OVERALL PROGRAM DESIGN

PART A – SIMPLE DIFFIE HELLMAN PUBLIC KEY ENCRYPTION

1. Client prompt enter two prime which a store in a dictionary which are shared publicly
2. Using the two primes that were shared both the client and the server. Both party computer their public key using a shared primes and a randomly generated secret keys.
3. These public keys then used to calculate a symmetric key which is private to both parties.
4. This symmetric key is then used as the key for a simplified AES encryption to communicate with each other.
5. A nonce was encrypted by the client using AES and Symmetric Key (Shared Secret Key) and sent to the server.
   The Server then decrypts and subtracts 5 encrypted then send back to client. The client then decrypts and compares the original nonce less 5 to see if the communication is sent from the Server.
6. After this validation then the On Time Pad is completed after which both parties starts the lair's dice game.

PART C –ONE TIME PAD ENCRYPTION

1. The server converts the symmetric key to a 15 bit binary and a random binary key of the same length was generated.
2. Each bit of the same position in the symmetric key and generated key was combined using the exclusive OR (XOR) operation to generate the cipher text.
3. The cipher text was sent to client for decryption. The generated key was communicated between both parties privately.
4. The client then decrypts the cipher using the generated to which results in the original plaintext or symmetric that was sent to the server.
5. Program was design to allow the generated key to be entered in either base 2 and 10.
6. Connect was not disconnected if key was incorrect, only a message to let the user know that decryption was not correct. Validation was already done in part A and this was just a test of one time padding.

Server (client implementation was similar to server both inverses in some the functions outline below):

1. The server would listen using an infinite loop until a message is received from the client if not a message loop break
2. After a message is received the message would be processed using a process message function
3. The process message function is then use to execute code base on the messages sent by the client. Example: Reply to client and say hello, roll on clients request and bid or challenge the request of client.
4. Lists were implemented globally to store the rolls and bids of both Server and Client. This was done so the list weren't override with each cycle of the while loop to receive message from the client.
5. Function to turn message to Dice were implemented to update the lists each a bid was made dice roll was sent. The list was emptied at the start of the function and new elements would append creating the new list.
6. Bid was design to take the face value of the dice and the frequency of it appearance. If the bid fulfils the criteria a message would be send to the client with the bid if the bid is invalid a. Value greater than 6, Frequency greater than 10 and If neither the value nor frequency of the bid is larger than the client's bid. Then the system is design to automatically challenge the client prior bid.
7. Challenge was design to take the client bid and challenge it be used a checkBoard function which count the face value in the service roll and client roll and compare with frequency. If the frequency is less than or equal to the count then the function returns True to the challenge function. Which will display client as winners else the server win the challenge.
8. After the check board the winner was communicated to the other party after which a status not equal to 1 was return to end the game/ connection between client and server.
9. Prompt to have the user press enter before closure of connect was added to validate gaming session before it is completely closed.