```java
import java.io.*;
import java.math.BigInteger;
import java.net.*;
import java.security.*;
import javax.crypto.*;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import java.nio.charset.StandardCharsets;

public class DHClient {

    public static int keyLength = 128;

    public static void main(String[] args) throws IOException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException, IllegalBlockSizeException, BadPaddingException, C

        SecureRandom sr = new SecureRandom();
        // This line creates an instance of SecureRandom.
        //  SecureRandom is used to generate cryptographically strong random numbers.

        while (true) {
            BigInteger q = new BigInteger(keyLength, 10, sr);
            BigInteger a = new BigInteger(keyLength - 1, sr);
            BigInteger xa = new BigInteger(keyLength - 1, sr);
            BigInteger ya = a.modPow(xa, q);

//             Inside the infinite loop, the code generates four BigInteger values:
// q: A large prime number generated with a bit length of keyLength.
// a: A random number generated with a bit length of keyLength - 1.
// xa: Another random number generated with a bit length of keyLength - 1.
// ya: Calculated as a.modPow(xa, q), which represents the public key of the client.
            System.out.println("##################################################");
            System.out.println("Enter host name or IP of server");
            System.out.println("##################################################");
            String host = new BufferedReader(new InputStreamReader(System.in)).readLine();
            Socket link = new Socket(host, 11111);

            // The user is prompted to enter the host name or IP address of the server. A Socket named
            // link is then created by connecting to the server using the specified
            // host and port number (11111).

            BufferedReader in = new BufferedReader(new InputStreamReader(link.getInputStream()));
            PrintStream out = new PrintStream(link.getOutputStream());

            out.println(q);
            out.println(a);
            out.println(ya);
// Input and output streams (in and out) are set up
// to communicate with the
// server through the socket. The values q, a, and ya are
// sent to the server using the out stream.
            BigInteger yb = new BigInteger(in.readLine());
            BigInteger key = yb.modPow(xa, q);
            System.out.println("##################################################");
            System.out.println("The secret key is:\n" + key);
            System.out.println("##################################################");
            // The server's public key (yb) is received from the server, and the shared secret
            // key (key) is computed using yb.modPow(xa, q). The computed key is then printed.

            BufferedReader userEntry = new BufferedReader(new InputStreamReader(System.in));
            // Creates a BufferedReader named userEntry to read input from the console.

            while (true) {
//                 Inside an infinite loop, the following steps are performed for encryption:
// A new Cipher instance is created using the AES algorithm in CBC mode with PKCS5 padding.
// The shared secret key (key) is converted to a byte array and used to create a SecretKeySpec.
// An initialization vector (IV) is created with the same block size as the cipher.
// The cipher is initialized in encryption mode with the shared secret key and IV.
// The user is prompted to enter a message to be encrypted. If the message is "exit," the loop breaks.
                try {
                    // Encryption
                    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
                    byte[] sharedKeyBytes = key.toByteArray();
                    SecretKeySpec sharedSecretKey = new SecretKeySpec(sharedKeyBytes, "AES");

                    // Add IV (Initialization Vector)
                    byte[] ivBytes = new byte[cipher.getBlockSize()];
                    IvParameterSpec ivSpec = new IvParameterSpec(ivBytes);

                    cipher.init(Cipher.ENCRYPT_MODE, sharedSecretKey, ivSpec);

                    // Input message to be encrypted
                    System.out.println("###################################################");
                    System.out.println("Enter the message to be encrypted (type 'exit' to quit):");
                    System.out.println("###################################################");
                    String message = userEntry.readLine();

                    // Check for exit condition
                    if ("exit".equalsIgnoreCase(message)) {
                        break;
                    }

                    byte[] encryptedMessage = cipher.doFinal(message.getBytes(StandardCharsets.UTF_8));

                    // Display the encrypted version of the message
                    System.out.println("###################################################");
                    System.out.println("Sent encrypted message: " + new String(encryptedMessage, StandardCharsets.UTF_8));
                    System.out.println("###################################################");

                    // The entered message is encrypted using cipher.doFinal(),
                    //  and the encrypted bytes are printed to the console.

                    ObjectOutputStream outputStream = new ObjectOutputStream(link.getOutputStream());
                    outputStream.writeObject(encryptedMessage);
//                     An ObjectOutputStream named outputStream is created to write
//                      objects to the output stream of the socket (link).
// The encrypted message (encryptedMessage) is written to the output
//  stream using outputStream.writeObject().

                    // Receive and decrypt the reply message
                    Cipher replyCipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
                    // PKCS5Padding is a padding scheme used in cryptographic
                    //  operations, particularly in block cipher modes such as
                    //   Cipher Block Chaining (CBC) mode. Padding is necessary when the
                    //    length of the data to be encrypted is not a multiple of the block
                    //     size of the cipher. In the case of AES (Advanced Encryption Standard),
                    //      which has a block size of 128 bits (16 bytes),
                    //  PKCS5Padding adds padding to the plaintext so that its length
                    //  becomes a multiple of the block size.
                    replyCipher.init(Cipher.DECRYPT_MODE, sharedSecretKey, ivSpec);
                    // A new Cipher instance is created for decryption, using the AES algorithm
                    //  in CBC mode with PKCS5 padding. The replyCipher is initialized in decryption
                    // mode using the shared secret key (sharedSecretKey) and the same IV (ivSpec) used for encryption.

                    ObjectInputStream replyInputStream = new ObjectInputStream(link.getInputStream());
                    byte[] encryptedReply = (byte[]) replyInputStream.readObject();

//                     An ObjectInputStream named replyInputStream is created to
//                     read objects from the input stream of the socket (link).
// The encrypted reply message (encryptedReply)
// is read from the input stream using replyInputStream.readObject().

                    byte[] decryptedReply = replyCipher.doFinal(encryptedReply);
                    String replyMessage = new String(decryptedReply, StandardCharsets.UTF_8);
            // The encrypted reply message is decrypted using replyCipher.doFinal(), and the decrypted bytes
            //  are converted to a string (replyMessage) using UTF-8 encoding.
                    // Display the decrypted reply message
                    System.out.println("##################################################");
                    System.out.println("Received encrypted reply: " + new String(encryptedReply, StandardCharsets.UTF_8));
                    System.out.println("##################################################");

                    System.out.println("##################################################");
                    System.out.println("Decrypted reply: " + replyMessage);
                    System.out.println("##################################################");
                    // The received encrypted reply message and the
                    //  corresponding decrypted message are printed to the console.
                } catch (EOFException e) {
                    System.out.println("##################################################");
                    System.out.println("Server connection closed unexpectedly.");
                    break;
                }
            }

            // Close resources for this client
            link.close();
            in.close();
            out.close();
        }
    }
}
```