Security in Computing & Information Technology

COSC2536/2537

Assignment 2

Name: Duncan Do

Student Number: s3718718

Question 1 – Privacy-Preserving Computation

a) Generating the Public Key

$p = 59$

$q = 97$

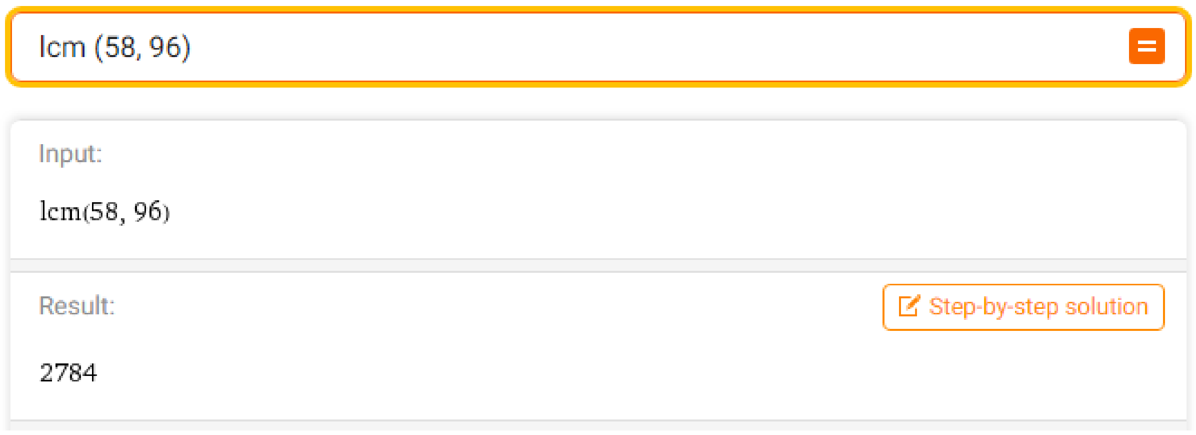
$g = 5{,}724$

$n = 5{,}723$

Public Key: (5723, 5724)

b) Generating the Private Key

$\lambda = \text{lcm}\ (59 - 1, 97 - 1) = \text{lcm}\ (58, 96)$



lcm (58, 96)

Input:

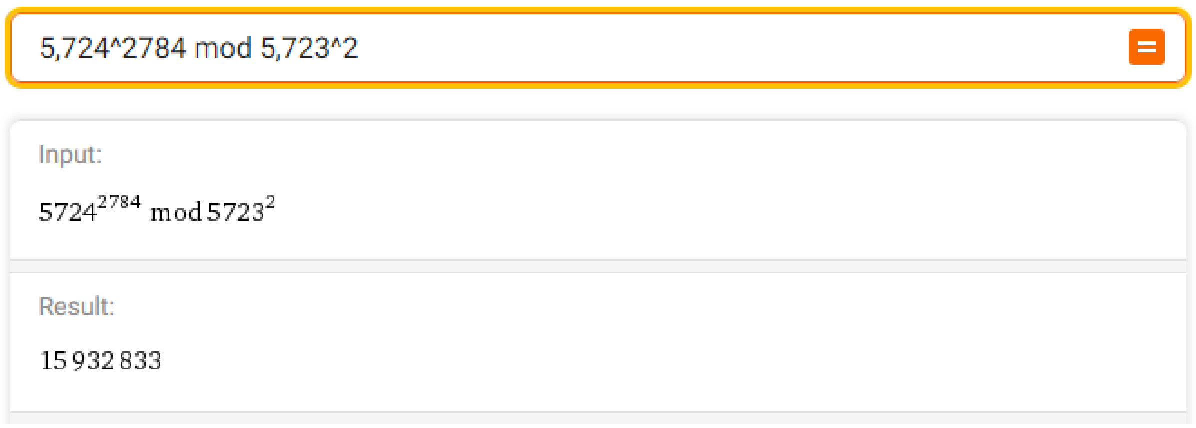
$\text{lcm}(58, 96)$

Result:

2784

$\lambda = 2{,}784$

$k = L\ (5{,}724^{2784} \bmod 5{,}723^2)$



5,724^2784 mod 5,723^2

Input:

$5724^{2784} \bmod 5723^2$

Result:

15 932 833

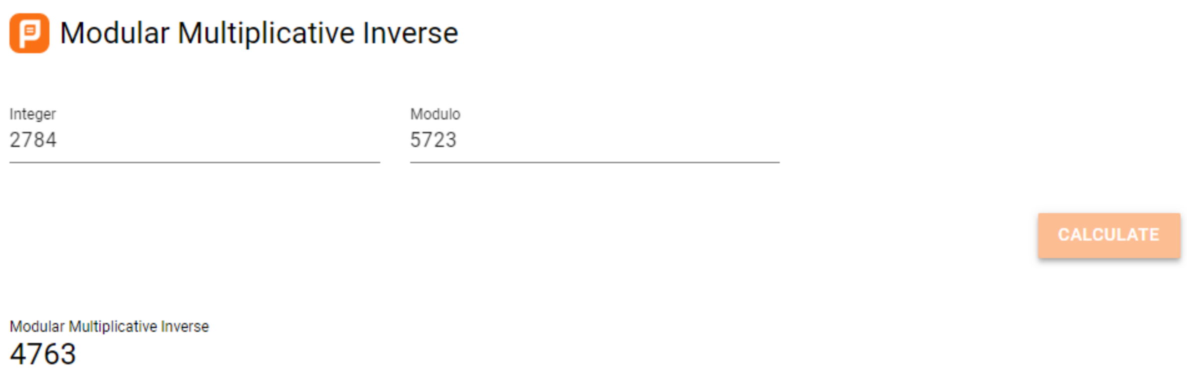
$k = L(15{,}932{,}833)$

$k = (15{,}932{,}833 - 1)/5{,}723$

$k = 15{,}932{,}832/5{,}723$

$k = 2{,}784$

$\mu = 2{,}784^{-1} \bmod 5{,}723$



Modular Multiplicative Inverse

Integer: 2784

Modulo: 5723

Modular Multiplicative Inverse: 4763

$\mu = 4{,}763$

Private Key: (2784, 4,763)

c) Encrypting Votes

Voter No.	Voter's Private Number, *r*	Vote	Voting message, *m*
1	40	YES	**00010000 = 16**
2	41	YES	**00010000 = 16**
3	42	YES	**00010000 = 16**
4	43	YES	**00010000 = 16**
5	44	YES	**00010000 = 16**
6	45	YES	**00010000 = 16**
7	46	YES	**00010000 = 16**
8	47	NO	**00000001 = 1**
9	48	NO	**00000001 = 1**
10	49	NO	**00000001 = 1**
11	50	NO	**00000001 = 1**

$g = 5{,}724$

$n = 5{,}723$

$5{,}724^{16} \times 40^{5{,}723} \bmod 5{,}723^{2} = 22{,}848{,}230$

$5{,}724^{16} \times 41^{5{,}723} \bmod 5{,}723^{2} = 24{,}785{,}522$

$5{,}724^{16} \times 42^{5{,}723} \bmod 5{,}723^{2} = 19{,}405{,}678$

$5{,}724^{16} \times 43^{5{,}723} \bmod 5{,}723^{2} = 21{,}780{,}777$

$5{,}724^{16} \times 44^{5{,}723} \bmod 5{,}723^{2} = 21{,}683{,}720$

$5{,}724^{16} \times 45^{5{,}723} \bmod 5{,}723^{2} = 4{,}823{,}473$

$5{,}724^{16} \times 46^{5{,}723} \bmod 5{,}723^{2} = 8{,}614{,}744$

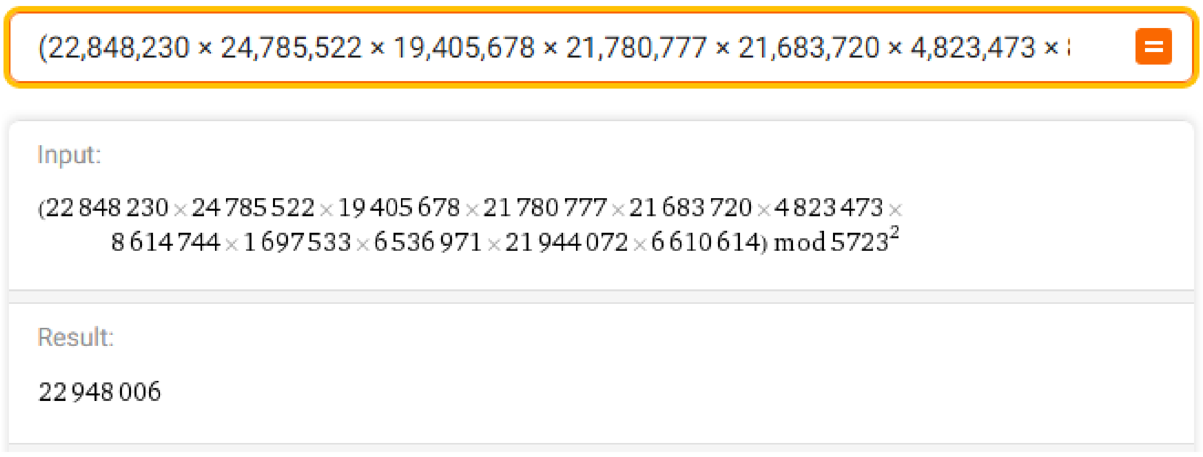
$5{,}724^{1} \times 47^{5{,}723} \bmod 5{,}723^{2} = 1{,}697{,}533$

$5{,}724^{1} \times 48^{5{,}723} \bmod 5{,}723^{2} = 6{,}536{,}971$

$5{,}724^{1} \times 49^{5{,}723} \bmod 5{,}723^{2} = 21{,}944{,}072$

$5{,}724^{1} \times 50^{5{,}723} \bmod 5{,}723^{2} = 6{,}610{,}614$

$C = (22{,}848{,}230 \times 24{,}785{,}522 \times 19{,}405{,}678 \times 21{,}780{,}777 \times 21{,}683{,}720 \times 4{,}823{,}473 \times 8{,}614{,}744 \times 1{,}697{,}533 \times 6{,}536{,}971 \times 21{,}944{,}072 \times 6{,}610{,}614) \bmod 5{,}723^{2}$



(22,848,230 × 24,785,522 × 19,405,678 × 21,780,777 × 21,683,720 × 4,823,473 ×

Input:

$(22\,848\,230 \times 24\,785\,522 \times 19\,405\,678 \times 21\,780\,777 \times 21\,683\,720 \times 4\,823\,473 \times 8\,614\,744 \times 1\,697\,533 \times 6\,536\,971 \times 21\,944\,072 \times 6\,610\,614) \bmod 5723^{2}$

Result:

22 948 006

$C = 22{,}948{,}006$

d) Counting Votes

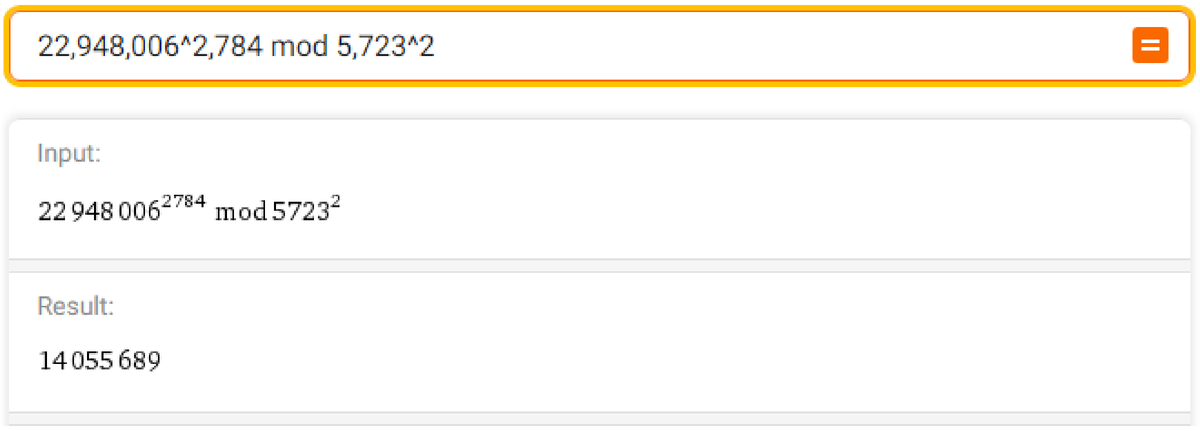
$\lambda = 2{,}784$

$n = 5{,}723$

$\mu = 4{,}763$

$C = 22{,}948{,}006$

$m = L(22{,}948{,}006^{2{,}784} \bmod 5{,}723^{2}) \times 4{,}763 \bmod 5{,}723$



22,948,006^2,784 mod 5,723^2

Input:

$22\,948\,006^{2784} \bmod 5723^2$

Result:

14 055 689

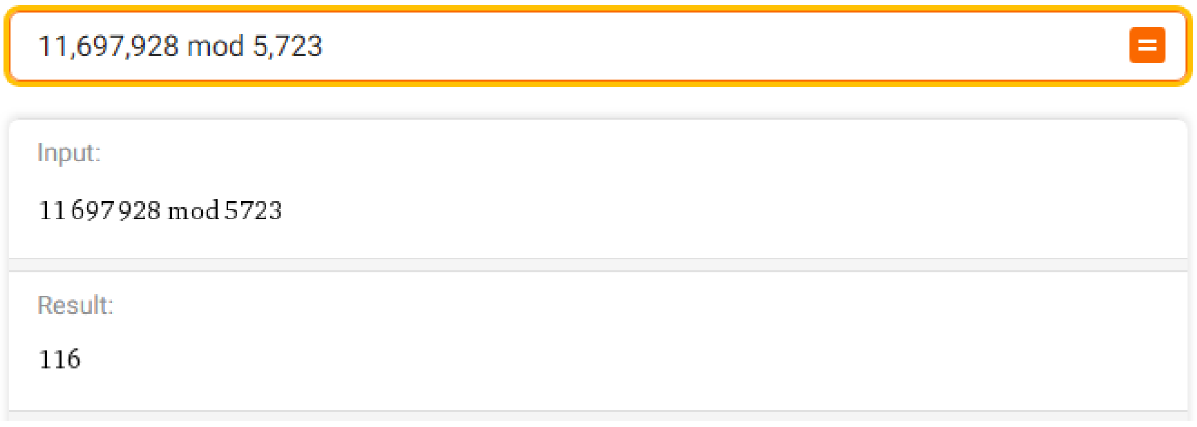
m = L(14,055,689) × 4,763 mod 5,723

L(14,055,689) = (14,055,689 – 1)/5,723

L(14,055,689) = (14,055,688)/5,723 = 2,456

m = 2,456 × 4,763 mod 5,723

m = 11,697,928 mod 5,723



11,697,928 mod 5,723

Input:

11 697 928 mod 5723

Result:

116

M = 116 (In binary: 0111 0100)

$0111_2 = 7_{10}$ (7 people voted 'yes')

$0100_2 = 4_{10}$ (4 people voted 'no')

Question 2 – Signatures

Q2.1 [RSA encryption algorithm]

a) Generating the Public Key

p = 10193

q = 8,287

e = 5,903

n = 10,193 × 8,287 = 84,469,391

φ(n) = (10,193 – 1) × (8,287 – 1) = 10,192 × 8,286 = 84,450,912

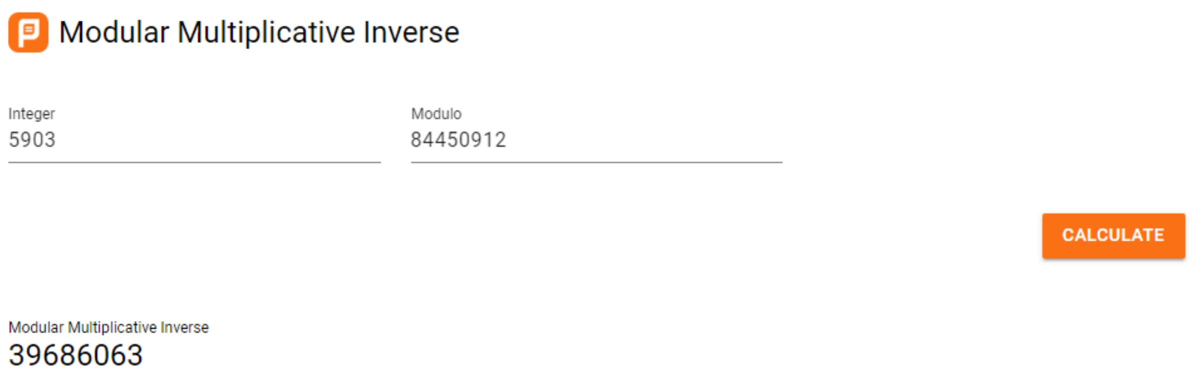
Public Key: (84,469,391, 5,903)

b) Generating the Private Key

φ(n) = 84,450,912

e = 5,903

d × 5,903 = 1 mod 84,450,912



Modular Multiplicative Inverse

Integer: 5903

Modulo: 84450912

Modular Multiplicative Inverse

39686063

d = 39,686,063

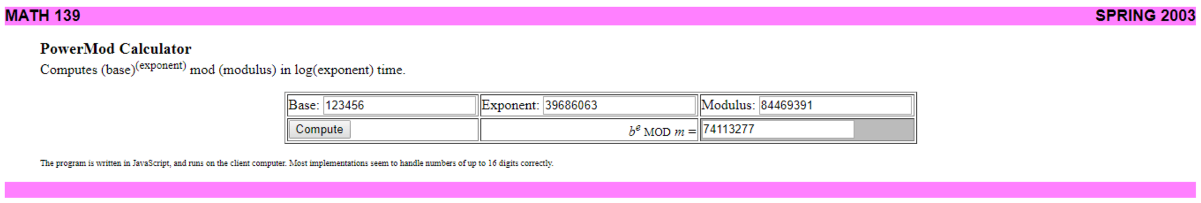
c) Signing

m = 123,456

n = 84,469,391

d = 39,686,063

s = $123456^{39,68,063}$ mod 84,469,391



MATH 139 SPRING 2003

PowerMod Calculator

Computes $(\text{base})^{(\text{exponent})}$ mod (modulus) in log(exponent) time.

Base: 123456 Exponent: 39686063 Modulus: 84469391

b^e MOD m = 74113277

The program is written in JavaScript, and runs on the client computer. Most implementations seem to handle numbers of up to 16 digits correctly.

s = 74,113,277

Signed Message: (123456, 74113277)

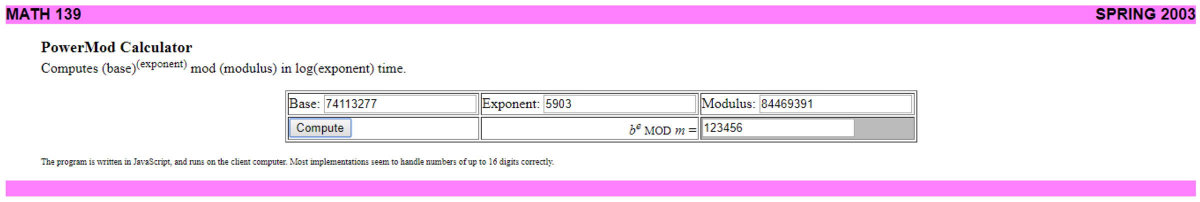
d) Verification

n = 84,469,391

e = 5,903

s = 74,113,277

$m^I = 74{,}113{,}277^{5{,}903} \bmod 84{,}469{,}391$



MATH 139 SPRING 2003

PowerMod Calculator

Computes (base)^(exponent) mod (modulus) in log(exponent) time.

Base: 74113277	Exponent: 5903	Modulus: 84469391
Compute	b^e MOD m =	123456

The program is written in JavaScript, and runs on the client computer. Most implementations seem to handle numbers of up to 16 digits correctly.

$m^I = 123{,}456$

$m = m^1$

∴ VERIFIED

Q2.2 [ElGamal encryption algorithm]

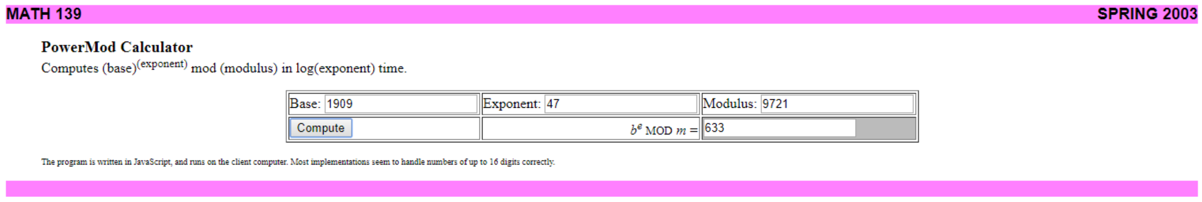
a) Generating the public key parameter 'y'

p = 9,721

g = 1,909

x = 47

$y = 1{,}909^{47} \bmod 9{,}721$



MATH 139 SPRING 2003

PowerMod Calculator

Computes (base)^(exponent) mod (modulus) in log(exponent) time.

Base: 1909	Exponent: 47	Modulus: 9721
Compute	b^e MOD m =	633

The program is written in JavaScript, and runs on the client computer. Most implementations seem to handle numbers of up to 16 digits correctly.

y = 633

b) Selecting k

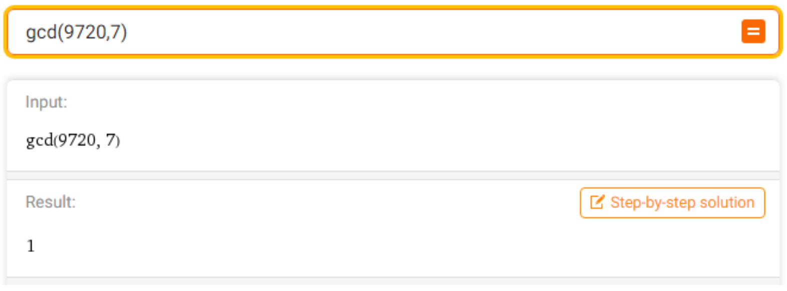
Selecting k

1 </ k </ 9,721 – 2 && gcd (k, 9,721 – 1)

k can be 7

7 < 9719

gcd (7, 9720) = 1


gcd(9720,7)

Input:

gcd(9720, 7)

Result:

1

c) Signing

m = 5,432

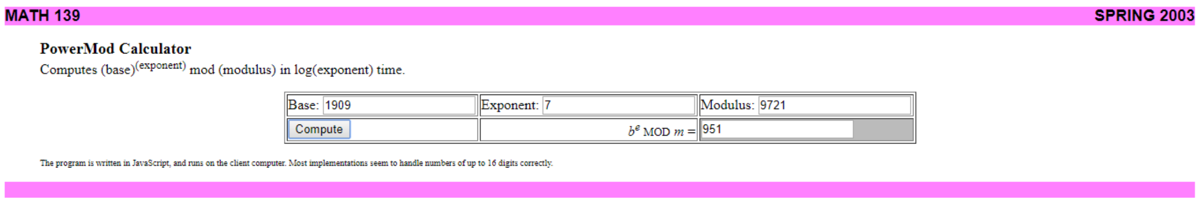
x = 47

g = 1,909

p = 9,721

k = 7

r = 1,909^{7} mod 9,721


MATH 139 SPRING 2003

PowerMod Calculator

Computes (base)$^{(exponent)}$ mod (modulus) in log(exponent) time.

Base: 1909 Exponent: 7 Modulus: 9721

b^e MOD m = 951

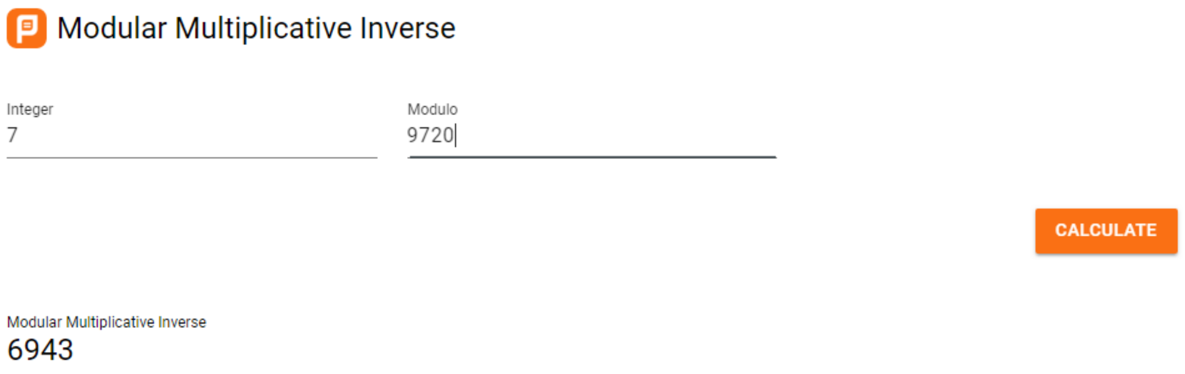
The program is written in JavaScript, and runs on the client computer. Most implementations seem to handle numbers of up to 16 digits correctly.

r = 951

s = 7^{-1} (5,432 – 47 × 951) mod (9,721 – 1)

s = 7^{-1} (-39,265) mod (9,720)

s = 7^{-1} mod (9,720) × (-39,265) mod (9,720)


Modular Multiplicative Inverse

Integer 7

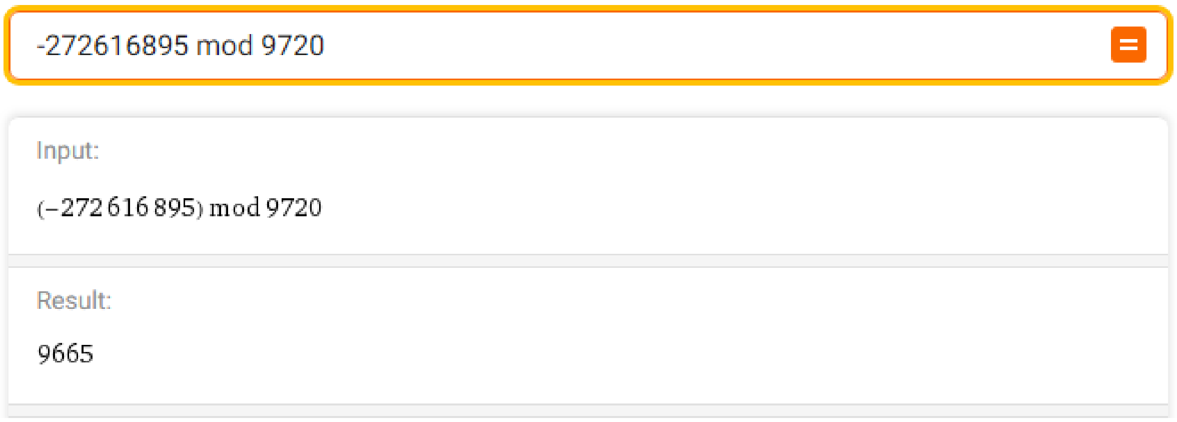
Modulo 9720

Modular Multiplicative Inverse

6943

s = 6943 × (-39,265) mod (9,720)

s = -272,616,895 mod (9,720)


-272616895 mod 9720

Input:

(−272 616 895) mod 9720

Result:

9665

s = 9,665

Signed Message: (5432, 951, 9665)

d) Verification

a. Checking ‘r’

r = 951

951 >/ 1

951 </ 9721 – 1

b. Determining if ‘v’ and ‘w’ are equivalent

m = 5,432

g = 1,909

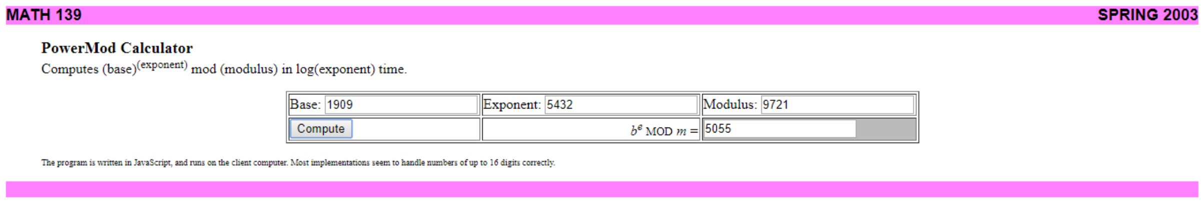
p = 9,721

y = 633

r = 951

s = 9,665

$v = 1{,}909^{5{,}432} \bmod 9{,}721$


MATH 139 SPRING 2003

PowerMod Calculator

Computes (base)$^{(exponent)}$ mod (modulus) in log(exponent) time.

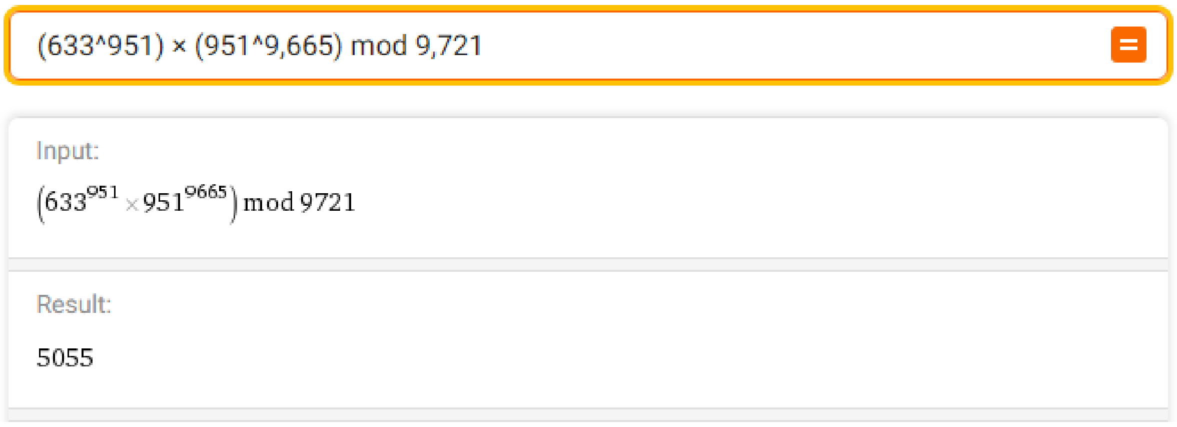
Base: 1909	Exponent: 5432	Modulus: 9721
Compute	b^e MOD m =	5055

The program is written in JavaScript, and runs on the client computer. Most implementations seem to handle numbers of up to 16 digits correctly.

v = 5055

$w = 633^{951} \times 951^{9{,}665} \bmod 9{,}721$

$w = ((633^{951} \bmod 9{,}721) \times (951^{9{,}665} \bmod 9{,}721)) \bmod 9{,}721$


(633^951) × (951^9,665) mod 9,721

Input:

$(633^{951} \times 951^{9665}) \bmod 9721$

Result:

5055

$w = 5055$

$v = w \therefore$ Signature is accepted

Q2.3

a) Generating the public key

$h(M) = $ dbafc095e552176dd482cea445d199a2

(In Decimal) = 292,013,489,125,751,596,553,767,941,623,740,733,858

$p = 307699126915021078949717556805305347641$

$q = 286189067004968539490940912607240844261$

$e = 47$

$n = p \times q =$
88,060,126,050,053,286,133,358,329,588,325,261,416,508,643,838,108,904,670,297,433,897,418,944,738,301

$\varphi(n) = (p - 1) \times (q - 1) =$
88,060,126,050,053,286,133,358,329,588,325,261,415,914,755,644,188,915,051,856,775,428,006,398,546,400

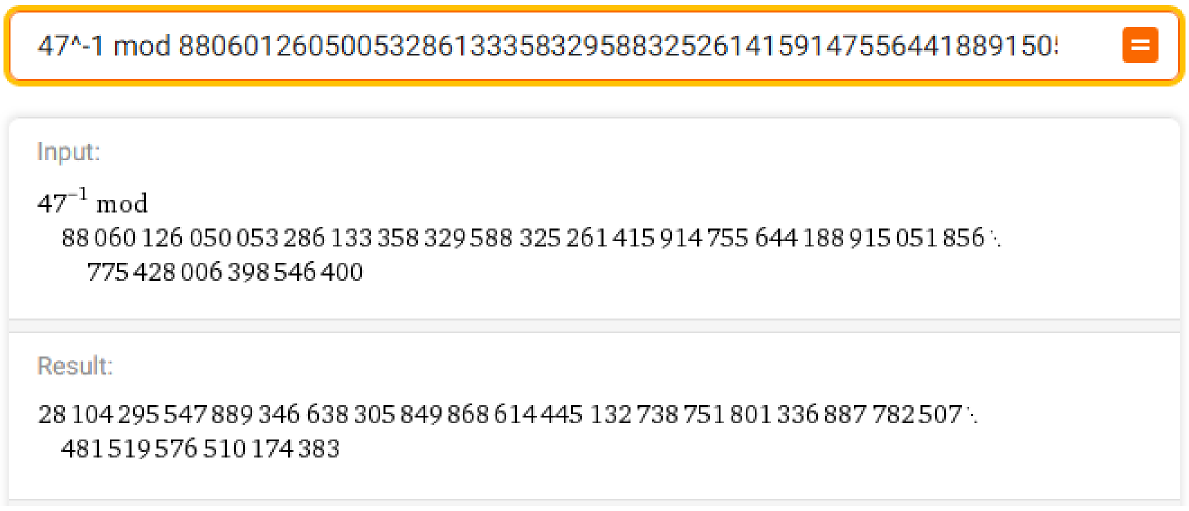
Public Key =
(8806012605005328613335832958832526141650864383810890467029743389741894473 8301, 47)

b) Generating the private key

$\varphi(n) =$
88,060,126,050,053,286,133,358,329,588,325,261,415,914,755,644,188,915,051,856,775,428,006,398,546,400

$e = 47$

$d \times 47 = 1 \bmod$
88,060,126,050,053,286,133,358,329,588,325,261,415,914,755,644,188,915,051,856,775,428,006,398,546,400


47^-1 mod 880601260500532861333583295883252614159147556441889150

Input:

47^{-1} mod
88 060 126 050 053 286 133 358 329 588 325 261 415 914 755 644 188 915 051 856 775 428 006 398 546 400

Result:

28 104 295 547 889 346 638 305 849 868 614 445 132 738 751 801 336 887 782 507 481 519 576 510 174 383

d = 28,104,295,547,889,346,638,305,849,868,614,445,132,738,751,801,336,887,782,507,481,519,576,510,174,383

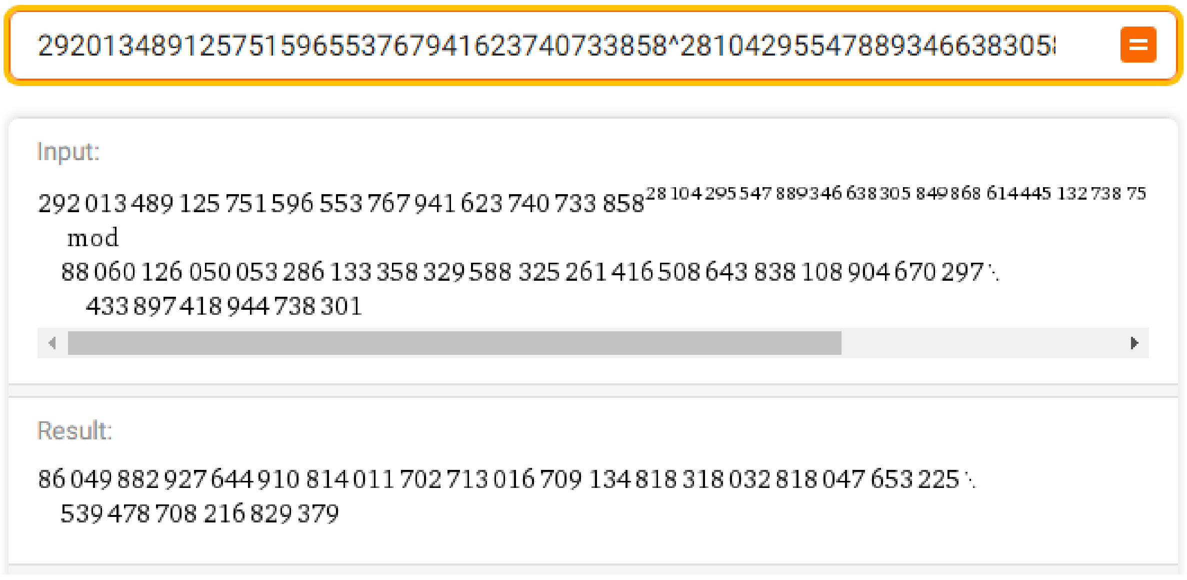
- Signing

h(M) = 292,013,489,125,751,596,553,767,941,623,740,733,858

n = 88,060,126,050,053,286,133,358,329,588,325,261,416,508,643,838,108,904,670,297,433,897,418,944,738,301

d = 28,104,295,547,889,346,638,305,849,868,614,445,132,738,751,801,336,887,782,507,481,519,576,510,174,383

s = m^d mod n


292013489125751596553767941623740733858^28104295547889346638305

Input:

$292\,013\,489\,125\,751\,596\,553\,767\,941\,623\,740\,733\,858^{28\,104\,295\,547\,889\,346\,638\,305\,849\,868\,614\,445\,132\,738\,75}$
mod
88 060 126 050 053 286 133 358 329 588 325 261 416 508 643 838 108 904 670 297 433 897 418 944 738 301

Result:

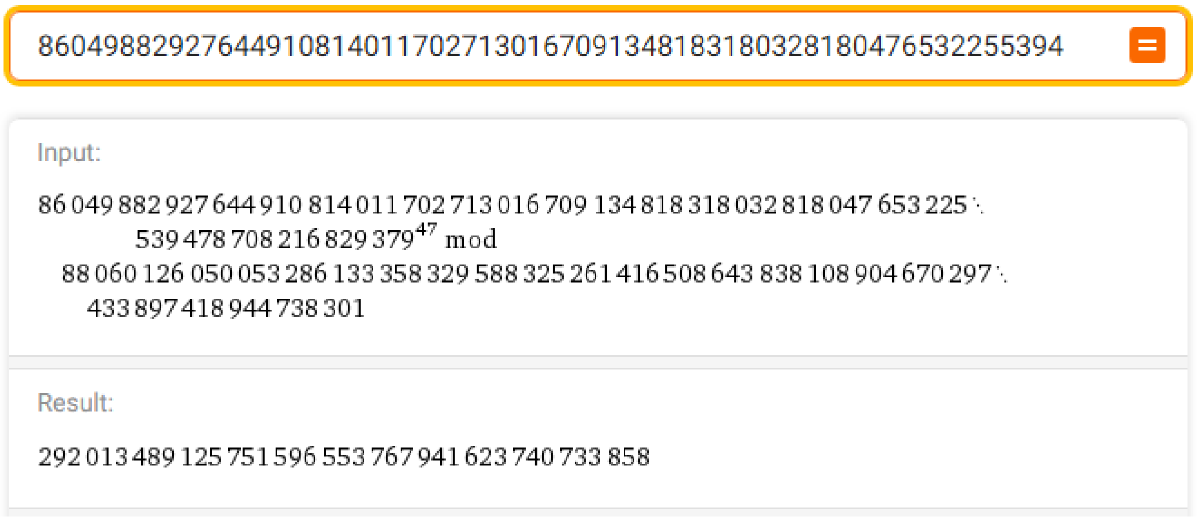
86 049 882 927 644 910 814 011 702 713 016 709 134 818 318 032 818 047 653 225 539 478 708 216 829 379

s = 86,049,882,927,644,910,814,011,702,713,016,709,134,818,318,032,818,047,653,225,539,478,708,216,829,379

Signed Message: (292013489125751596553767941623740733858, 8604988292764491081401170271301670913481831803281804765322553947870821682 9379)

- Verification

$m^1 = s^e \bmod n$



```
860498829276449108140117027130167091348183180328180476532255394

Input:

86 049 882 927 644 910 814 011 702 713 016 709 134 818 318 032 818 047 653 225 ⋱
          539 478 708 216 829 379^47 mod
  88 060 126 050 053 286 133 358 329 588 325 261 416 508 643 838 108 904 670 297 ⋱
     433 897 418 944 738 301

Result:

292 013 489 125 751 596 553 767 941 623 740 733 858
```

$m^1 = 292,013,489,125,751,596,553,767,941,623,740,733,858$

$h(m^1)$ = dbafc095e552176dd482cea445d199a2

$h(m) = h(m^1)$

∴ VERIFIED

Q2.4

Bob forwarded the message to Charlie, fooling Charlie to think that the message was from Alice, for Charlie. To prevent this, Alice would need to encrypt the message with Bobs public key first before signing it with her own private key. This way, if Bob tries to forward the message to Charlie (Trying to fool him to thinking it's from Alice); he would have to:

1) Verify the signature with Alice's public key
2) Decrypt the message with his private key
3) Re-encrypt it with Charlie's public key
4) Sign it again with Alice's private key

However, Bob does not know Alice's private key. Thus, he cannot re-sign the message to fool Charlie to thinking he got the message from Alice.

ORIGINAL CASE

$A \text{---}\{[M]_{Alice}\}_{Bob}\text{--->} B \text{---}\{[M]_{Alice}\}_{Charlie}\text{--->} C$

REVISED CASE

$A \text{---}[\{M\}_{Bob}]_{Alice}\text{--->} B$

$A \text{---}[\{M\}_{SYMM}]_{Alice}\text{--->} B$

However, Alice cannot prevent this attack using symmetric key. Alice does not intend to send this message to Charlie, therefore they do not share a symmetric key. She only shares one with Bob; so, Bob can decrypt the message. The only way to symmetric keys to prevent the attack is if Bob and Charlie share a symmetric key, so the only way for Charlie to decrypt the message is to use their symmetric key; but in doing that, Charlie knows it's from Bob and not Alice. This makes no logical sense as Bob would be undermining his own scheme by using symmetric key.

Alice can use a symmetric key she has with Bob instead of Bob's public key (to encrypt), but it does not prevent the attack any better.

Question 3 – Blockchain Technology

Consider a produce supply chain where the produce must make its way from the initial manufacturer to the consumer. The produce must be handed off to multiple carriers to add something to the product. For example, a packaging must be added to the produce, the package needs to be shipped to the retailer, and the retailer must sell the product to the consumer.

For the sake of the scenario, this produce must be kept in a temperature threshold. (Say, -5^{o}C to 5^{o}C). Thus, in every stage of the chain, the temperature must be maintained. Let's say for each stage of the block chain (block), a set list of data must be provided. E.g.

- Received time/date from the previous stage
- Delivery time/date of the next stage
- Temperature of the product when receiving and delivering
- A private ID of each department
- A hash of all the information

Example of a block

Previous Hash	Received Date	Received Time	Received Temp	Delivery Date	Delivery Time	Delivery Temp	Private ID

HASH(biiiiiiiiiggggggggHassssssssssssssshhhhh|5-5-18|2PM|-3|6-5-18|1PM|-2|123) = biiiiiiiiiiiiiiggggggggggeeeeeeeeeeeeeeerhaaaaaaaaaassshhhhhh

The initial stage would generate a genesis block (the first block in the block chain). Thus, it would only be able to provide parts of the required data. (E.g. no received time/date can be supplied).

Example of a genesis block

Hash: biiiiiiiiiigggggggeeeeerhashhhhhhhhhh
Previous Hash: 0
Private ID: 123
Delivery:

- **Date:** 6-5-18
- **Time:** 1PM
- **Temp:** -2

The previous hash is 0 because this is the first block in the chain, thus there is no previous block to have a hash. Furthermore, only delivery date/time/temp can be provided as the manufacturer make the product rather than receiving it from somewhere.

Once the produce goes through, manufacturer, packaging, shipment. and finally, retailer; there will be a chain of hashes provided building of the previous.

Example of the packaging block

Hash: biiiiiiiiiigggggggeeeeerhashhhhhhhhhh
Previous Hash: biiiiiiiiiiggggggghaaaaaaaaaashhhh
Private ID: 456
Received:
- **Date:** 7-5-18
- **Time:** 10AM
- **Temp:** -2

Delivery:
- **Date:** 8-5-18
- **Time:** 12PM
- **Temp:** 1

This time, a previous hash and received data can be provided sine there a block before in the chain. The blocks for each step in the chain follow the same format, gathering all their necessary data and hashing it to send to the next block in the chain.

Proof of Work

- Proof of work can be used in this blockchain based distribution system to ensure that someone adding to the chain is trustworthy
- Before the transaction in a stage can take place, they are required to solve a mathematical function first. Once the function is solved, we have proof of work for that stage, thus they can add to the blockchain
- Proof of work function

Previous Hash	+	Data Block	+	Nonce	=>	New Hash Value

Proof of work involves concatenating a nonce to the data such that the hash produced fits a certain criterion.
E.g.
Data: 1034
*Append a nonce to the end of the data until that concatenation results in a md5 hash that has 4 leading 0s

Md5(1034**1**) =
859b755563f548d008f936906a959c8f

(Starting with 1, the resulting hash failed to meet the criterion)

Md5(1034**2**) =
7b3564b05f78b6739d06a2ea3187f5ca

(Now trying 2, also a fail. Keep trying until a nonce yield the desired result)

Md5(1034**599**) =
0000fc70ad0a307d08f88a484dd99cb4

(After 599 attempts, we now have proof of work)

- Solving this function proves that the packaging/delivery etc. stage can be trusted; thus, they can add their hash to the blockchain

Integrity and Traceability

- The use of blockchain in this scenario allows an audit manager to determine if any stage did something wrong (E.g. had their temperature go over 5^oC). This is because the audit manager knows the data that each block had to include (E.g. delivery time/date/temp). Thus, the audit manager can reproduce the hashes of each stage (with accepted temperatures) and see if they match up with the one provided by that stage. If it is different, then it is clear which stage messed up.
  Furthermore, if a stage tried to forge some of their data (To, for example, pretend their temperature was in the correct threshold), they would only be able to change their own hash as they do not know the private keys of the other stages to go and change their temperatures. This would mean that the new hash that the stage 'forged' would be different to the actual hash of the block and any other block's hashes further along the chain. That means that the value cannot be changed.

Advantages and Disadvantages of using blockchain

Advantages	Disadvantages
- Able to easily identify a stage if they do something wrong to violate the conditions of what they are sending - Able to detect if a stage has tampered with the data	- Transaction speeds will be slower as each block must do several long processes - The data is less private as blockchain requires several unaffiliated participants to coordinate with each other

Question 4 – Authentication Protocol

Q4.1

ORIGINAL CASE

A --- "I'm Alice", R_A ---> B

A <--- R_B, E (R_A, K) --- B

If Trudy is posing as Alice, Trudy can then ask Bob to encrypt the nonce he just sent (R_B)

A --- "I'm Alice", R_B ---> B

A <--- R_C, E (R_B, K) --- B

Once Trudy has the encryption with R_B, Trudy can send it right back as a response to the second line; acting the same as Alice without ever knowing the key

REVISED CASE

A --- "I'm Alice", R_A ---> B

A <--- R_B, E ("Bob", R_A, K_{AB}) --- B

A --- R_B, E ("Alice", R_A, K_{AB}) ---> B

Including the user's name with the encryption verifies the identity of each party, thus we have mutual authentication.

4.2

ORIGINAL CASE

A --- “I’m Alice”, $[\{T, K\}_{Bob}]_{Alice}$ ---> B

A <--- $[\{T + 1, K\}_{Alice}]_{Bob}$ --- B

Trudy can intercept this and obtain $[\{T, K\}_{Bob}]_{Alice}$. The signature does nothing to protect the message, so all Trudy needs to do is get the message decrypted to get the session key, ‘K’. Although, a symmetric key can be used to encrypt the messages instead of the user’s public key; this does nothing to change the security against attacks but does involve symmetric key in the solution. In the end, the solution is to change the order of encryption and signing.

T --- “I’m Trudy”, $[\{T, K\}_{Bob}]_{Trudy}$ ---> B

T <--- $[\{T + 1, K\}_{Trudy}]_{Bob}$ --- B

Thus, Trudy can simply get bob to decrypt T, K and send it back encrypted with Trudy’s public key. Now Trudy can decrypt it with her private key and obtain ‘K’.

REVISED CASE

A --- “I’m Alice”, $\{[T, K]_{Bob}\}_{Alice}$ ---> B

A <--- $\{[T + 1, K]_{Alice}\}_{Bob}$ --- B

Now, even if Trudy intercepts $\{[T, K]_{Bob}\}_{Alice}$ and sends it to Bob to decrypt it; Bob will send it back encrypted with Alice’s public key. In this case, Trudy cannot decrypt it afterwards as he cannot change the signature from $\{[T, K]_{Bob}\}_{Alice}$.

Question 5 – Data Hiding

DONE IN DEMO

Question 6 – Open SSL and IPFS

DONE IN DEMO