Topic: Security Tools - Error detection codes

Names of Students:
Alfiya Anware - 16010121006
Arra Alekya - 16010121007
Azhar Bamne - 16010121011

Introduction:
The two error detection methods that we will be implementing are Parity Bit method and Checksum. The Parity Bit Method is a basic error detection technique used in digital communication systems, involving the addition of an extra bit to binary data to ensure either even or odd parity. This allows the receiver to detect single-bit errors during transmission, making it suitable for applications where simplicity and minimal overhead are prioritized. Conversely, Checksums provide a more robust error detection mechanism, generating a checksum value based on mathematical operations applied to data blocks. Widely employed in network protocols, file transfers, and storage systems, checksums offer enhanced error detection capabilities, capable of identifying not only single-bit errors but also burst errors, making them essential for ensuring data integrity in diverse digital communication scenarios.

List of features:

Parity Bit Method:

- Simple Implementation: The Parity Bit Method is straightforward to implement, involving the addition of a single bit to binary data.
- Even/Odd Parity Selection: Allow users to choose between even parity and odd parity based on their specific error detection requirements.
- Single-Bit Error Detection: It is primarily designed to detect single-bit errors in transmitted data, making it suitable for applications where error detection requirements are minimal.

Checksum:

- Robust Error Detection: Checksums provide robust error detection capabilities, capable of detecting not only single-bit errors but also burst errors and other common transmission errors.
- Block-Level Verification: Checksums operate on blocks of data rather than individual bits, ensuring integrity across larger data units.

Methodology:

Parity-bit Method

Code:

```
def calculate_parity(data, parity_type='even'):
    ones_count = sum(int(bit) for bit in data)
    if parity_type == 'even':
        if ones_count % 2 == 0:
            return '0'
        else:
            return '1'
    elif parity_type == 'odd':
        if ones_count % 2 == 0:
            return '1'
        else:
            return '0'

def add_parity_bit(data, parity_type='even'):
    parity_bit = calculate_parity(data, parity_type)
    return data + parity_bit

def check_parity(data, parity_type='even'):
    received_parity_bit = data[-1]
    calculated_parity_bit = calculate_parity(data[:-1], parity_type)
    if received_parity_bit == calculated_parity_bit:
        return "Parity check passed: No error detected"
    else:
        return "Parity check failed: Error detected"

def menu():
    print("1. Even Parity")
    print("2. Odd Parity")
    print("3. Exit")

while True:
    menu()
    choice = input("Enter your choice: ")

    if choice == '1':
        binary_data = input("Enter binary codeword: ")
```

```
        codeword = add_parity_bit(binary_data, 'even')
        print("Codeword with even parity bit:", codeword)

        while True:
            received_data = input("Enter received codeword to cross-check
(or type 'exit' to quit): ")
            if received_data.lower() == 'exit':
                print("Exiting...")
                break
            result = check_parity(received_data, 'even')
            print(result)

    elif choice == '2':
        binary_data = input("Enter binary codeword: ")
        codeword = add_parity_bit(binary_data, 'odd')
        print("Codeword with odd parity bit:", codeword)

        while True:
            received_data = input("Enter received codeword to cross-check
(or type 'exit' to quit): ")
            if received_data.lower() == 'exit':
                print("Exiting...")
                break
            result = check_parity(received_data, 'odd')
            print(result)

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

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

Output:

```
1. Even Parity
2. Odd Parity
3. Exit
Enter your choice: 1
Enter binary codeword: 101101
Codeword with even parity bit: 1011010
Enter received codeword to cross-check (or type 'exit' to quit): 100101
Parity check failed: Error detected
Enter received codeword to cross-check (or type 'exit' to quit): 101101
Parity check passed: No error detected
Enter received codeword to cross-check (or type 'exit' to quit): 100001
Parity check passed: No error detected
Enter received codeword to cross-check (or type 'exit' to quit): exit
Exiting...
1. Even Parity
2. Odd Parity
3. Exit
Enter your choice: 2
Enter binary codeword: 111
Codeword with odd parity bit: 1110
Enter received codeword to cross-check (or type 'exit' to quit): 1101
Parity check passed: No error detected
Enter received codeword to cross-check (or type 'exit' to quit): 1110
Parity check passed: No error detected
Enter received codeword to cross-check (or type 'exit' to quit): 0110
Parity check failed: Error detected
Enter received codeword to cross-check (or type 'exit' to quit): exit
Exiting...
1. Even Parity
2. Odd Parity
3. Exit
Enter your choice: 3
Exiting...
```

Checksum

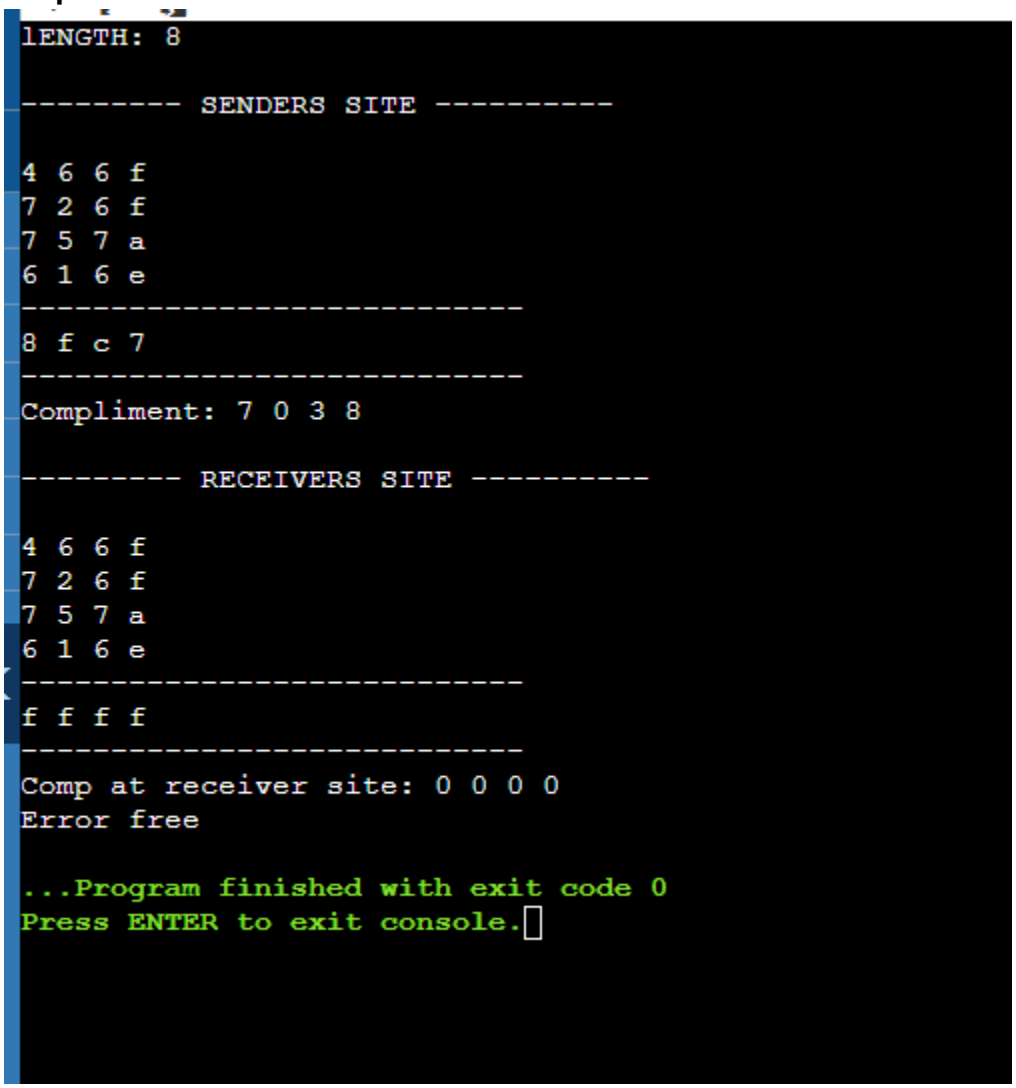
Code:

```
#include<stdio.h>
#include<string.h>
int main(){
    char ch[8]="Forouzan";
    int n=strlen(ch);
    printf("lENGTH: %d", n);
    printf("\n\n--------- SENDERS SITE ----------\n");
    int sum[4];
    for(int i=0;i<n;i=i+2)
    {
        int n1=ch[i];
        int n2=ch[i+1];

        int q1=n1/16;
        int q2=n2/16;
        int r1=n1%16;
        int r2=n2%16;
        sum[0]+=q1;
        sum[1]+=r1;
        sum[2]+=q2;
        sum[3]+=r2;
        printf("\n%x %x %x %x",q1,r1,q2,r2);
    }

    for(int i=3;i>=0;i--)
    {
        int q=sum[i]/16;
        int r=sum[i]%16;

        if(i>0) sum[i-1]+=q;
        else sum[3]+=q;
        sum[i]=r;
    }
    printf("\n----------------------------");
        printf("\n%x %x %x %x",sum[0],sum[1],sum[2],sum[3]);

    for(int i=3;i>=0;i--)
        sum[i]=15-sum[i];
```

```
printf("\n----------------------------");
printf("\nCompliment: %x %x %x %x",sum[0],sum[1],sum[2],sum[3]);

        printf("\n\n--------- RECEIVERS SITE ----------\n");

    int sum2[4];
for(int i=0;i<n;i=i+2)
{
    int n1=ch[i];
    int n2=ch[i+1];

    int q1=n1/16;
    int q2=n2/16;
    int r1=n1%16;
    int r2=n2%16;
    sum2[0]+=q1;
    sum2[1]+=r1;
    sum2[2]+=q2;
    sum2[3]+=r2;
    printf("\n%x %x %x %x",q1,r1,q2,r2);
}
sum2[0]+=sum[0];
sum2[1]+=sum[1];
sum2[2]+=sum[2];
sum2[3]+=sum[3];



for(int i=3;i>=0;i--)
{
    int q=sum2[i]/16;
    int r=sum2[i]%16;

    if(i>0) sum2[i-1]+=q;
    else sum2[3]+=q;
    sum2[i]=r;
}
printf("\n----------------------------");
printf("\n%x %x %x %x",sum2[0],sum2[1],sum2[2],sum2[3]);
for(int i=3;i>=0;i--)
```

```
        sum2[i]=15-sum2[i];
    printf("\n----------------------------");
                printf("\nComp    at    receiver    site:    %x    %x    %x
%x",sum2[0],sum2[1],sum2[2],sum2[3]);
    if(sum2[2]==0 && sum2[1]==0 && sum2[0]==0 && sum2[3]==0){
        printf("\nError free");
    }
    else {
        printf("\nError detected");
    }
}
```

Output:



Results:

In this report, we investigated three commonly used methods for error detection in digital data transmission: parity bit method, checksum, and error detection codes. Each method has its own strengths and weaknesses, which we explored in detail.

Parity Bit Method:

- The parity bit method involves adding an extra bit to each transmitted byte to ensure that the total number of bits set to 1 is either even (even parity) or odd (odd parity).
- It is a simple and easy-to-implement method that can detect single-bit errors.
- However, it has limitations in detecting multiple-bit errors, and it cannot correct errors, only detect them.
- Despite its simplicity, the parity bit method is still widely used in applications where error detection needs to be lightweight and efficient.

Checksum:

- Checksum is a technique that involves summing up the data bytes and appending the result as a checksum value at the end of the transmission.
- It provides a higher level of error detection compared to the parity bit method, as it can detect not only single-bit errors but also some multiple-bit errors.
- However, checksums have limitations in detecting certain types of errors, such as transposition errors, and they cannot correct errors either.
- Checksums are commonly used in network protocols like UDP and IP to ensure data integrity during transmission.

Conclusion:

In conclusion, each of the error detection methods discussed in this report offers different levels of reliability and complexity. The choice of method depends on factors such as the desired level of error detection, computational resources available, and the specific requirements of the application. For simple applications where lightweight error detection is sufficient, the parity bit method can be a suitable choice due to its simplicity and low computational overhead. Checksums offer a higher level of error detection compared to parity bits and are commonly used in network protocols and file transfer applications.