MIT World Peace University

Information and Cybersecurity
Second Year B. Tech, Semester 1

Implementation of Diffie - Hellman Key Exchange

Lab Assignment 6

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

March 19, 2023

Contents

1 Aim

Write a program using JAVA or Python or C++ to implement Diffie-Hellman Key Exchange Algorithm

2 Objectives

To learn how to distribute the key.

3 Theory

3.1 What is the Diffie-Hellman Key Exchange Algorithm?

Diffie-Hellman key exchange is a cryptographic protocol that allows two parties to establish a shared secret key over an insecure communication channel without any prior knowledge of each other. The protocol is based on the discrete logarithm problem and modular arithmetic, and it is widely used in secure communication systems such as SSL/TLS, VPNs, and SSH.

3.2 Working of the Algorithm

Here's how the Diffie-Hellman key exchange works, using a simple example:

1. Alice and Bob agree on a large prime number p and a primitive root of p, g. These values are public and can be shared over an insecure channel.
2. Alice chooses a random secret number a and computes $A = g^a$ mod p. She sends A to Bob over the insecure channel.
3. Bob chooses a random secret number b and computes $B = g^b$ mod p. He sends B to Alice over the insecure channel.
4. Alice computes the shared secret key as $K = B^a$ mod p.
5. Bob computes the shared secret key as $K = A^b$ mod p.

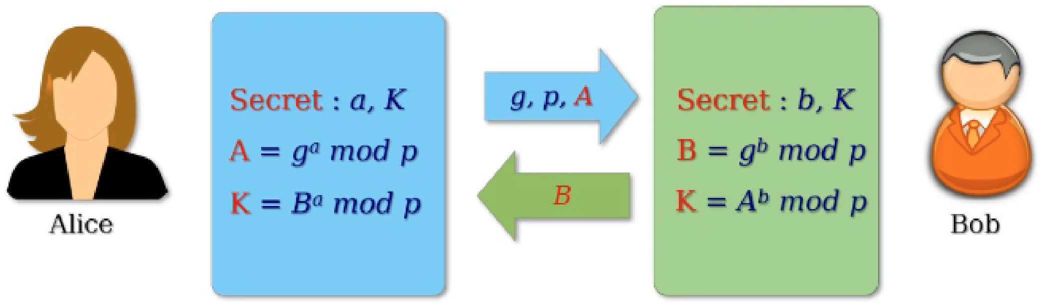

Figure 1: Diffie Hellman Protocol

Both Alice and Bob end up with the same shared secret key K, which can be used for further encryption and decryption of messages.

3.3 Example

Alice and Bob agree on a large prime number $p = 11$ and a primitive root of p, $g = 2$. These values are public and can be shared over an insecure channel.

1. Alice chooses a random secret number $a = 7$ and computes $A = g^a \bmod p = 2^7 \bmod 11 = 7$. She sends $A = 7$ to Bob over the insecure channel.
2. Bob chooses a random secret number $b = 5$ and computes $B = g^b \bmod p = 2^5 \bmod 11 = 10$. He sends $B = 10$ to Alice over the insecure channel.
3. Alice receives $B = 10$ from Bob and computes the shared secret key as $K = B^a \bmod p = 10^7 \bmod 11 = 7$.
4. Bob receives $A = 7$ from Alice and computes the shared secret key as $K = A^b \bmod p = 7^5 \bmod 11 = 10$.

Now Alice and Bob both have the same shared secret key $K = 7 = 10$, which they can use to encrypt and decrypt their messages using a symmetric encryption algorithm such as AES.

4 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Visual Studio Code
Compilers or Interpreters : Python 3.10.1

5 Input and Output

```
p:  239
g:  7
Shared secret key from A 95
Shared secret key from B 95
```

6 Code

```python
# Diffie Hellman algorithm.
from pathlib import Path
import sys, os
ROOT_DIR = str(Path(__file__).parent.parent)
sys.path.insert(1, os.path.join(ROOT_DIR, "Assignment_4"))
from rsa import isPrime, getLowLevelPrime
import random

def diffie_hellman(p, g):
    x_a = random.randint(2, p - 1)
    x_b = random.randint(2, p - 1)
    p_a = user_a(p, g, x_a)
    p_b = user_b(p, g, x_b)
    shared_secret_key = pow(p_a, x_b) % p
    print("Shared secret key from A", shared_secret_key)
    shared_secret_key = pow(p_b, x_a) % p
    print("Shared secret key from B", shared_secret_key)

    return shared_secret_key

def user_a(p, g, x):
    p_b = pow(g, x) % p
    return p_b

def user_b(p, g, x):
    p_b = pow(g, x) % p
    return p_b

def check_primitive_root(p):
    for i in range(2, p):
        temp_list = []
        for j in range(0, p-2):
            temp_list.append(pow(i, j) % p)
        # print(temp_list)
        if len(set(temp_list)) == len(temp_list):
            return i

# print("Enter the value of p: ")
# p = int(input())

p = getLowLevelPrime(8)
print("p: ", p)
# p = 5
g = check_primitive_root(p)
print("g: ", g)

shared_secret_key = diffie_hellman(p, g)
```

Listing 1: "SHA Integrity Check"

7 Conclusion

Thus, we have successfully implemented the Diffie-Hellman Key Exchange Algorithm. We learnt about the working of the algorithm and how it is used to distribute any key between 2 parties.

8 FAQ

1. **What are other key exchange protocols, other than DH algorithm?**

   (a) **RSA Key Exchange**: This protocol uses the RSA algorithm to exchange keys securely between two parties.

   (b) **Elliptic Curve Diffie-Hellman (ECDH)**: This is a variant of the DH algorithm that uses elliptic curve cryptography to provide stronger security and more efficient key exchange.

   (c) **Kerberos**: This is a network authentication protocol that uses a trusted third party (a Key Distribution Center or KDC) to distribute secret keys between two parties.

   (d) **Secure Remote Password (SRP)**: This is a password-based key exchange protocol that allows two parties to establish a shared secret key without revealing their passwords to each other or to an eavesdropper.

   (e) **Signal Protocol**: This is a modern and widely used protocol for secure messaging that uses a combination of the Double Ratchet Algorithm and the DH algorithm to perform key exchange.

   (f) **Station-to-Station (STS)**: This is a protocol that combines elements of the DH and RSA key exchange protocols to provide stronger security and more efficient key exchange.

2. **Explain the different types of keys.**

   In cryptography, keys refer to the secret values used for encryption and decryption of messages.

   (a) **Symmetric Keys**: Also known as shared keys, these are secret keys that are used for both encryption and decryption of messages. The same key is used by both the sender and the receiver. Examples of symmetric key algorithms include AES, DES, and Blowfish.

   (b) **Public Keys**: Also known as asymmetric keys, these are key pairs consisting of a public key and a private key. The public key is widely distributed and is used for encryption, while the private key is kept secret and is used for decryption. Examples of public key algorithms include RSA, Diffie-Hellman, and elliptic curve cryptography.

   (c) **Session Keys**: These are temporary symmetric keys that are generated for a single session of communication between two parties. They are used to encrypt and decrypt messages exchanged during the session and are discarded once the session is over. Session keys are often used to provide forward secrecy, which means that compromising one session's key does not compromise the security of past or future sessions.

   (d) **Key Exchange Keys**: These are public keys used specifically for exchanging symmetric keys between two parties. Key exchange algorithms like Diffie-Hellman and Elliptic Curve Diffie-Hellman are used to establish a shared secret key between two parties without actually transmitting the key over the communication channel.

3. **Explain different key management issues.**

   Key management is the process of securely generating, storing, distributing, and revoking cryptographic keys. Here are some of the most common key management issues in cryptography:

(a) **Key Generation**: The process of generating strong cryptographic keys is essential to ensuring the security of cryptographic systems. However, generating keys that are both random and unpredictable can be difficult. Key generation must be done securely, and the keys must be protected from disclosure or compromise.

(b) **Key Storage**: Cryptographic keys must be securely stored to prevent unauthorized access or disclosure. The storage of keys is often the weakest link in key management. Keys must be stored in a secure environment, and access to the keys must be tightly controlled.

(c) **Key Distribution**: Cryptographic keys must be securely distributed to all parties involved in the communication. Key distribution can be challenging, especially when there are multiple parties involved. Key exchange protocols like Diffie-Hellman and RSA can be used to securely exchange keys between parties.

(d) **Key Revocation**: Keys must be revoked when they are no longer needed or when they have been compromised. Revocation is necessary to prevent unauthorized access to data that was encrypted using the compromised key. Revocation can be challenging, especially in large systems where there are many keys in use.

(e) **Key Expiration**: Cryptographic keys have a limited lifespan, and must be periodically updated or replaced to maintain their security. Key expiration policies must be carefully designed to balance security and usability.