CTIS 496

Computer and Network Security

HW3

Özlem Kılıçkıran

Emircan Kılıçaslan

Q1-a)

Q1-a) Affine Cipher Formula for the first guess ($X \to E$)

$E(x) = (ax+b) \pmod{26}$ encryption

$D(y) = a^{-1}(y-b) \pmod{26}$ decryption

Applying frequency analysis :

The most frequent letter in the ciphertext is "X" with 1.12.3 ($X=23$)

The most frequent ~~cipher text~~ letter in English is 'E' ($E=4$)

(1) $4a + b \equiv 23 \mod 26$

(2) $19a + b \equiv 5 \mod 26$

Substract (1) from (2) :

$19a - 4a \equiv (5-23) \mod 26$

$15a \equiv -18 \equiv 8 \mod 26$

$15a \equiv 8 \mod 26$

Inverse of 15 mod 26 $15^{-1} \mod 26$. For $x=7$,

$15 \cdot 7 = 105 \mod 26 = 1$.

$15^{-1} \equiv 7 \mod 26$

Multiple both sides by 7 :

$a \equiv 7.8 = 56 \mod 26 = 4$

$a = 4$

Using the equation $4a + b \equiv 23 \mod 26$

$4.4 + b \equiv 23$

$16 + b \equiv 23$

$b \equiv 7$

Final key :

$K = (a, b) = (4, 7)$

Q1-a)

Q1) Second Guess (W → T)

Second most frequent ciphertext letter is 'W' with i:9 (W = 22)
Second most frequent letter in English is 'T' (T = 19)

$$19a + b \equiv 22 \mod 26$$

(1) $4a + b \equiv 23 \mod 26$

(2) $19a + b \equiv 22 \mod 26$

Substract (1) from (2) :

$$(19a + 4a) + (b - b) \equiv 22 - 23 \mod 26$$

$$15a \equiv -1 \mod 26$$

$$15a \equiv 25 \mod 26$$

Solve for a using modular inverse to find the inverse of 15 mod 26 :

for $a^{-1} = 7$

$15 \cdot 7 = 105 \mod 26 = 1$

Multiply both sides :

$a = 7 \cdot 25 = 175 \mod 26 = 19$

$a = 19$

Using $4a + b \equiv 23 \mod 26$:

$4 \cdot 19 + b \equiv 23$

$76 + b \equiv 23 \mod 26$

$b \equiv 23 - 76 = -53 \mod 26 = 25$

$b = 25$

So, using letter frequencies (most common X = 23 likely maps to E = 4 . Making the second assumption, W = 22 likely maps to T = 19.

a is valid (gcd(19, 26) = 1), and it has an inverse $a^{-1} = 11 \mod 26$

Q1-b)

Q1-b) Applying Affine decryption formula to the ciphertext

$$D(y) = a^{-1} \cdot (y-b) \bmod 26 = 11 \cdot (y-25) \bmod 26$$

We convert letters to numbers (A=0, B=1 ... Z=25), then apply the decryption formula, then convert back to letters. We do this repeatedly until plaintext becomes readable.

Q2)

Q2) a) plaintext : A B C D E H I J L N O R S T U Y

key : J T H N S A C I B L E U D U C R

b) The number of possible keys are 16 ! since a substitution Ciphet key is a permutation of 16 letters.

c) chosen plaintext : HOUSE

Ciphertext : AECDS

Q3)

Q3- Özlem

Plaintext: r e c o r d o t o r y

Indices: 0 1 2 3 4 5 6 7 8 9 10

Ciphertext: r d + r r c a e o y o

Indices: 0 1 2 3 4 5 6 7 8 9 10

Possible Mappings

Plain → Cipher

r(0,4,9) → r(0,3,4)

e(1) → e(7)

c(2) → c(5)

o(3,8) → o(8,10)

d(5) → d(1)

a(6) → a(6)

+(7) → +(2)

y(10) → y(9)

6 Candidate keys

1- 0 7 5 8 3 1 6 2 10 9

2- 0 7 5 4 3 1 6 2 10 9

3- 3 7 5 0 4 1 6 2 10 9

4- 3 7 5 4 0 1 6 2 10 9

5- 4 7 5 0 3 1 6 2 10 9

6- 4 7 5 4 3 1 6 2 10 9

Q3- Emircan

). tetraborate tebrroteoto

- plaintext = tetraborate
- Ciphertext= tebrroteoto

→ This is a Permutation Cipher with known plaintext Attack (KPA)

Plaintext	t	e	t	r	a	b	o	r	a	t	e
Indices	0	1	2	3	4	5	6	7	8	9	10

Ciphertext	t	e	b	r	r	o	t	e	o	t	o
Indices	0	1	2	3	4	5	6	7	8	9	10

- t (at 0,2,9 in plaintext) can map to t (at 0,6,8 in ciphertext)
- e (at 1,10 in plaintext) can map to e (at 1,7 in ciphertext)
- r (at 3,7 in plaintext) can map to r (at 3,4 in ciphertext)
- a (at 4,8 in plaintext) can map to a (at 5,10 in ciphertext)
- b (at 5 in plaintext) must map to b (at 2 in ciphertext)
- o (at 6 in plaintext) must map to o (at 9 in ciphertext)

6 Candidate keys

1- 0 1 6 3 5 2 9 4 10 8 7

2- 8 10 3 5 2 9 4 10 6 7

3- 0 7 6 3 5 2 9 4 10 8 1

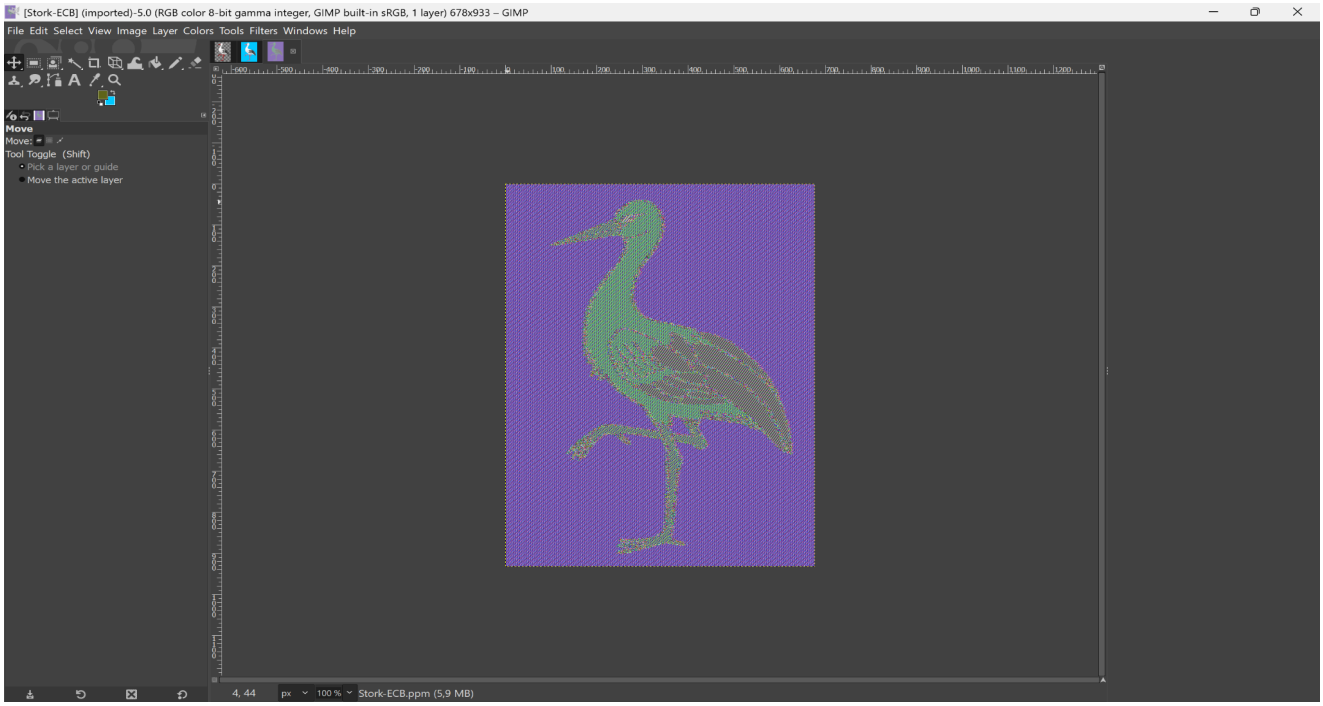
4- 8 7 0 3 5 2 9 4 10 6 1

5- 0 1 6 4 5 2 9 3 10 8 7

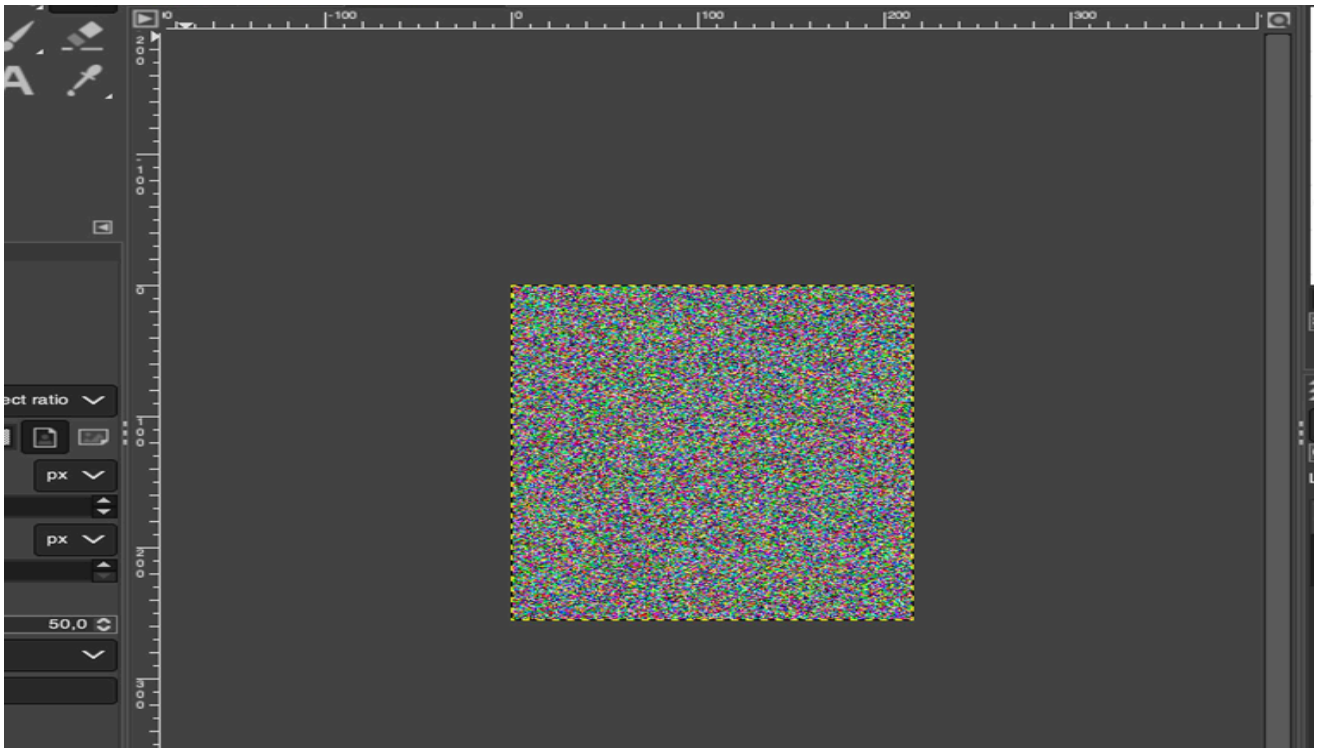
6- 8 10 4 5 2 9 3 10 6 1

Q4)

ECB MODE



CBC MODE



Encryption of Stork.ppm with:

- **AES block cipher with key size 128 in ECB mode**
- **AES block cipher with key size 128 in CBC mode.**

ECB (Electronic Codebook) Encryption:

The output file Stork-ECB.ppm still displayed recognizable shapes and patterns from the original image when opened in GIMP. Although the image was encrypted, the structure of the original image was still partially visible.

This occurred because ECB encrypts identical blocks into identical ciphertext blocks, which leaks visual information when used on structured data like images.

CBC (Cipher Block Chaining) Encryption:

The output file Stork-CBC.ppm appeared as complete visual noise or distortion when opened in GIMP. No visible shapes or patterns from the original image could be recognized.

This is because CBC mode chains encryption: each block is encrypted using the ciphertext of the previous block, which ensures that even identical plaintext blocks produce different ciphertext.

OBSERVATIONS

ECB mode is **not secure** for encrypting images or structured data, as it fails to fully hide repeating patterns. It should not be used for any sensitive visual or patterned data.

CBC mode is much **more secure** than ECB for encrypting images, as it hides patterns and produces ciphertext that looks random, making visual analysis ineffective.

Q5)

This C program is a safe, annotated implementation of a sorting and random selection task that demonstrates good secure coding practices following CERT C coding rules.
Following SEI CERT Coding Standards codes are used for the C program:

1. **PRE30-C:** Do not create a null pointer dereference.
2. **PRE31-C:** Avoid dangerous macros.
3. **MSC30-C:** Do not use the rand() function for generating random numbers.
4. **ARR36-C:** Do not subtract or compare pointers that do not refer to the same array.
5. **MEM30-C:** Do not access freed memory.

main.c

Share Run

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Rule PRE31-C: Avoid dangerous macros
// Instead of a macro, use an inline function for swapping
static inline void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main() {
    int n;

    printf("Enter the number of elements: ");

    // Rule PRE30-C: Ensure no null pointer dereference
    // Always check the validity of pointers before using them
    if (scanf("%d", &n) != 1 || n <= 0) {
        fprintf(stderr, "Invalid input.\n");
        return 1;
    }

    // Rule MEM30-C: Do not access freed memory
    // Dynamically allocate memory and ensure it is freed appropriately
    int *arr = (int *)malloc(n * sizeof(int));
    if (arr == NULL) {
        fprintf(stderr, "Memory allocation failed.\n");
        return 1;
    }
```

main.c

Share Run

```c
    printf("Enter %d integers:\n", n);
    for (int i = 0; i < n; i++) {
        if (scanf("%d", &arr[i]) != 1) {
            fprintf(stderr, "Invalid input.\n");
            free(arr); // Free memory if input fails
            return 1;
        }
    }
    // Bubble sort implementation
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            // Rule ARR36-C: Do not compare pointers outside the same array
            // arr[j] and arr[j + 1] are within the bounds of the same array
            if (arr[j] > arr[j + 1]) {
                swap(&arr[j], &arr[j + 1]);
            }
        }
    }
    // Rule MSC30-C: Do not use rand() for generating random numbers
    // Example of using a better random number generator with seed
    srand((unsigned int)time(NULL)); // Seeding
    int random_index = rand() % n;

    printf("Sorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    printf("Randomly selected element: %d\n", arr[random_index]);

    // Free memory to prevent memory leaks
    free(arr);
    return 0;
```

Output

Clear

```
Enter the number of elements: 5
Enter 5 integers:
11 37 2 89 66
Sorted array: 2 11 37 66 89
Randomly selected element: 89


=== Code Execution Successful ===
```

Explanation of SEI CERT Coding Standards Applied

1. **PRE30-C: Null Pointer Dereference**
    - Checked the return value of scanf and the validity of malloc to prevent null pointer dereferencing.
2. **PRE31-C: Avoid Dangerous Macros**
    - Used an inline function (swap) instead of a macro for better safety and debugging.
3. **MSC30-C: Do not use rand() for generating random numbers**
    - Used srand with time for proper seeding to ensure better random number generation.
4. **ARR36-C: Do not subtract or compare pointers outside the same array**
    - All pointer comparisons (arr[j] > arr[j + 1]) are within the bounds of the array.
5. **MEM30-C: Do not access freed memory**
    - Ensured memory allocated with malloc is freed after use, and memory access is carefully handled.

References

Carnegie Mellon University Software Engineering Institute. (2024, September 19). *SEI CERT Coding Standards*. Retrieved from https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards