MFC Take home solution

Ahmad Maghaireh

23110205

2024

Table of contents

# Question 1

## Step (I): Key Generation

### (a) Select two primes p and q so that

### i. p is a three-digit factorial prime. Show that the number you picked is indeed a factorial prime.

A factorial prime is a prime number that is one less than or one more than a factorial of some positive integer.

P=719

719=6!−1=720−1=719

Since 719 is a prime number and is one less than 720 (which is 6!), 719 is a factorial prime.

Show that 719 is a prime number :

The square root of 719 is 26.8

So lets take the 26 and make a divisibility check for numbers form 2 to 26

If all them doesn’t divid the number 719 , that’s mean its prime

And have two factors (1 and 719)

Since 719 is odd, it is not divisible by any even number. Therefore, we only need to test odd numbers up to the square root of 719 (approximately 26.8)

Integer Calculation Result Divisible?

3 719 ÷ 3 ≈ 239.6667 Not an integer No

5 719 ÷ 5 ≈ 143.8 Not an integer No

7 719 ÷ 7 ≈ 102.7143 Not an integer No

9 719 ÷ 9 ≈ 79.8889 Not an integer No

11 719 ÷ 11 ≈ 65.3636 Not an integer No

13 719 ÷ 13 ≈ 55.3077 Not an integer No

15 719 ÷ 15 ≈ 47.9333 Not an integer No

17 719 ÷ 17 ≈ 42.2941 Not an integer No

19 719 ÷ 19 ≈ 37.8421 Not an integer No

21 719 ÷ 21 ≈ 34.2381 Not an integer No

23 719 ÷ 23 ≈ 31.2609 Not an integer No

25 719 ÷ 25 ≈ 28.76 Not an integer No

### ii. q is a four-digit Mersenne prime. Show that the number you picked is indeed a Mersenne prime.

A Mersenne prime is a prime number that is one less than a power of two.

Let's choose q=8191q=8191. To show that 8191 is a Mersenne prime:

8191=(2^13)−1

Since 8191 is a prime number and is one less than 2^13, 8191 is a Mersenne prime.

### Remark: Here, you need to show first that the chosen number is a prime.

Integer Calculation Result Divisible?

3 8191 ÷ 3 ≈ 2730.3333 Not an integer No

5 8191 ÷ 5 ≈ 1638.2 Not an integer No

7 8191 ÷ 7 ≈ 1170.1429 Not an integer No

9 8191 ÷ 9 ≈ 910.1111 Not an integer No

11 8191 ÷ 11 ≈ 744.6364 Not an integer No

13 8191 ÷ 13 ≈ 630.8462 Not an integer No

15 8191 ÷ 15 ≈ 546.0667 Not an integer No

17 8191 ÷ 17 ≈ 481.8235 Not an integer No

19 8191 ÷ 19 ≈ 431.1053 Not an integer No

21 8191 ÷ 21 ≈ 390.0476 Not an integer No

23 8191 ÷ 23 ≈ 356.1304 Not an integer No

25 8191 ÷ 25 ≈ 327.64 Not an integer No

27 8191 ÷ 27 ≈ 303.3704 Not an integer No

29 8191 ÷ 29 ≈ 282.4483 Not an integer No

31 8191 ÷ 31 ≈ 264.2258 Not an integer No

33 8191 ÷ 33 ≈ 248.2121 Not an integer No

35 8191 ÷ 35 ≈ 234.0286 Not an integer No

37 8191 ÷ 37 ≈ 221.3784 Not an integer No

39 8191 ÷ 39 ≈ 210.0256 Not an integer No

41 8191 ÷ 41 ≈ 199.7805 Not an integer No

43 8191 ÷ 43 ≈ 190.4884 Not an integer No

45 8191 ÷ 45 ≈ 182.0222 Not an integer No

47 8191 ÷ 47 ≈ 174.2766 Not an integer No

49 8191 ÷ 49 ≈ 167.1633 Not an integer No

51 8191 ÷ 51 ≈ 160.6078 Not an integer No

53 8191 ÷ 53 ≈ 154.5472 Not an integer No

55 8191 ÷ 55 ≈ 148.9273 Not an integer No

57 8191 ÷ 57 ≈ 143.5263 Not an integer No

59 8191 ÷ 59 ≈ 138.8136 Not an integer No

61 8191 ÷ 61 ≈ 134.2787 Not an integer No

63 8191 ÷ 63 ≈ 130.4921 Not an integer No

65 8191 ÷ 65 ≈ 126.7846 Not an integer No

67 8191 ÷ 67 ≈ 122.9851 Not an integer No

69 8191 ÷ 69 ≈ 118.7246 Not an integer No

71 8191 ÷ 71 ≈ 115.3521 Not an integer No

73 8191 ÷ 73 ≈ 112.1918 Not an integer No

75 8191 ÷ 75 ≈ 109.2133 Not an integer No

77 8191 ÷ 77 ≈ 106.3766 Not an integer No

79 8191 ÷ 79 ≈ 103.6848 Not an integer No

81 8191 ÷ 81 ≈ 101.1235 Not an integer No

83 8191 ÷ 83 ≈ 98.6988 Not an integer No

85 8191 ÷ 85 ≈ 96.3647 Not an integer No

87 8191 ÷ 87 ≈ 94.1494 Not an integer No

89 8191 ÷ 89 ≈ 92.0225 Not an integer No

Since 8191 is not divisible by any of these odd integers, it confirms that 8191 is a prime number.

### (b) Compute n = p · q (the system modulus) and φ(n) (the Euler’s Phi-Function of n).

Compute n

n=p⋅q=719⋅8191=5889329

Compute ϕ(n)

ϕ(n)=(p−1)⋅(q−1)=(719−1)⋅(8191−1)=718⋅8190=5880420

### (c) Write the prime-power factorization of φ(n). What are the prime factors of φ(n)?

ϕ(n)=5880420

The prime-power factorization of 5880420 is: 2^2 \* 3^2 \* 5 \* 7 \* 13 \* 359

The prime factors of 5880420 are: 2, 3, 5, 7, 13, and 359.

### (d) Choose e to be a Pythagorean prime satisfying the inequality 200 < e < 240. Show that the number you picked is indeed a Pythagorean prime.

**Choose e: A Pythagorean prime satisfying 200 < e < 240**

* A Pythagorean prime is a prime number of the form 4n+1.
* Let's choose e=229 . show that 229 is a Pythagorean prime:

229=4⋅57+1

To prove that 229 is a prime number, we need to check its divisibility by all odd integers up to the square root of 229. The square root of 229 is approximately 15.1, so we only need to check for divisibility by odd numbers up to 15.

Here is a table showing the divisibility checks for odd integers from 3 to 15:

| **Integer** | **Calculation** | **Result** | **Divisible?** |
| --- | --- | --- | --- |
| 3 | 229 ÷ 3 ≈ 76.3333 | Not an integer | No |
| 5 | 229 ÷ 5 ≈ 45.8 | Not an integer | No |
| 7 | 229 ÷ 7 ≈ 32.7143 | Not an integer | No |
| 9 | 229 ÷ 9 ≈ 25.4444 | Not an integer | No |
| 11 | 229 ÷ 11 ≈ 20.8182 | Not an integer | No |
| 13 | 229 ÷ 13 ≈ 17.6154 | Not an integer | No |
| 15 | 229 ÷ 15 ≈ 15.2667 | Not an integer | No |

Since 229 is not divisible by any of these odd integers, it confirms that 229 is a prime number.

Since 229 is a prime number and can be written in the form 4n+1, it is a Pythagorean prime.

### (e) Explain, from part (c), why e and φ(n) are relatively prime. Hence, the multiplicative inverse of e in Zφ(n) exists.

**(e) Verify that eee and ϕ(n)\phi(n)ϕ(n) are relatively prime**

1. **Check if e=229 is relatively prime to ϕ(n)=5880420**
   * The prime factors of ϕ(n) are 2, 3, 5, 7, 13, and 359
   * 229 is a prime number and does not share any prime factors with ϕ(n), Therefore, the only common factor they share is 1, making them relatively prime.

### (f) Showing step-by-step details, calculate d, the multiplicative inverse of e in Zφ(n).

Given:

e=229e

ϕ(n)=5880420

We need to find dd such that:

e×d≡1 (mod ϕ(n))

This requires computing the modular multiplicative inverse of e modulo ϕ(n).

Using the Extended Euclidean Algorithm, we find dd:

Compute the GCD using the Euclidean algorithm:

5880420=25682×229+122

229=1×122+107

122=1×107+15

107=7×15+2

15=7×2+1

2=2×1+0

Since the GCD is 1, e and ϕ(n) are coprime, and the modular inverse exists.

Using the Extended Euclidean Algorithm, back substitute to find the coefficients:

1=15−7×2

2=107−7×152=107

15=122−1×107

107=229−1×122

122=5880420−25682×229

Combining these, we find:

1=2567869×229−1115520×5880420

This gives:

d=2567869

Therefore, d=2567869

To verify:

e×d=229×2567869=5880421

5880421 (mod 5880420)=1

This confirms that the relationship e×d≡1 (mod ϕ(n))e×d≡1 (mod ϕ(n)) is true.

the private exponent d is 2567869.

### (g) Summarize the integers you have obtained above in the following table.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| p | q | n | φ(n) | e | d |
| 719 | 8191 | 5889329 | 5880420 | 229 | 2567869 |

## Step (II): The RSA Encryption Process

Public Key (e, n): (229,5889329)

Private Key (d, n): (2567869,5889329)

Encryption: c=m^e mod  n

Decryption: m=c^d mod  n

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***   
 **Al-Hussein Technical University Maths for Computing**    
 **Department of Basic Sciences The RSA Algorithm**    
**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***   
  
  
**Key Generation**  
  
 Enter the value of the prime p:

719

Enter the value of the prime q:

8191

--------------------------------------------  
  
The system modulus (n) = 5889329 and phi(n) = 5880420  
  
--------------------------------------------  
  
 Enter the value of e:

229

--------------------------------------------  
  
The public key, PU = {e, n} = {229,5889329}  
  
The private key, PR = {d, n} = {2567869,5889329}  
  
--------------------------------------------  
  
**The Encryption-Decryption Process**  
  
 Enter the number of characters you need to encrypt:

13

======================================================================  
  
 Enter the ASCII code equivalent to the character (M):

73

The code you have entered is: 73  
 The encrypted value is: 4448657  
 The decrypted value is: 73  
  
  
======================================================================  
  
 Enter the ASCII code equivalent to the character (M):

32

The code you have entered is: 32  
 The encrypted value is: 2825897  
 The decrypted value is: 32  
  
  
======================================================================  
  
 Enter the ASCII code equivalent to the character (M):

108

The code you have entered is: 108  
 The encrypted value is: 1266991  
 The decrypted value is: 108  
  
  
======================================================================  
  
 Enter the ASCII code equivalent to the character (M):

111

The code you have entered is: 111  
 The encrypted value is: 3420245  
 The decrypted value is: 111  
  
  
======================================================================  
  
 Enter the ASCII code equivalent to the character (M):

118

The code you have entered is: 118  
 The encrypted value is: 4500813  
 The decrypted value is: 118  
  
  
======================================================================  
  
 Enter the ASCII code equivalent to the character (M):

101

The code you have entered is: 101  
 The encrypted value is: 4800529  
 The decrypted value is: 101  
  
  
======================================================================  
  
 Enter the ASCII code equivalent to the character (M):

32

The code you have entered is: 32  
 The encrypted value is: 2825897  
 The decrypted value is: 32  
  
  
======================================================================  
  
 Enter the ASCII code equivalent to the character (M):

34

The code you have entered is: 34  
 The encrypted value is: 5367403  
 The decrypted value is: 34  
  
  
======================================================================  
  
 Enter the ASCII code equivalent to the character (M):

82

The code you have entered is: 82  
 The encrypted value is: 1898098  
 The decrypted value is: 82  
  
  
======================================================================  
  
 Enter the ASCII code equivalent to the character (M):

83

The code you have entered is: 83  
 The encrypted value is: 5806087  
 The decrypted value is: 83  
  
  
======================================================================  
  
 Enter the ASCII code equivalent to the character (M):

65

The code you have entered is: 65  
 The encrypted value is: 4405352  
 The decrypted value is: 65  
  
  
======================================================================  
  
 Enter the ASCII code equivalent to the character (M):

34

The code you have entered is: 34  
 The encrypted value is: 5367403  
 The decrypted value is: 34  
  
  
======================================================================  
  
 Enter the ASCII code equivalent to the character (M):

33

The code you have entered is: 33  
 The encrypted value is: 5851580  
 The decrypted value is: 33  
  
  
======================================================================  
 **Have a good day**    
======================================================================

Given Values:

p=719

q=8191

n=5889329

ϕ(n)=5880420

e=229

d=2567869

RSA Encryption Process:

Key Generation:

Public Key: (e,n)=(229,5889329)

Private Key: (d,n)=(2567869,5889329)

Encryption:

Convert the Message:

Suppose the plaintext message M is the integer 42.

Convert the plaintext message M to an integer m.

Compute the Ciphertext:

Using the public key (e,n), compute the ciphertext c:

c=m^e mod  n

For m=42, e=229, and n=5889329

c=42^229mod  5889329

Perform Modular Exponentiation:

Matlab code matlab code provided

Output: 2136034

The ciphertext c is 2136034

.

Decryption:

Compute the Plaintext:

Using the private key (d,n), compute the original message m:

m=c^d mod  n

For c=2136034, d=2567869, and n=5889329:

m=2136034^2567869 mod  5889329

Perform Modular Exponentiation:

Using matlab code

Output: 42

The decrypted message m is 42.

Encryption Table

Character ASCII Code Encrypted ASCII Code

I 73 4448657

(space) 32 2825897

l 108 1266991

o 111 3420245

v 118 4500813

e 101 4800529

(space) 32 2825897

" 34 5367403

R 82 1898098

S 83 5806087

A 65 4405352

" 34 5367403

! 33 5851580

# Question 2

The code:

clc

clear

n = 63; % the number of servers

request\_n = 12000000; % the number of requests

% An example of an IPv6 address is '2001:0DB8:0000:0000:0008:0800:200C:417A'.

D = hex2dec('FFFFFFFFFFFF'); % the first 12 digits in client IP address

client\_IP\_rand = randi([0 D], 1, request\_n, 'single');

%% Example of a hash function based on mid-square

sqr\_client\_IP\_rand = client\_IP\_rand.^2; % square the client IP address

st2 = compose("%18.0f", sqr\_client\_IP\_rand); % convert number to string with more digits (extended to 18 digits)

hash\_mid\_square = zeros(1, request\_n); % Preallocate for efficiency

for i = 1:request\_n

L = strlength(st2(i));

mid1 = round(L / 2) - 2; % adjust to capture more middle digits (start 2 positions before the exact middle)

mid2 = mid1 + 3; % capture 4 digits for better distribution

st = st2{i}(mid1:mid2); % used as key, middle 4 digits of the square client IP address

hash\_mid\_square(i) = str2double(st); % convert string to number

end

hash\_mid\_square = mod(hash\_mid\_square, n); % mod by number of servers

subplot(121);

histogram(hash\_mid\_square, n); % plot histogram

title('Mid-Square Method');

xlabel('Server Index');

ylabel('Number of Requests');

mean\_mid\_square = mean(hash\_mid\_square); % calculate mean

std\_mid\_square = std(hash\_mid\_square); % calculate standard deviation

%% Example of a hash function h(x)=((ax + b) mod p) mod n

% Modular Arithmetic

p = 2^37 - 1; % a large prime number, closer to 2^37

a = 123457; % chosen constant a, large prime number

b = 678910; % chosen constant b, large prime number

hash\_f = mod(a .\* client\_IP\_rand + b, p); % Hash function based on Modular Arithmetic

hash\_f = mod(hash\_f, n); % mod by number of servers

subplot(122);

histogram(hash\_f, n); % plot histogram

title('Modular Arithmetic Method');

xlabel('Server Index');

ylabel('Number of Requests');

mean\_Modular = mean(hash\_f); % calculate mean

std\_Modular = std(hash\_f); % calculate standard deviation

% Display mean and standard deviation

disp(['Mean (Mid-Square): ', num2str(mean\_mid\_square)]);

disp(['Standard Deviation (Mid-Square): ', num2str(std\_mid\_square)]);

disp(['Mean (Modular Arithmetic): ', num2str(mean\_Modular)]);

disp(['Standard Deviation (Modular Arithmetic): ', num2str(std\_Modular)]);

## a)

### Summary of Implementations:

**Mid-square Method:**

* Squares the client IP address.
* Converts the squared value to a string.
* Extracts the middle digit(s) of the squared value.
* Computes the hash using modulo operation with the number of servers.

**Modular Arithmetic Method:**

* Uses the formula h(x) = ((ax + b) mod p) mod n with given constants a, b, and p.
* Computes the hash by performing modulo operations with a large prime number p and the number of servers n.

### Analysis:

The Mid-square method focuses on extracting middle digits from the squared client IP address, which might lead to non-uniform distribution if the middle digits do not vary sufficiently. The Modular Arithmetic method relies on the chosen constants and prime number to spread the hash values, which might also lead to clustering if the constants are not optimal.

* + .

## b) Describe why the current implementations may lead to a non-uniform distribution of requests across servers for each method.

**Mid-square Method**:

* The mid-square method may lead to non-uniform distribution due to the small number of digits used from the middle of the squared value. Small variations in the input can result in limited variation in the extracted digits, leading to clustering of hash values.

**Modular Arithmetic Method**:

* The choice of constants a, b, and p significantly impacts the distribution. If a and b are not well-chosen, or if p is not large enough, the resulting hash values may not be uniformly distributed. Small values or non-optimal primes can cause patterns and clustering.

## c) done

## **d) Explain your modifications for each method and discuss how they contribute to achieving a more uniform distribution.**

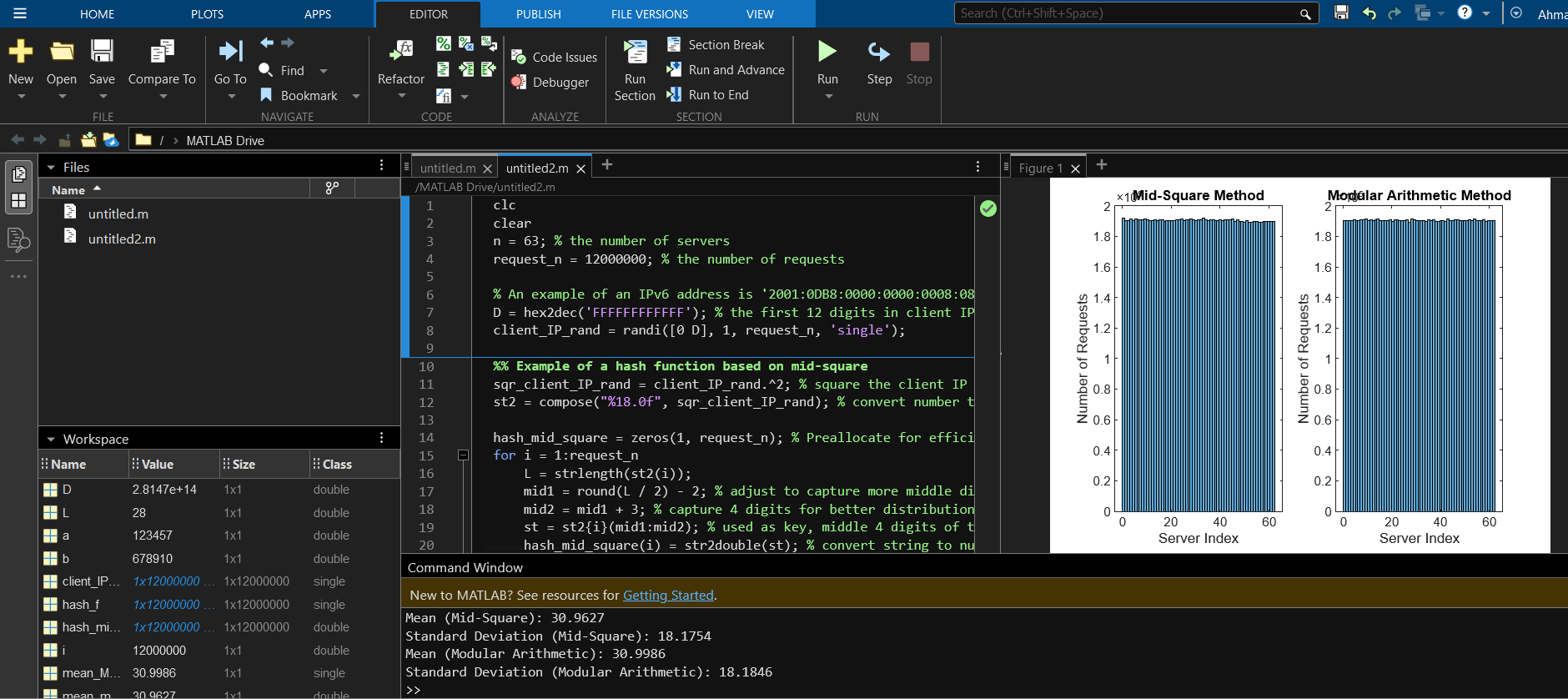
**Mid-square Method**:

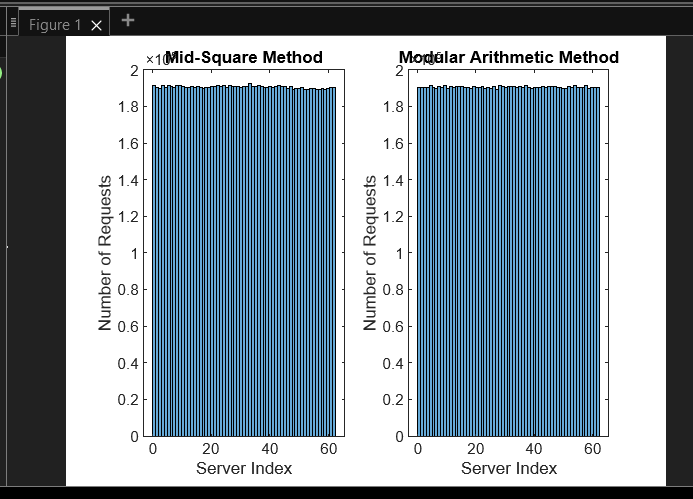
* **Modification**: Extended the string representation of squared values to 18 digits and adjusted the middle index to capture four digits around the center.
  + compose("%18.0f", sqr\_client\_IP\_rand): Extended the string representation to 18 digits to include more digits for better extraction.
  + mid1 = round(L / 2) - 2: Adjusted the middle index calculation to start 2 positions before the exact middle to capture more variability.
  + mid2 = mid1 + 3: Captured 4 digits (instead of 2) for better distribution.
  + hash\_mid\_square = mod(hash\_mid\_square, n): Ensured the resulting hash values are within the range of available servers
* **Contribution**: By capturing more digits, the resulting hash values have higher variability, reducing clustering and promoting a more uniform distribution.

**Modular Arithmetic Method**:

* **Modification**: Chose a larger prime number p closer to 2372^{37}237 and selected a and b as large primes.
* **Contribution**: Larger prime numbers and well-chosen constants help distribute the hash values more uniformly across the range, minimizing patterns and clustering.
  + p = 2^37 - 1: Chose a large prime number closer to 2372^{37}237 to ensure a better spread and avoid clustering patterns.
* a = 123457 and b = 678910: Selected a and b as large prime numbers to maximize randomness and improve the distribution of hash values

e)





## f) Compare and contrast the distributions obtained from the Mid-square method and Modular Arithmetic method, discussing their respective strengths and weaknesses in achieving uniform distribution.

**Mid-square Method**:

* **Strengths**: Simple to implement, and the extended digit extraction can improve distribution.
* **Weaknesses**: Still dependent on the variability of the input IP addresses. Large input values may lead to clustering around certain digits.

**Modular Arithmetic Method**:

* **Strengths**: More mathematically robust, especially with well-chosen parameters. Larger primes and random constants a and b help in achieving a more uniform distribution.
* **Weaknesses**: Requires careful selection of p, a, and b. If not chosen correctly, can still result in non-uniform distribution.

# Question 3:

The code :

#include <stdio.h>

#include <stdlib.h> // Include stdlib.h for abs() function

#define MATRIX\_SIZE 100 // Increased matrix size for more space

void drawRectangle(int x, int y, int width, int height, char matrix[MATRIX\_SIZE][MATRIX\_SIZE]);

void drawLine(int x1, int y1, int x2, int y2, char matrix[MATRIX\_SIZE][MATRIX\_SIZE]);

void drawCircle(int centerX, int centerY, int radius, char matrix[MATRIX\_SIZE][MATRIX\_SIZE]);

void drawEmptyCircle(int centerX, int centerY, int radius, char matrix[MATRIX\_SIZE][MATRIX\_SIZE]);

void drawVerticalLine(int x, int startY, int endY, char matrix[MATRIX\_SIZE][MATRIX\_SIZE]);

void drawHorizontalLine(int y, int startX, int endX, char matrix[MATRIX\_SIZE][MATRIX\_SIZE]);

void drawSlantedLine(int x1, int y1, int x2, int y2, char matrix[MATRIX\_SIZE][MATRIX\_SIZE]);

void displayMatrix(char matrix[MATRIX\_SIZE][MATRIX\_SIZE]);

int main() {

char matrix[MATRIX\_SIZE][MATRIX\_SIZE];

// Initialize matrix with spaces

for (int i = 0; i < MATRIX\_SIZE; i++) {

for (int j = 0; j < MATRIX\_SIZE; j++) {

matrix[i][j] = ' ';

}

}

drawRectangle(53, 3, 9, 3, matrix);

drawEmptyCircle(57, 17, 9, matrix);

drawEmptyCircle(55,17,1,matrix);

drawEmptyCircle(61,17,1,matrix);

drawCircle(55,17,1,matrix);

drawCircle(61,17,1,matrix);

drawHorizontalLine(21, 55, 65, matrix);

drawSlantedLine(55, 21, 77, 29, matrix);

drawSlantedLine(65, 21, 77, 29, matrix);

drawSlantedLine(65, 27, 77, 39, matrix);

drawSlantedLine(71, 39, 73, 41, matrix);

drawSlantedLine(69, 39, 75,45, matrix);

drawSlantedLine(67, 41, 73, 45, matrix);

drawSlantedLine(65, 43, 61, 47, matrix);

drawEmptyCircle(55,49,5,matrix);

drawSlantedLine(53, 55, 47, 61, matrix);

drawRectangle(45, 59, 3, 7, matrix);

drawRectangle(39, 59, 3, 7, matrix);

drawHorizontalLine(67, 49, 55, matrix);

drawSlantedLine(49, 65, 51, 63, matrix);

drawSlantedLine(39, 57, 37, 59, matrix);

drawVerticalLine(35, 53, 69, matrix);

drawVerticalLine(31, 55, 69, matrix);

drawHorizontalLine(71, 31, 35, matrix);

drawVerticalLine(27, 57, 75, matrix);

drawVerticalLine(23, 59, 75, matrix);

drawHorizontalLine(77, 23, 27, matrix);

drawVerticalLine(19, 61, 69, matrix);

drawVerticalLine(15, 63, 69, matrix);

drawHorizontalLine(71, 15, 19, matrix);

drawSlantedLine(13, 65, 9, 69, matrix);

drawSlantedLine(5, 69, 17, 55, matrix);

drawSlantedLine(17, 53, 7, 55, matrix);

drawSlantedLine(5, 55, 21, 39, matrix);

drawSlantedLine(21, 39, 33, 51, matrix);

drawSlantedLine(33, 51, 47, 17, matrix);

drawSlantedLine(21, 39, 47, 17, matrix);

// Display matrix

displayMatrix(matrix);

return 0;

}

void drawRectangle(int x, int y, int width, int height, char matrix[MATRIX\_SIZE][MATRIX\_SIZE]) {

for (int i = y; i < y + height; i++) {

for (int j = x; j < x + width; j++) {

matrix[i][j] = '\*';

}

}

}

void drawCircle(int centerX, int centerY, int radius, char matrix[MATRIX\_SIZE][MATRIX\_SIZE]) {

for (int y = 0; y < MATRIX\_SIZE; y++) {

for (int x = 0; x < MATRIX\_SIZE; x++) {

if ((x - centerX) \* (x - centerX) + (y - centerY) \* (y - centerY) <= radius \* radius) {

matrix[y][x] = '\*';

}

}

}

}

void drawEmptyCircle(int centerX, int centerY, int radius, char matrix[MATRIX\_SIZE][MATRIX\_SIZE]) {

for (int y = 0; y < MATRIX\_SIZE; y++) {

for (int x = 0; x < MATRIX\_SIZE; x++) {

// Check if the point is on the circumference of the circle using circle equation

if ((x - centerX) \* (x - centerX) + (y - centerY) \* (y - centerY) == radius \* radius) {

matrix[y][x] = '\*'; // Mark only the boundary points

}

}

}

}

void drawVerticalLine(int x, int startY, int endY, char matrix[MATRIX\_SIZE][MATRIX\_SIZE]) {

for (int i = startY; i <= endY; i++) {

matrix[i][x] = '\*';

}

}

void drawHorizontalLine(int y, int startX, int endX, char matrix[MATRIX\_SIZE][MATRIX\_SIZE]) {

for (int i = startX; i <= endX; i++) {

matrix[y][i] = '\*';

}

}

void drawSlantedLine(int x1, int y1, int x2, int y2, char matrix[MATRIX\_SIZE][MATRIX\_SIZE]) {

int dx = abs(x2 - x1);

int dy = abs(y2 - y1);

int sx = x1 < x2 ? 1 : -1;

int sy = y1 < y2 ? 1 : -1;

int err = (dx > dy ? dx : -dy) / 2;

int e2;

while (1) {

matrix[y1][x1] = '\*';

if (x1 == x2 && y1 == y2) break;

e2 = err;

if (e2 > -dx) { err -= dy; x1 += sx; }

if (e2 < dy) { err += dx; y1 += sy; }

}

}

void displayMatrix(char matrix[MATRIX\_SIZE][MATRIX\_SIZE]) {

for (int i = 0; i < MATRIX\_SIZE; i++) {

for (int j = 0; j < MATRIX\_SIZE; j++) {

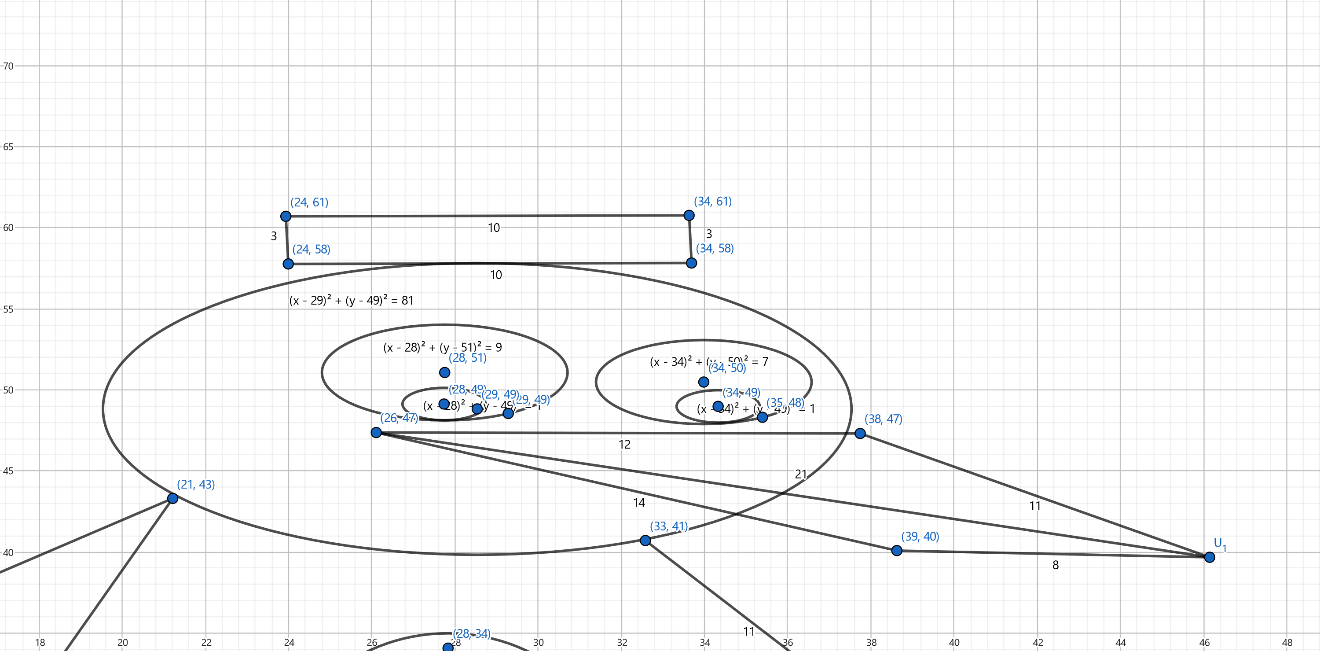
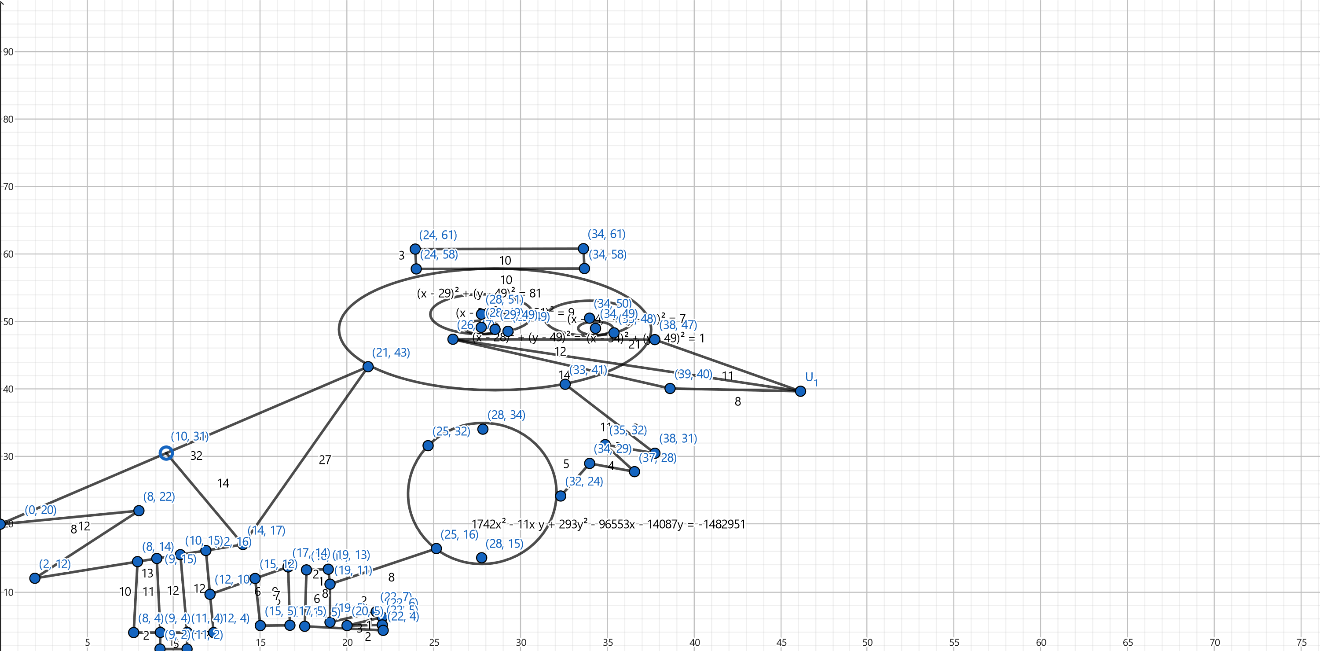
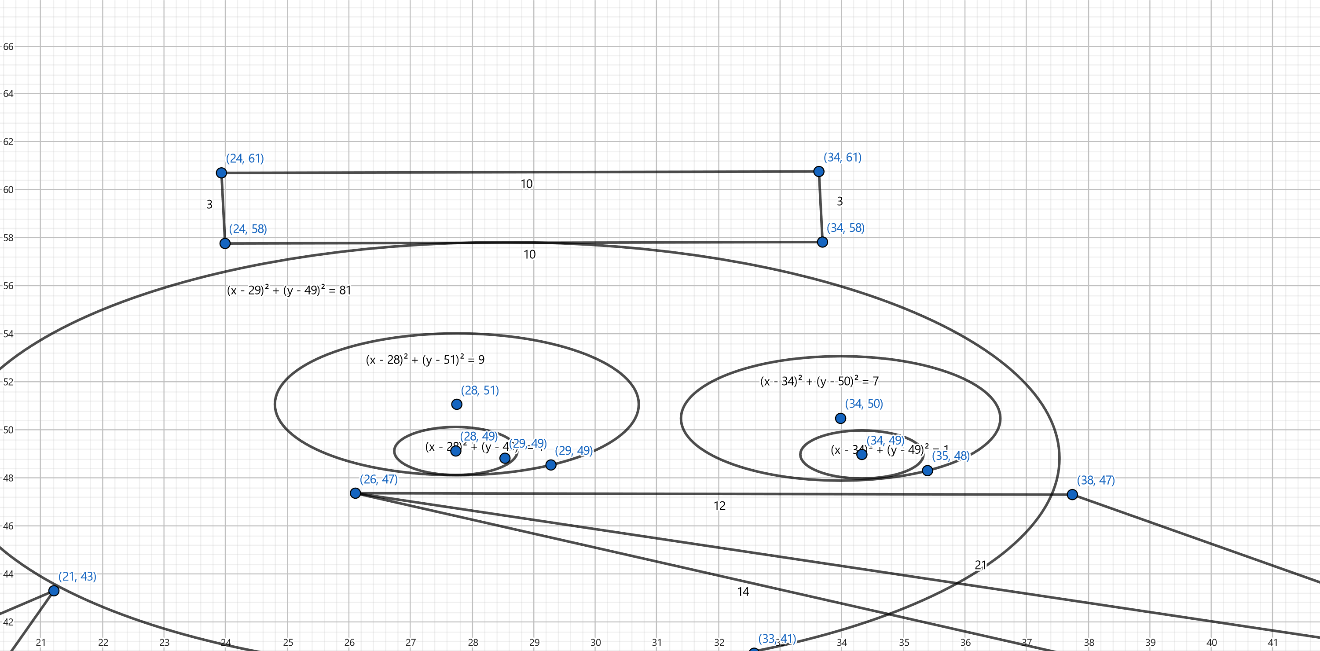
printf("%c ", matrix[i][j]); // Add a space between characters

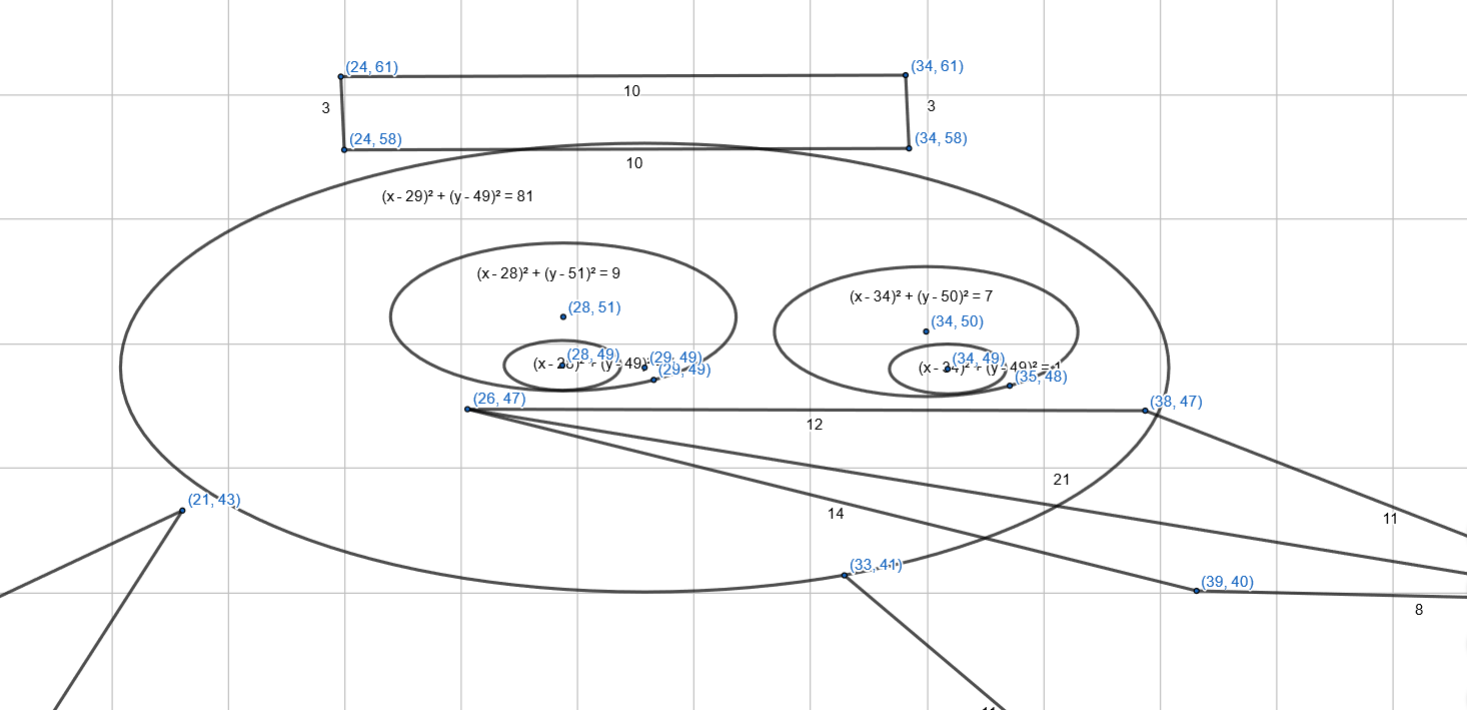
}

printf("\n");

}

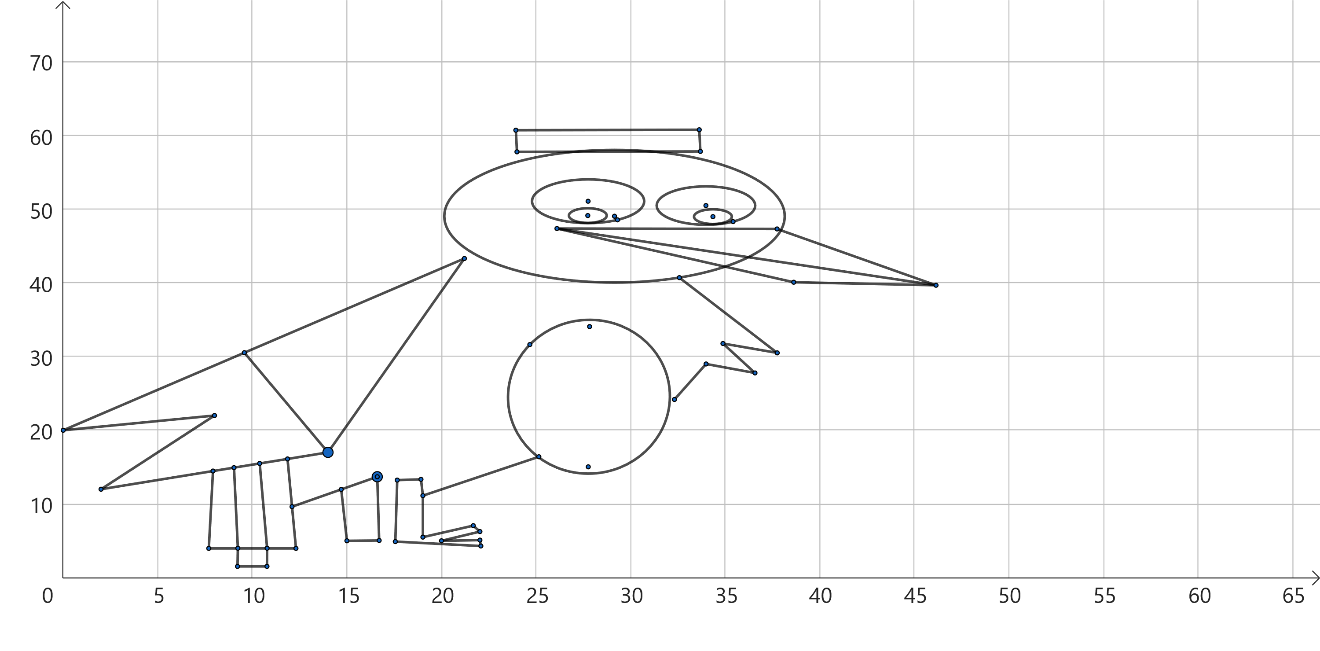
}





A graph of a diagram

Description automatically generated with medium confidence



Press the last bottom on the right to copy the output then past it on a text file

Zoom out to see the full result

