CS101
Discrete Mathematics

Title: Implementation of Diffie-Hellman key exchange algorithm

Group Members:

- **Asad Alam** 2021CSB1271
- **Devansh Rawat** 2022CSB1076
- **Jai Anurag Y** 2022CSB1086
- **Parth Varma** 2021MCB1255

Objective:

To get an insight into the implementation and understanding the Diffie-Hellman key exchange algorithm

Individual Contributions:

- **Asad Alam** Code Contribution, presentation and editing
- **Devansh Rawat** Report Contribution and Research
- **Jai Anurag Y** Code Contribution and Algo design
- **Parth Varma** Report Contribution and Research

Implementation of Diffie-Hellman key exchange problem:

Abstract:

The Diffie-Hellman key exchange algorithm is a fundamental cryptographic protocol that allows two parties to securely establish a shared secret key over an insecure communication channel. This report presents the implementation and analysis of the Diffie-Hellman key exchange algorithm, exploring its mathematical foundations, practical implementation, and security properties. The project involves implementing the algorithm, analyzing its security, evaluating its performance, and discussing its strengths and limitations.

Algorithm:

Consider two people, Alice and Bob, who want to utilize the Diffie-Hellman key exchange in their messages. The algorithm for generating the key and sharing it is as follows:

Agree on public parameters:
Select a large prime number, 'p'.
Choose a generator ‘g'

Alice's steps:
Choose a secret random number, **a**.
Calculate **A*** as **A*** $= g^a \pmod p$
Send **A*** to Bob. (**A*** is the public key of A)

Bob's steps:
Choose a secret random number, **b.**
Calculate **B*** as **B*** $= g^b \pmod p$
Send **B*** to Alice. (**B*** is the public key of B)

Shared private key calculation:
Alice receives **B*** from Bob and calculates the shared secret key as $K = (B^*)^a \pmod p$
Bob receives **A*** from Alice and calculates the shared secret key as $K = (A^*)^b \pmod p$

The K obtained by Alice and Bob is the same.
After generating the key that is common to both Alice and Bob, Alice will further use this key to encrypt the message. It will then be passed through the communication medium where attackers may be trying to access the info. Since it's locked by Alice, no one can open it easily. After reaching Bob, it will be decrypted by him, by his own key which is the same as used by Alice to encrypt it. In this way, secure communication can happen through a single key.
More details can be found in the **Asymmetric Key Encryption** section**.**

Implementation:

```
long long int genrandomprime(long long int ll, long long int ul)

{
    long long int range = ul - ll + 1;
    long long int randno = rand() % range + ll;
    while (!isprime(randno))
        randno = rand() % range + ll;
    return randno;
}
```

The above function generates the random primes p and g, that are public parameters, within a certain upper limit and lower limit.

```
long long int secure(long long int g, int x, long long int n)
{
    int i;
    long long int l = 1;
    for (i = 0; i < x; i++)
        l = (l % n) * g;
    return l % n;
}
```

The above function calculates the value of $g^x \bmod n$.

```
int x, y;
long long int A, B, K1, K2, g, n, ll = 1000000000, ul = 9999999999; // lower and upper limits for prime no. generation
srand(time(NULL));
g = genrandomprime(ll, ul);
n = genrandomprime(ll, ul);
printf("(%lld,%lld)\n", g, n);
printf("Enter the numbers private to person 1 : ");
scanf("%d", &x);
printf("Enter the numbers private to person 2 : "); // Only Alicee knows x and only Bob knows y
scanf("%d", &y);
A = secure(g, x, n);
B = secure(g, y, n); // Both A and B are exchanged between Alice and Bob via a public domain .
printf("Public key generated by person 1 : %lld\n", A);
printf("Public key generated by person 2 : %lld\n", B);
K1 = secure(B, x, n);
K2 = secure(A, y, n); // line 21,24 computed privately in Alice's system and line 22,25 computed privately in Bob's system
if (K1 == K2)
    printf("K1 = K2 = %lld\nHence a symmetric key is generated .\n", K1);
else
    printf("The Keys generated are not symmetric .\nK1 = %lld K2 = %lld\n", K1, K2);
return 0;
```

The above code utilizes the previous functions to implement the Diffie-Hellman key exchange algorithm and generate a shared private key.

```c
void encryptFile(const char *filename, long long key) {
    int div_count = 0;
    FILE *inputFile, *outputFile;
    char ch;
    long long key_copy=key;

    inputFile = fopen(filename, "r");
    if (inputFile == NULL) {
        printf("Error opening the file.\n");
        return;
    }
    outputFile = fopen("encrypted.txt", "w");
    if (outputFile == NULL) {
        printf("Error creating the encrypted file.\n");
        fclose(inputFile);
        return;
    }
    while ((ch = fgetc(inputFile)) != EOF) {
        if (!isspace(ch)) {
            ch += (key_copy % 10); // Encrypting the character

            // Wrap around if necessary
            if (ch > 127)
                ch = 32 + (ch - 127);
            else if (ch < 32)
                ch = 127 - (32 - ch);

            key_copy /= 10; // Move to the next digit in the key
            div_count++;
            if(div_count>7)
            {
                key_copy=key;
                div_count=0;
            }
        }
        fputc(ch, outputFile);
        if (isspace(ch)) {
            // Restore the key
            key_copy = key;
        }
    }
    fclose(inputFile);
    fclose(outputFile);
    printf("File encrypted successfully!\n");
}
```

```c
void decryptFile(const char *filename, long long key) {
    FILE *inputFile, *outputFile;
    char ch;
    int div_count=0;
    long long key_copy=key;

    inputFile = fopen(filename, "r");
    if (inputFile == NULL) {
        printf("Error opening the file.\n");
        return;
    }
    outputFile = fopen("decrypted.txt", "w");
    if (outputFile == NULL) {
        printf("Error creating the decrypted file.\n");
        fclose(inputFile);
        return;
    }
    while ((ch = fgetc(inputFile)) != EOF) {
        if (!isspace(ch)) {
            ch -= (key_copy % 10); // Decrypting the character
            // Wrap around if necessary
            if (ch < 32)
                ch = 127 - (32 - ch);
            else if (ch > 127)
                ch = 32 + (ch - 127);

            key_copy /= 10; // Move to the next digit in the key
            div_count++;
            if(div_count>7)
            {
                key_copy=key;
                div_count=0;
            }
        }
        fputc(ch, outputFile);
        if (isspace(ch)) {
            // Restore the key
            key_copy = key;
            // key += ch - ' ';
        }
    }

    fclose(inputFile);
    fclose(outputFile);
    printf("File decrypted successfully!\n");
}
```

Function: encryptFile
Description: Encrypts the content of a file using a provided key, creating an "encrypted.txt" file with the encrypted data.
Function: decryptFile
Description: Decrypts the content of a file using a provided key, creating a "decrypted.txt" file with the decrypted data.

Mathematical Analysis:

Let us perform the same operations as elaborated in the algorithm

$A^* = g^{\boldsymbol{a}} \bmod p$

$B^* = g^{\boldsymbol{b}} \ mod \ p$

Here, we will use the property

$$a(mod\ n)^*\ b(mod\ n) = (a^*b)(mod\ n)$$

$(A^*)^{\boldsymbol{b}} \bmod p = (g^{\boldsymbol{a}} \ mod \ p)^{\boldsymbol{b}} \ mod \ p = g^{\boldsymbol{ab}} \ (mod \ p)$

$(B^*)^{\boldsymbol{a}} \bmod p = (g^{\boldsymbol{b}} \ mod \ p)^{\boldsymbol{a}} \ mod \ p = g^{\boldsymbol{ba}} \ (mod \ p)$

Hence, $A^{\boldsymbol{b}} \bmod p = B^{\boldsymbol{a}} \bmod p$ and we obtain the same shared private key.

Strengths:

- **Easy to execute & manage:** Users have only one key for encryption and decryption so it's easy to execute and manage.
- **Forward secrecy:** Forward secrecy in Diffie-Hellman means that even if someone steals a private key, they can't use it to decrypt previous conversations. Each session generates a new secret key, so compromising one session doesn't affect others. This adds an extra layer of security and keeps past communications confidential.
- **Key Exchange without a Trusted Third Party:** Diffie-Hellman allows for secure key exchange between two parties without the need for a trusted third party. The two parties can independently generate their own private keys and exchange public keys over an insecure channel. This decentralized approach eliminates the need for a central authority or pre-shared keys, making it suitable for scenarios where a trusted third party may not be available or desirable.

Limitations:

In a Diffie-Hellman key exchange, a **middleman attack**, also known as a **Bucket Bridge Attack**, can occur when an unauthorized individual intercepts and alters the communication between two parties by manipulating and gaining access to the private keys of the sender and receiver.
This is how it takes place:

- **Initial key exchange**: Alice and Bob agree on a large prime number (p) and a generator (g), which are publicly known parameters. Alice chooses a secret random number (a) and calculates **A*** g^a (mod p).
  Bob also chooses a secret random number (b) and calculates **B*** g^b (mod p). They exchange **A*** and **B***.
- **Intercepting the exchange**: The attacker, Eve, intercepts the messages containing **A*** and **B*** from Alice and Bob, respectively.
- **Generating fake keys**: Eve generates her own secret random number (e) and calculates E= g^e (mod p). Instead of forwarding Bob's original **B*** to Alice, Eve sends her own generated value, E, to Alice, making Alice believe that she is communicating with Bob.

- **Creating a separate exchange**: Eve also intercepts Alice's message containing **A*** and sends her own generated value, E, to Bob, making Bob believe that he is communicating with Alice.
- **Final key exchange**: Alice and Bob both calculate a shared secret key using the values they received. Alice calculates K1 =E^{a}(mod p), while Bob calculates K2 =E^{b} (mod p).
- **Intercepting and altering messages**: From this point forward, Eve has access to the shared secret keys K1 and K2. She can intercept subsequent messages exchanged between Alice and Bob, decrypt them using the respective secret keys, read their contents, and potentially alter them before forwarding them to the intended recipient.

By performing these steps, Eve effectively becomes an intermediary between Alice and Bob, and can now manipulate the information being shared.
To mitigate middleman attacks in the Diffie-Hellman key exchange, additional measures can be taken, such as digital signatures, certificates, and authentication protocols.

Asymmetric key encryption:

Asymmetric Key Encryption:
The Diffie-Hellman key exchange algorithm serves as a foundation for asymmetric key encryption. In asymmetric encryption, two different keys are used: a public key and a private key. The keys are mathematically related but computationally difficult to derive from each other.

Using the shared private key obtained through the Diffie-Hellman key exchange, Alice can further utilize this key as her private key for encryption. She keeps this private key secret while sharing her corresponding public key with Bob.

Alice's steps:
1. Generate a pair of asymmetric keys: a private key (PR_Alice) and a corresponding public key (PU_Alice).
2. Encrypt the message using her private key: Ciphertext = encryptFile(Message, PR_Alice).
3. Send the encrypted message and her public key (PU_Alice) to Bob.

Bob's steps:
1. Receive the encrypted message and Alice's public key (PU_Alice).
2. Decrypt the message using Alice's public key: DecryptedMessage = decryptFile(Ciphertext, PU_Alice).

Asymmetric key encryption offers several advantages:
1. Confidentiality: Only the intended recipient with the corresponding private key can decrypt the message, ensuring the confidentiality of the information.
2. Key Distribution: Asymmetric encryption eliminates the need for a secure key exchange mechanism since private keys are kept secret, and only public keys are shared openly.
3. Non-repudiation: Since the sender's private key is used to encrypt the message, the recipient can provide the encrypted message as proof of the sender's identity. This prevents the sender from denying their involvement in the communication.

However, asymmetric encryption is computationally more intensive than symmetric encryption, making it less suitable for encrypting large amounts of data. Therefore, a **common approach is to combine the two encryption methods by using asymmetric encryption for securely exchanging a symmetric encryption key**. This method is adopted in Diffie-Hellman Key Exchange algorithm. Once the symmetric key is shared using asymmetric encryption, the actual data can be encrypted and decrypted using faster symmetric encryption algorithms.