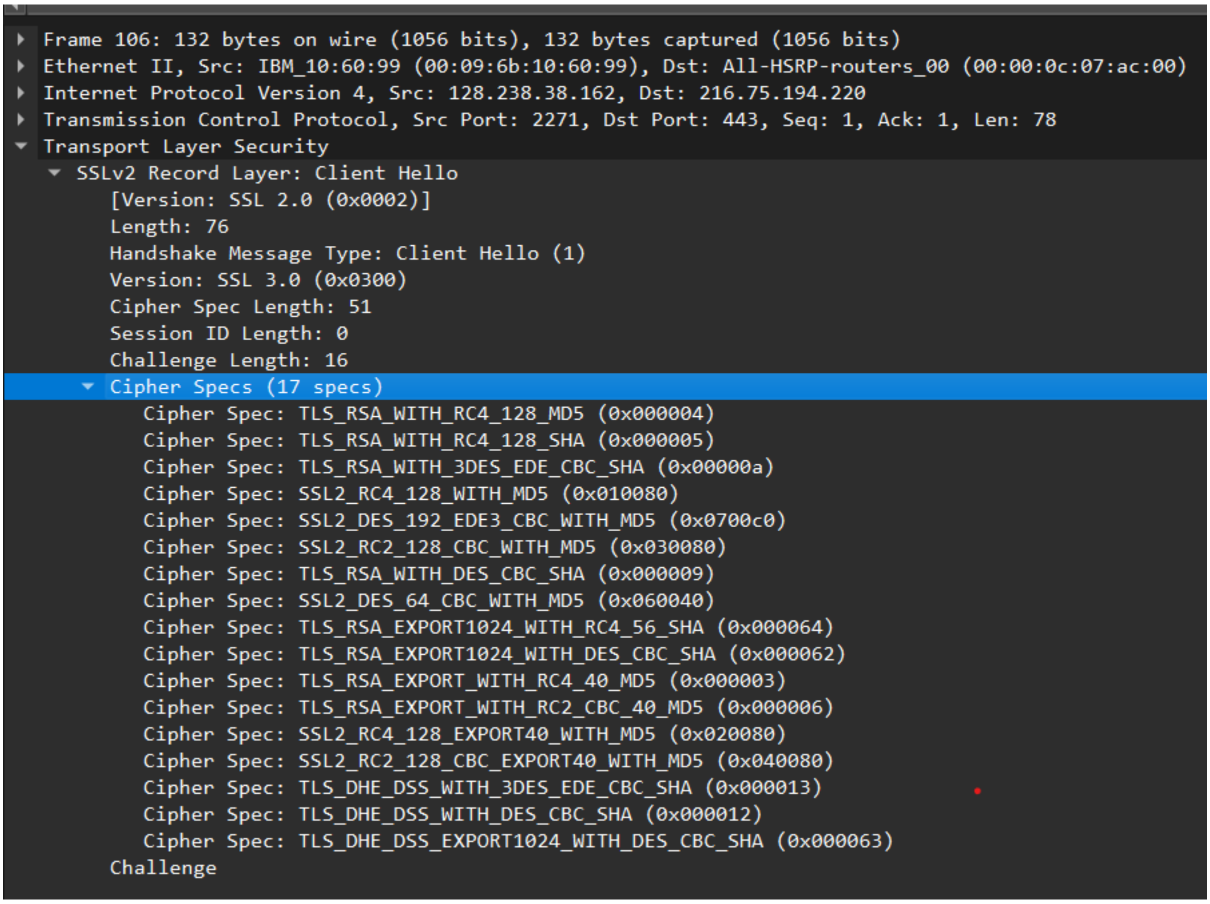
Yes, the ClientHello record does advertise the cipher suites it supports.



In the first listed cipher suite, which is **TLS_RSA_WITH_RC4_128_MD5 (0x000004)**, the following algorithms are used:

- **Public-key algorithm**: RSA
- **Symmetric-key algorithm**: RC4 (with a key length of 128 bits)
- **Hash algorithm**: MD5

This combination indicates that the client supports this suite for secure communication.

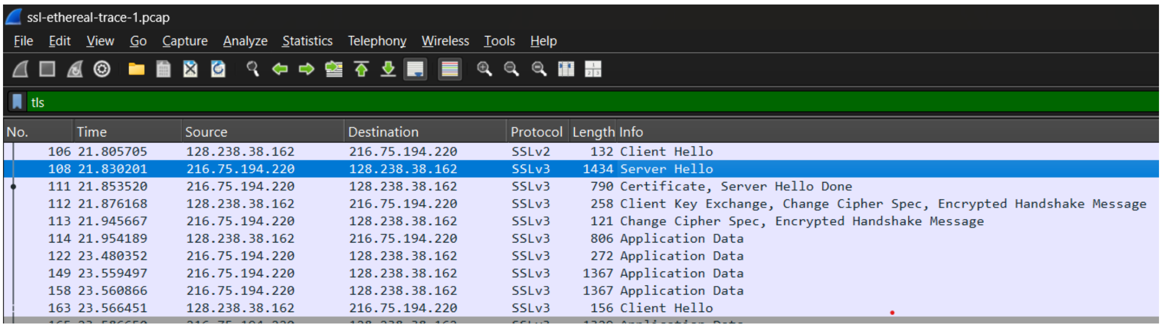


```
Frame 106: 132 bytes on wire (1056 bits), 132 bytes captured (1056 bits)
Ethernet II, Src: IBM_10:60:99 (00:09:6b:10:60:99), Dst: All-HSRP-routers_00 (00:00:0c:07:ac:00)
Internet Protocol Version 4, Src: 128.238.38.162, Dst: 216.75.194.220
Transmission Control Protocol, Src Port: 2271, Dst Port: 443, Seq: 1, Ack: 1, Len: 78
Transport Layer Security
  SSLv2 Record Layer: Client Hello
    [Version: SSL 2.0 (0x0002)]
    Length: 76
    Handshake Message Type: Client Hello (1)
    Version: SSL 3.0 (0x0300)
    Cipher Spec Length: 51
    Session ID Length: 0
    Challenge Length: 16
    Cipher Specs (17 specs)
      Cipher Spec: TLS_RSA_WITH_RC4_128_MD5 (0x000004)
      Cipher Spec: TLS_RSA_WITH_RC4_128_SHA (0x000005)
      Cipher Spec: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x00000a)
      Cipher Spec: SSL2_RC4_128_WITH_MD5 (0x010080)
      Cipher Spec: SSL2_DES_192_EDE3_CBC_WITH_MD5 (0x0700c0)
      Cipher Spec: SSL2_RC2_128_CBC_WITH_MD5 (0x030080)
      Cipher Spec: TLS_RSA_WITH_DES_CBC_SHA (0x000009)
      Cipher Spec: SSL2_DES_64_CBC_WITH_MD5 (0x060040)
      Cipher Spec: TLS_RSA_EXPORT1024_WITH_RC4_56_SHA (0x000064)
      Cipher Spec: TLS_RSA_EXPORT1024_WITH_DES_CBC_SHA (0x000062)
      Cipher Spec: TLS_RSA_EXPORT_WITH_RC4_40_MD5 (0x000003)
      Cipher Spec: TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5 (0x000006)
      Cipher Spec: SSL2_RC4_128_EXPORT40_WITH_MD5 (0x020080)
      Cipher Spec: SSL2_RC2_128_CBC_EXPORT40_WITH_MD5 (0x040080)
      Cipher Spec: TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA (0x000013)
      Cipher Spec: TLS_DHE_DSS_WITH_DES_CBC_SHA (0x000012)
      Cipher Spec: TLS_DHE_DSS_EXPORT1024_WITH_DES_CBC_SHA (0x000063)
    Challenge
```

ServerHello Record:

6. Locate the ServerHello SSL record. Does this record specify a chosen cipher suite? What are the algorithms in the chosen cipher suite?

Ans:



No.	Time	Source	Destination	Protocol	Length	Info
106	21.805705	128.238.38.162	216.75.194.220	SSLv2	132	Client Hello
108	21.830201	216.75.194.220	128.238.38.162	SSLv3	1434	Server Hello
111	21.853520	216.75.194.220	128.238.38.162	SSLv3	790	Certificate, Server Hello Done
112	21.876168	128.238.38.162	216.75.194.220	SSLv3	258	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
113	21.945667	216.75.194.220	128.238.38.162	SSLv3	121	Change Cipher Spec, Encrypted Handshake Message
114	21.954189	128.238.38.162	216.75.194.220	SSLv3	806	Application Data
122	23.480352	216.75.194.220	128.238.38.162	SSLv3	272	Application Data
149	23.559497	216.75.194.220	128.238.38.162	SSLv3	1367	Application Data
158	23.560866	216.75.194.220	128.238.38.162	SSLv3	1367	Application Data
163	23.566451	128.238.38.162	216.75.194.220	SSLv3	156	Client Hello

```
> Frame 108: 1434 bytes on wire (11472 bits), 1434 bytes captured (11472 bits)
> Ethernet II, Src: Cisco_83:e4:54 (00:b0:8e:83:e4:54), Dst: IBM_10:60:99 (00:09:6b:10:60:99)
> Internet Protocol Version 4, Src: 216.75.194.220, Dst: 128.238.38.162
> Transmission Control Protocol, Src Port: 443, Dst Port: 2271, Seq: 1, Ack: 79, Len: 1380
v Transport Layer Security
  v SSLv3 Record Layer: Handshake Protocol: Server Hello
      Content Type: Handshake (22)
      Version: SSL 3.0 (0x0300)
      Length: 74
    v Handshake Protocol: Server Hello
        Handshake Type: Server Hello (2)
        Length: 70
        Version: SSL 3.0 (0x0300)
      > Random: 0000000042dbed248b8831d04cc98c26e5badc4e267c391944f0f070ece57745
        Session ID Length: 32
        Session ID: 1bad05faba02ea92c64c54be4547c32f3e3ca63d3a0c86ddad694b45682da22f
        Cipher Suite: TLS_RSA_WITH_RC4_128_MD5 (0x0004)
        Compression Method: null (0)
        [JA3S Fullstring: 768,4,]
        [JA3S: 1f8f5a3d2fd435e36084db890693eafd]
    TLS segment data (1301 bytes)
```

Yes, the ServerHello SSL record specifies a chosen cipher suite. The chosen cipher suite is TLS_RSA_WITH_RC4_128_MD5 (0x0004).

The algorithms in this chosen cipher suite are:

- **Public-key algorithm**: RSA
- **Symmetric-key algorithm**: RC4 (with a key length of 128 bits)
- **Hash algorithm**: MD5

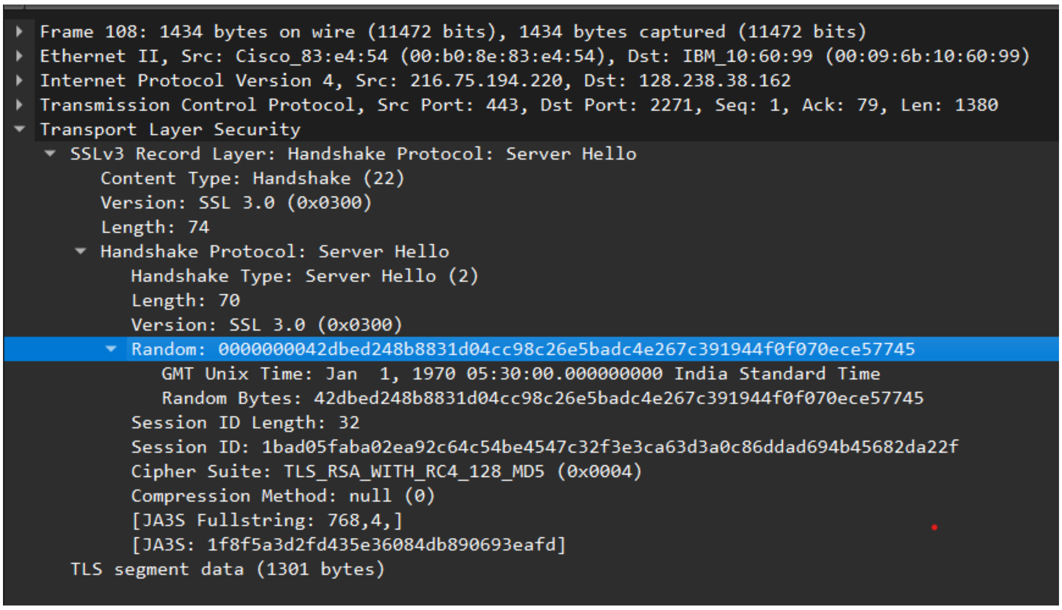
This indicates the server's selected encryption method for the session.

7. Does this record include a nonce? If so, how long is it? What is the purpose of the client and server nonces in SSL?

<u>**Ans:**</u>

Locate the Nonce:

- The **ServerHello** response may not explicitly list a nonce like the **ClientHello** does, but it usually includes a **Session ID** and potentially a **Server Random** value (which acts similarly to a nonce).
- Look for fields labeled **Session ID Length**, **Session ID**, and **Random**.



```
Frame 108: 1434 bytes on wire (11472 bits), 1434 bytes captured (11472 bits)
Ethernet II, Src: Cisco_83:e4:54 (00:b0:8e:83:e4:54), Dst: IBM_10:60:99 (00:09:6b:10:60:99)
Internet Protocol Version 4, Src: 216.75.194.220, Dst: 128.238.38.162
Transmission Control Protocol, Src Port: 443, Dst Port: 2271, Seq: 1, Ack: 79, Len: 1380
Transport Layer Security
  SSLv3 Record Layer: Handshake Protocol: Server Hello
    Content Type: Handshake (22)
    Version: SSL 3.0 (0x0300)
    Length: 74
    Handshake Protocol: Server Hello
      Handshake Type: Server Hello (2)
      Length: 70
      Version: SSL 3.0 (0x0300)
      Random: 0000000042dbed248b8831d04cc98c26e5badc4e267c391944f0f070ece57745
        GMT Unix Time: Jan  1, 1970 05:30:00.000000000 India Standard Time
        Random Bytes: 42dbed248b8831d04cc98c26e5badc4e267c391944f0f070ece57745
      Session ID Length: 32
      Session ID: 1bad05faba02ea92c64c54be4547c32f3e3ca63d3a0c86ddad694b45682da22f
      Cipher Suite: TLS_RSA_WITH_RC4_128_MD5 (0x0004)
      Compression Method: null (0)
      [JA3S Fullstring: 768,4,]
      [JA3S: 1f8f5a3d2fd435e36084db890693eafd]
  TLS segment data (1301 bytes)
```

Yes, the ServerHello record includes a nonce, which is referred to as the "Random" value. In this case, the nonce is 32 bytes long (256 bits).

Purpose of Nonces in SSL:

1. **Prevent Replay Attacks**: Nonces ensure that each session is unique, preventing attackers from reusing old messages to impersonate a user or a session.
2. **Key Generation**: Nonces are used in the key generation process during the SSL handshake. They contribute to creating session keys that are unique for each session, ensuring that even if the same keys were used in different sessions, the actual keys derived will be different due to the unique nonces.

   By using nonces, SSL enhances the security and integrity of the communication between the client and server.

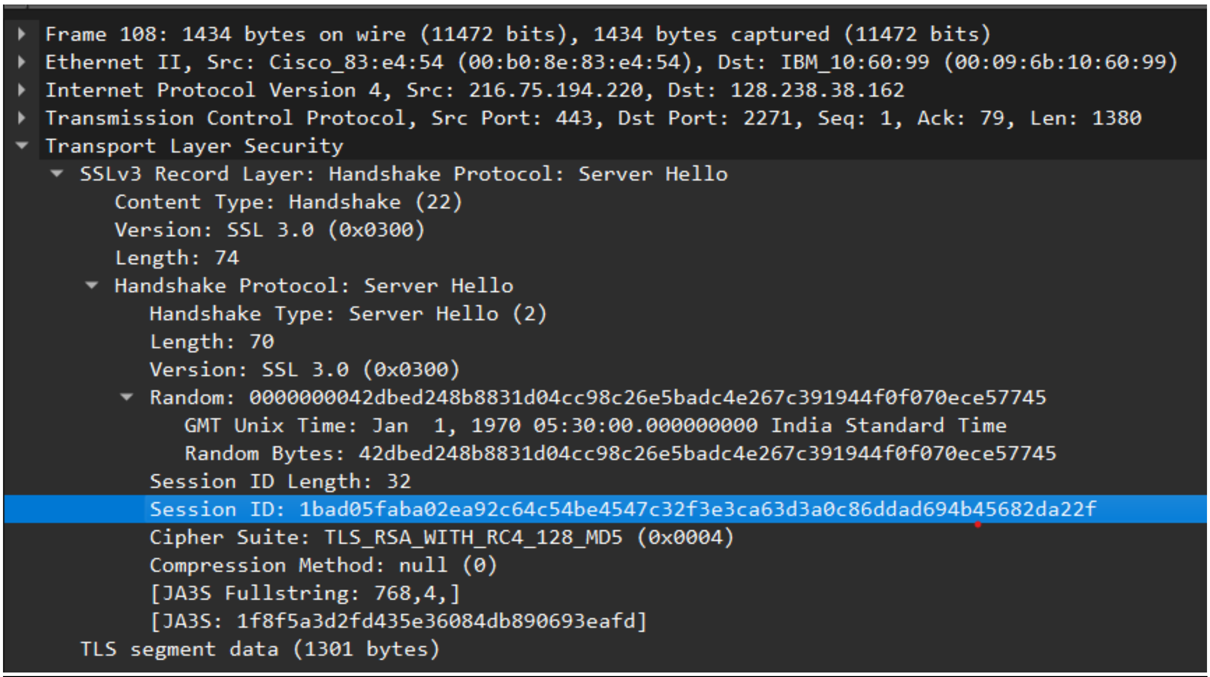
Purpose of Nonce in the ServerHello Record:

1. **Session Uniqueness**:
   - Similar to the **ClientHello**, the **Server Random** value helps ensure that the session is unique. It differentiates this session from previous ones.

2. **Key Derivation**:
   - The **Server Random** value is combined with the **Client Random** value (from the **ClientHello**) during the key derivation process to create session keys for encrypting the data exchanged in the session. This ensures that the keys are unique for each session.
3. **Preventing Replay Attacks**:
   - Just as with the client, the server's nonce (or **Server Random**) helps protect against replay attacks, ensuring that each session is independent and cannot be reused maliciously.

8. Does this record include a session ID? What is the purpose of the session ID?

Ans:


```
Frame 108: 1434 bytes on wire (11472 bits), 1434 bytes captured (11472 bits)
Ethernet II, Src: Cisco_83:e4:54 (00:b0:8e:83:e4:54), Dst: IBM_10:60:99 (00:09:6b:10:60:99)
Internet Protocol Version 4, Src: 216.75.194.220, Dst: 128.238.38.162
Transmission Control Protocol, Src Port: 443, Dst Port: 2271, Seq: 1, Ack: 79, Len: 1380
Transport Layer Security
  SSLv3 Record Layer: Handshake Protocol: Server Hello
    Content Type: Handshake (22)
    Version: SSL 3.0 (0x0300)
    Length: 74
    Handshake Protocol: Server Hello
      Handshake Type: Server Hello (2)
      Length: 70
      Version: SSL 3.0 (0x0300)
      Random: 0000000042dbed248b8831d04cc98c26e5badc4e267c391944f0f070ece57745
        GMT Unix Time: Jan  1, 1970 05:30:00.000000000 India Standard Time
        Random Bytes: 42dbed248b8831d04cc98c26e5badc4e267c391944f0f070ece57745
      Session ID Length: 32
      Session ID: 1bad05faba02ea92c64c54be4547c32f3e3ca63d3a0c86ddad694b45682da22f
      Cipher Suite: TLS_RSA_WITH_RC4_128_MD5 (0x0004)
      Compression Method: null (0)
      [JA3S Fullstring: 768,4,]
      [JA3S: 1f8f5a3d2fd435e36084db890693eafd]
  TLS segment data (1301 bytes)
```

Yes, the ServerHello record includes a session ID, which has a length of 32 bytes in this case.

Purpose of the Session ID:

1. **Session Resumption**: The session ID allows clients and servers to resume a previous SSL/TLS session without needing to perform a full handshake again. This speeds up the connection process for subsequent sessions between the same client and server.
2. **State Management**: The session ID helps manage session states on the server. It allows the server to recognize and retrieve session parameters (like the cipher suite and keys) associated with that ID, facilitating quicker reconnections.
3. **Security**: By using a session ID, SSL/TLS can provide continuity and consistency for secure communications, ensuring that established session parameters are reused securely.

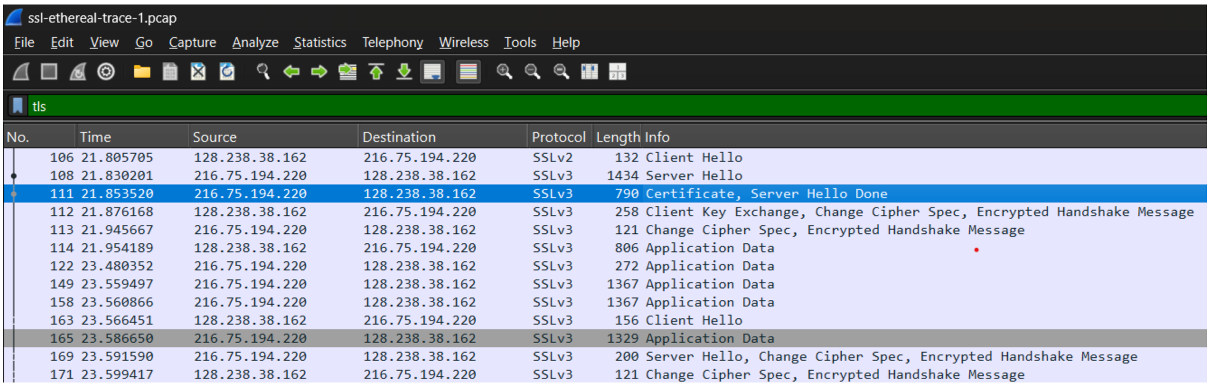
Overall, the session ID is crucial for improving performance and maintaining security during repeated connections.

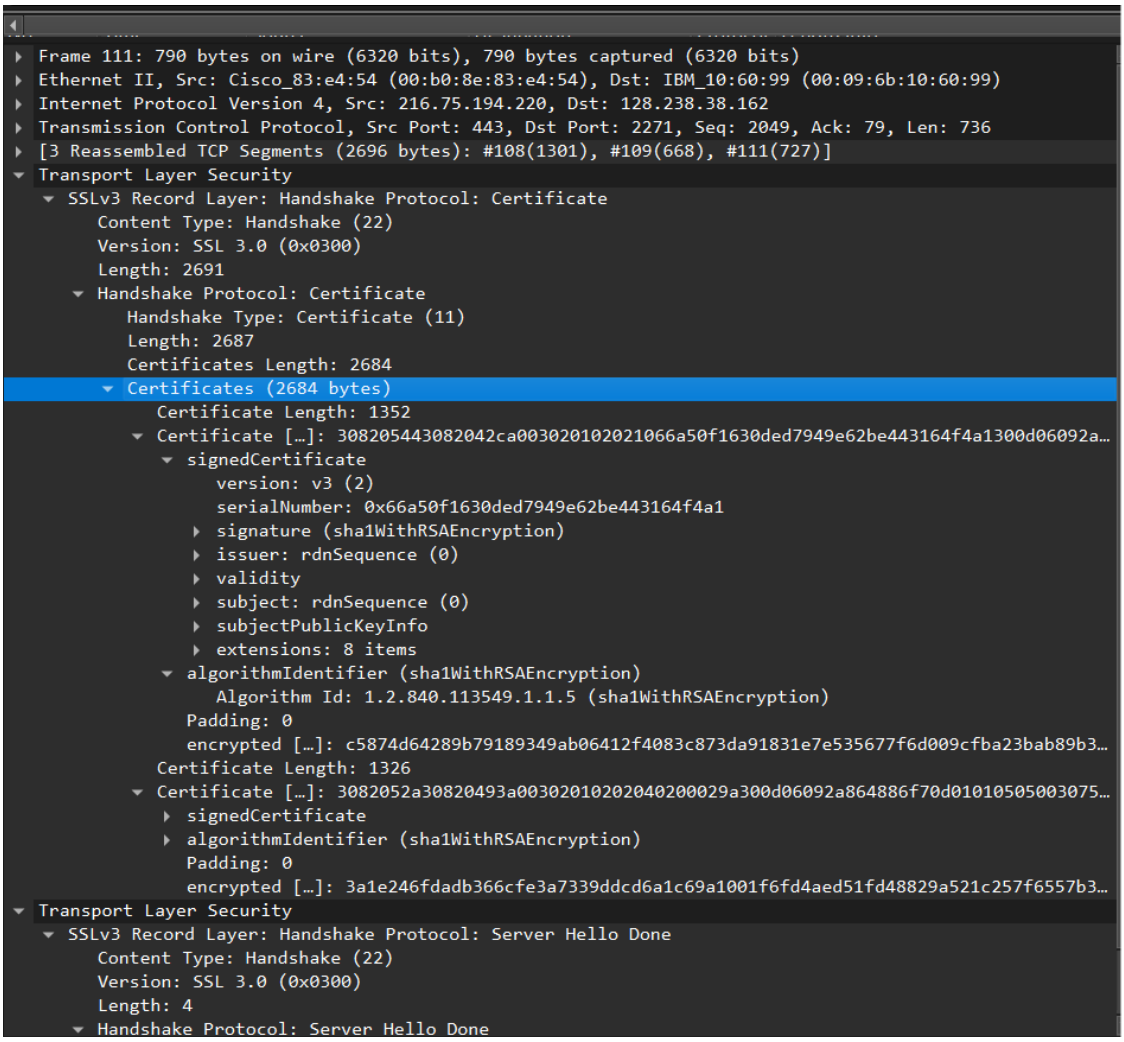
9. Does this record contain a certificate, or is the certificate included in a separate record. Does the certificate fit into a single Ethernet frame?

Ans:

The ServerHello record itself does not contain a certificate; certificates are included in a separate record, specifically in the Certificate message that follows the ServerHello during the handshake process.

Regarding the size of the certificate, whether it fits into a single Ethernet frame depends on the actual size of the certificate being transmitted. Ethernet frames typically have a maximum payload size of around 1500 bytes. If the certificate's size is within this limit, it can fit into a single frame; otherwise, it will be fragmented across multiple frames.

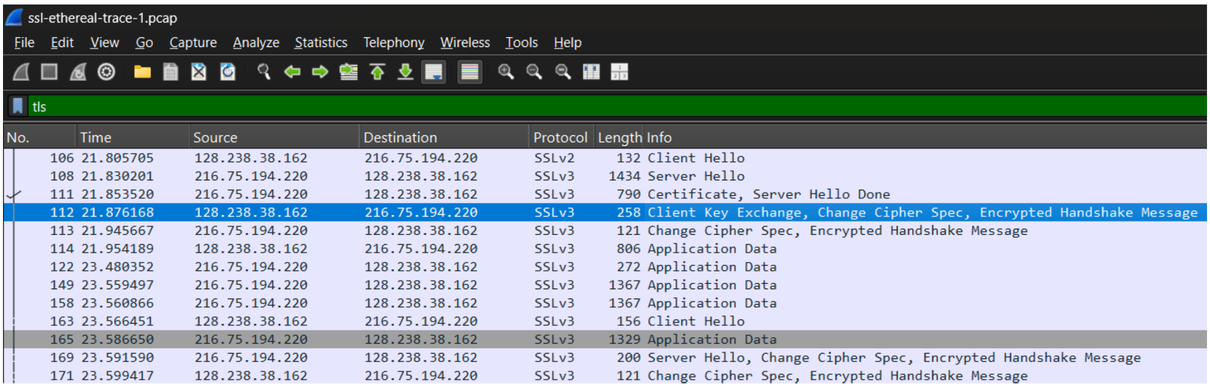




Client Key Exchange Record:

10. Locate the client key exchange record. Does this record contain a pre-master secret? What is this secret used for? Is the secret encrypted? If so, how? How long is the encrypted secret?

<u>Ans:</u>



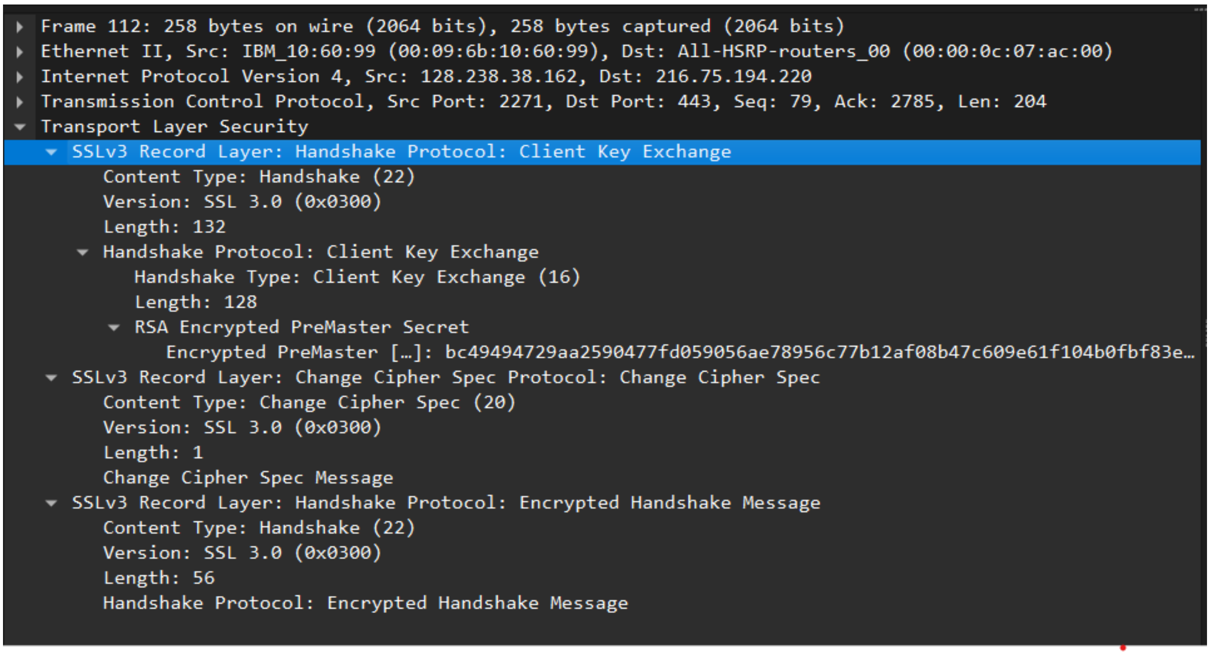
No.	Time	Source	Destination	Protocol	Length	Info
106	21.805705	128.238.38.162	216.75.194.220	SSLv2	132	Client Hello
108	21.830201	216.75.194.220	128.238.38.162	SSLv3	1434	Server Hello
111	21.853520	216.75.194.220	128.238.38.162	SSLv3	790	Certificate, Server Hello Done
112	21.876168	128.238.38.162	216.75.194.220	SSLv3	258	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
113	21.945667	216.75.194.220	128.238.38.162	SSLv3	121	Change Cipher Spec, Encrypted Handshake Message
114	21.954189	128.238.38.162	216.75.194.220	SSLv3	806	Application Data
122	23.480352	216.75.194.220	128.238.38.162	SSLv3	272	Application Data
149	23.559497	216.75.194.220	128.238.38.162	SSLv3	1367	Application Data
158	23.560866	216.75.194.220	128.238.38.162	SSLv3	1367	Application Data
163	23.566451	128.238.38.162	216.75.194.220	SSLv3	156	Client Hello
165	23.586650	216.75.194.220	128.238.38.162	SSLv3	1329	Application Data
169	23.591590	216.75.194.220	128.238.38.162	SSLv3	200	Server Hello, Change Cipher Spec, Encrypted Handshake Message
171	23.599417	128.238.38.162	216.75.194.220	SSLv3	121	Change Cipher Spec, Encrypted Handshake Message

Yes, the Client Key Exchange record contains a pre-master secret. This secret is used to derive the session keys for encryption and integrity in the SSL/TLS session.

Details:

1. **Use of the Pre-Master Secret**: The pre-master secret is combined with the client and server nonces to generate the session keys used for symmetric encryption and message authentication during the session.
2. **Encryption of the Pre-Master Secret**: The pre-master secret is encrypted using RSA encryption. This ensures that only the intended recipient (the server) can decrypt it using its private key.
3. **Length of the Encrypted Secret**: The length of the encrypted pre-master secret is 128 bytes (1024 bits). This length corresponds to the size of the RSA-encrypted data, which typically is larger than the original pre-master secret due to the encryption overhead.

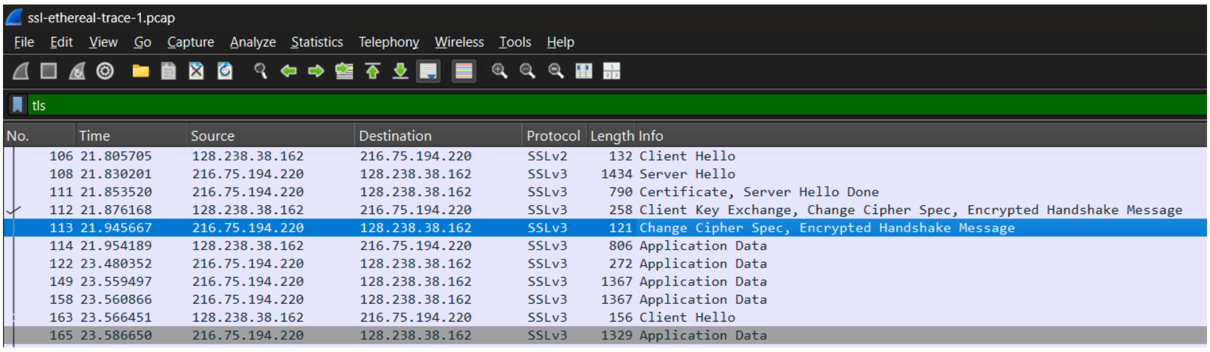
In summary, the Client Key Exchange record does contain an encrypted pre-master secret, which is essential for establishing secure communication between the client and server.



Change Cipher Spec Record (sent by client) and Encrypted Handshake Record:

11. What is the purpose of the Change Cipher Spec record? How many bytes is the record in your trace?

Ans:

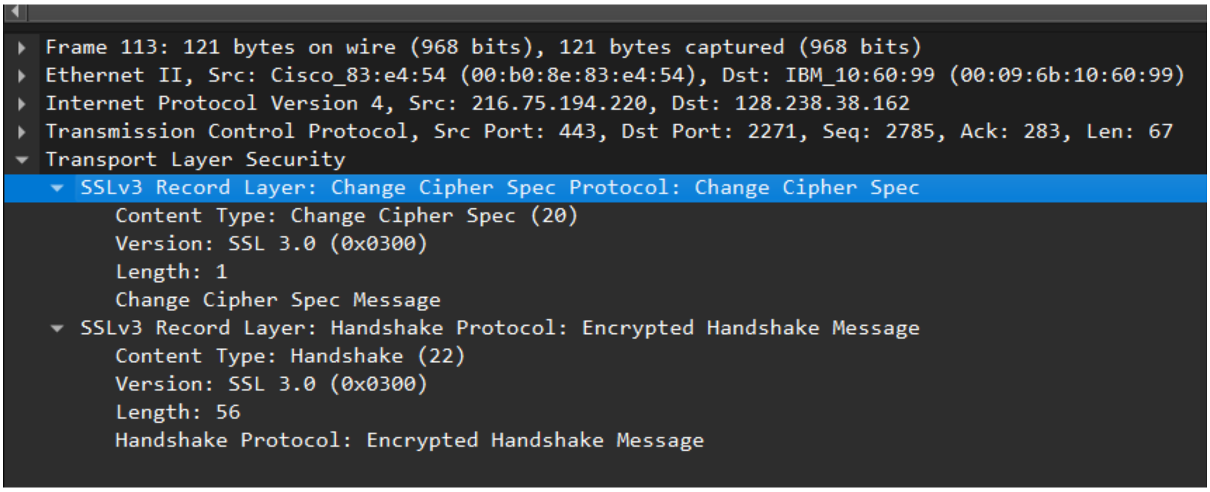


Purpose: The Change Cipher Spec record indicates that the sender is ready to switch to encrypted communication using the new cipher suite and keys.

The **Change Cipher Spec** record signals that the sender will start using the newly negotiated cipher suite and keys for subsequent messages. This record is part of the SSL/TLS handshake process and indicates that the sender is ready to switch from using the previous cipher settings to the new ones established during the handshake.

Size: The **Change Cipher Spec record** in above trace is **121 bytes** in total. This includes the Change Cipher Spec message itself, which is 1 byte, plus additional bytes related to the transmission (such as the Ethernet and IP headers).

This transition ensures that the communication is now encrypted using the new parameters established during the handshake.



12. In the encrypted handshake record, what is being encrypted? How?

<u>Ans:</u>

In the Encrypted Handshake Message record, the **data being encrypted is the handshake messages that were exchanged during the SSL/TLS handshake process after the Client Key Exchange.** This typically includes messages such as the ServerHelloDone message and any other necessary messages required to complete the handshake.

<u>**How is it encrypted?**</u>

1. Encryption Method: The messages are encrypted using the symmetric encryption algorithm specified by the negotiated cipher suite (in this case, TLS_RSA_WITH_RC4_128_MD5).
2. Keying Material: The encryption uses the session keys derived from the pre-master secret, along with the client and server nonces. This ensures that the data can only be decrypted by the intended recipient (the server in this case) who possesses the corresponding session key.
3. Integrity Check: Additionally, a Message Authentication Code (MAC) is often applied to ensure the integrity and authenticity of the messages, helping to prevent tampering.

Using symmetric-key algorithms determined by the negotiated cipher suite, leveraging session keys derived from the pre-master secret. The messages are encrypted and often accompanied by a MAC for integrity and authenticity.

In summary, the Encrypted Handshake Message record contains encrypted handshake messages that are protected using the newly established symmetric keys and algorithms.

13. Does the server also send a change cipher record and an encrypted handshake record to the client? How are those records different from those sent by the client?

Ans:

Yes, the server also sends a Change Cipher Spec record and an Encrypted Handshake Message record to the client. Here's how these records differ from those sent by the client:

Change Cipher Spec Record:

- Client: The Change Cipher Spec record sent by the client indicates that it is ready to switch to the new cipher suite and keys established during the handshake.

- Server: The server sends a Change Cipher Spec record to notify the client that it is also switching to the new cipher settings. The purpose is the same: to confirm that the server will now use the new encryption parameters.

Encrypted Handshake Message:

- Client: The Encrypted Handshake Message sent by the client contains handshake messages that have been encrypted using the negotiated cipher suite and session keys.
- Server: The server sends its own Encrypted Handshake Message that contains its handshake messages (such as ServerHelloDone) encrypted in the same way.

Key Differences:

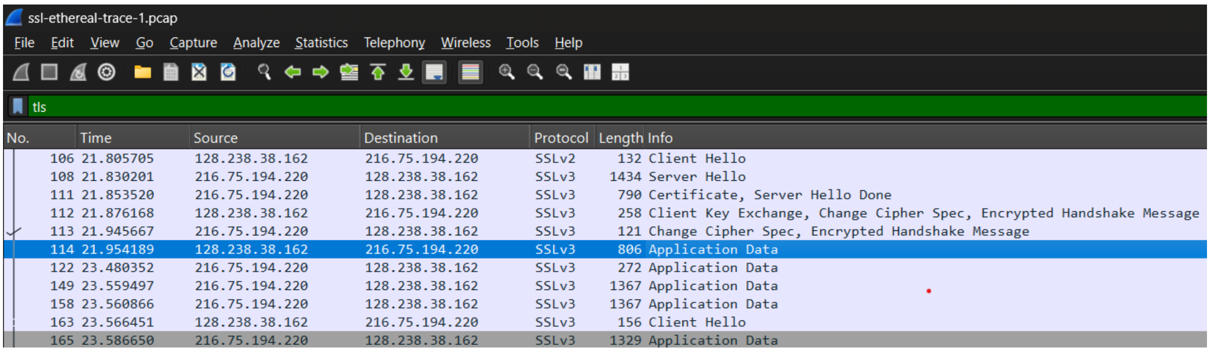
1. Content: The content of the records differs. The client's records will contain its specific handshake messages, while the server's records will contain its own messages.
2. Direction: The client’s Change Cipher Spec and Encrypted Handshake records are sent from the client to the server, while the server’s records are sent in the opposite direction.

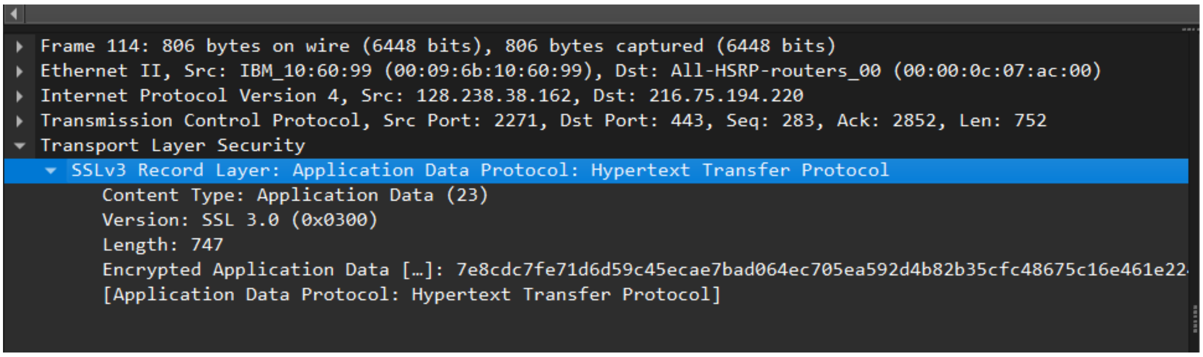
Overall, while the structure and purpose of the Change Cipher Spec and Encrypted Handshake Message records are consistent between client and server, the actual content and direction of communication differ.

Application Data:

14. How is the application data being encrypted? Do the records containing application data include a MAC? Does Wireshark distinguish between the encrypted application data and the MAC?

Ans:





How is the Application Data being Encrypted?

The application data is encrypted using the symmetric encryption algorithm specified by the negotiated cipher suite (in this case, likely TLS_RSA_WITH_RC4_128_MD5). The encryption is done using the session keys derived during the handshake process, which are based on the pre-master secret and the nonces.

Does the Record Containing Application Data Include a MAC?

Yes, records containing application data include a Message Authentication Code (MAC). The MAC ensures the integrity and authenticity of the data, protecting it from tampering and ensuring that it comes from a legitimate source.

Does Wireshark Distinguish Between the Encrypted Application Data and the MAC?

Wireshark typically does not display the MAC as a separate field in the decrypted view of the application data. Instead, the MAC is included in the encrypted payload. When the encrypted application data is analyzed, Wireshark shows it as a single encrypted block. The MAC is calculated over the plaintext data and is used during decryption to verify the integrity of the received data.

In summary, the application data is encrypted with session keys, includes a MAC for integrity, and Wireshark displays the data as a single encrypted entity without separating the MAC in its output.

15. Comment on and explain anything else that you found interesting in the trace.

Ans:

Use of Different SSL Versions:

The trace indicates a transition from SSLv2 to SSLv3. It's interesting to note the evolution of the SSL protocol versions, as SSLv2 is considered outdated and insecure. Modern applications primarily use TLS, which is the successor to SSL. The presence of SSLv2 could indicate compatibility settings or legacy systems.

Cipher Suite Negotiation:

The ClientHello message lists multiple cipher suites supported by the client. The server chooses one from this list for the session, which can reveal insights into the security posture and configurations of both the client and server. Observing this negotiation process can be critical for understanding potential vulnerabilities.

Challenge and Nonce Usage:

The ClientHello message includes a nonce (challenge), which is a random value used to prevent replay attacks. This is an interesting feature of SSL/TLS that enhances security by ensuring that each session is unique. The presence of nonces shows the protocols' design to handle specific security threats effectively.

Certificate Exchange:

The certificate exchange step during the ServerHello message and subsequent records is crucial for establishing trust. This trace shows the server providing its certificate, which may be signed by a trusted Certificate Authority (CA). The ability to verify this certificate is essential for the client to ensure that it is communicating with the legitimate server.

Packet Sizes and Performance:

Analyzing the sizes of the packets in the trace could provide insights into network performance. Larger packets may indicate bulk data transfers, while smaller packets might signify many small transactions. Identifying patterns in packet sizes could help in optimizing application performance and network resource utilization.

Timing of Records:

Observing the timing between records can provide insights into latency and performance issues. For example, if there are significant delays between the ClientHello and ServerHello messages, it could indicate network congestion or processing delays.

Application Data Records:

The presence of application data records after the handshake signals that secure communication has commenced. Analyzing the types of application data exchanged can provide insights into the nature of the application traffic, whether it's HTTP requests, file transfers, etc.

Network Security Considerations:

The trace can help identify potential security concerns, such as unencrypted traffic, or weak cipher suites. It is important to ensure that strong encryption practices are followed, as vulnerabilities in these areas could lead to exposure of sensitive data.

These points provide a deeper understanding of the SSL handshake process and the resulting secure communication, illustrating both the complexity and importance of cryptographic protocols in modern network security.