VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Implementing Secure DataTransfer using Diffie-Hellman Algorithm with AES Encryption

J COMPONENT PROJECT

CSE3501 – Information Security Audit and Analysis

by -

Team Members

Sudhanshu Gupta	**19BCE0188**
G.M.Rishitha	**19BCE0269**
Vidhi Moteria	**19BCE0525**
Mohit Chotani	**19BCE0893**
Shantha Lakshme. S	**19BDS0118**

Prof. Raja S P

Table of Contents

ABSTRACT

Communication over web, internet and through Social media are some of the popular ways through which the users communicate with each other via the use of messages.These messages are sent from the source to the destination effortlessly. But then there is no guarantee that the messages or the data shared are not intruded by the intruder in between the source and the destination? Encryption is one of the techniques for providing security for the data that has been sent by the sender. There are different algorithms used in order to achieve the encryption of the data. To address this issue the secure text transfer using Diffie Hellman Key Exchange algorithm is introduced and will be used by us in this project. This application will allow the user to send the plain text to the destination. Prior to sending the message the user needs to select the particular algorithm like AES, DES etc. Then the message will be encrypted through the use of such algorithms. The key will be sent to the destination in order to decrypt the message.

However, since the general Diffie-Hellman is prone to several attacks which makes it less reliable for performing the secret key exchange. So, the idea is to first make the Diffie-Hellman more secure and therefore,we are trying to implement an improved Diffie-Hellman algorithm that will make it easier for exchanging the secret key generated which can then be used in symmetric algorithms like AES to perform the encryption and decryption operations making it even more secure.

The project will be implemented in real time with great ease and users can easily rely on this application after using it. This project will be developed using Python Language.

Keywords: Cryptography, Encryption, Symmetric Key Encryption and Asymmetric Key Encryption, Diffie-Hellman, AES, DES, RSA, Public-Key Cryptosystem

INTRODUCTION

Security is one of the most vital concerns in any field, especially today in the current scenario when there is an extensive rise in the usage of the internet, people tend to require secure transmission and secure ways of having end to end encryption. Thus, an effective text encryption algorithm is needed for achieving an immense amount of privacy.In order to provide privacy, we propose a safe text encryption algorithm by utilizing an upgraded variant of Diffe-Hellman and AES. By making improvements in the Diffie Hellman algorithm a highly secure key can be generated which can then be used in the AES algorithm to perform the encryption and decryption operations on the text.

One of the fundamental issues that Cryptography attempts to address includes communication occurring over an unreliable channel protected from unauthorized access. This issue leads to the requirement for a key trade calculation that can be utilized to exchange keys safely over an insecure channel. However, sharing the key to the other party over a large teleprocessing network can be costly and tedious. This is where Diffie-Hellman came into the picture with the intent of making two parties exchange a session key which will then be utilised in a symmetric algorithm called AES(Advanced Encryption Standard)..The features that can be included in the secure text transfer using Diffie Hellman Key Exchange are as follows:

Secure: As the algorithm is used the encryption and decryption will be secure.

Confidentiality: This application can help in sending the confidential message without any attacks.

Attacks: This application can help in prevention of any types of attacks with great ease.

Ease: This application is easy to use.

PROBLEM STATEMENT

A brute force method for stable transfer entails user 1 to encrypt the data using a key and sharing the key with a second user for decryption. Although slightly secure, this technique entails a high chance of eavesdropping through a third party who could pick up the key.

Which is why we need a better system, wherein users can securely switch encrypted records, without understanding the secret key - a machine that independently generates the equal key at both ends.

OBJECTIVE

Our Objective is to improve and implement the existing Diffie Hellman algorithm, which is a method of securely exchanging cryptographic keys over a public channel, and thus create a Platform for Secure Data Transfer.

MOTIVATION

The main motivation behind this project is to develop a secure data transfer technique that could provide end to end encryption. In spite of being generally utilized, Diffie-Hellman is inclined to different attacks like Man-in-middle, plain-text, DoS and others. Thus, we've proposed an algorithm which is an update to the Diffie-Hellman for producing an exceptionally protected key which can safeguard against the above mentioned attacks and we then, at that point, utilize this key to perform encryption and decoding of the text utilizing AES algorithm technique.

TECHNIQUES IDENTIFIED

In order to solve the above issue we looked into various techniques that could be used to solve the problem, and we came across some of the key exchange algorithms that are mentioned below**:**

DIFFIE HELLMAN :

The Diffie Hellman key exchange method is mostly used to exchange the keys over an insecure channel also.

Diffie hellman is one of the best key exchange protocols which exchange a key which undergoes AES encryption, between sender and receiver.

Diffie hellman alone can be prone to some vulnerabilities like man-in-the middle attack. Hence here diffie hellman algorithm is being used along with the combination of AES encryption technique.

Neither the sender nor the receiver need any prior knowledge of each other is being one of the coolest advantages of this algorithm.

Diffie-Hellman's security rests primarily on its difficulty in computing discrete logarithms.

This method of key exchange is frequently used in security protocols such as TLS, IPsec, SSH, PGP, and others since it is one of the most reliable means of safely distributing keys.

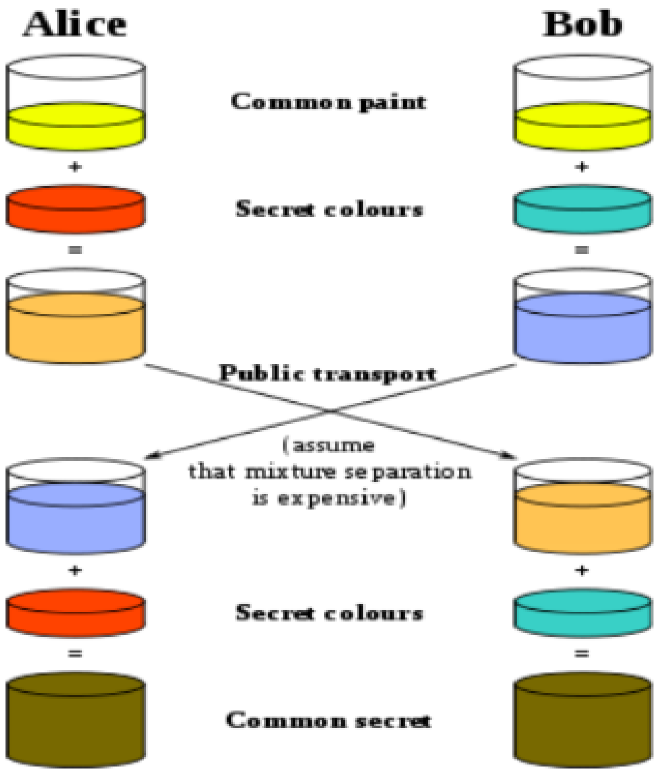


Figure 1: Illustration of idea behind Diffie-Hellman

AES ENCRYPTION:

It is an encryption which requires a secret key to encrypt as well as decrypt the messages.

In this encryption both the sender and receivers use the same secret key generated.

It provides the safety by encrypting using AES algorithm during the exchange using the above technique mentioned.

Unlike its predecessor DES, AES does not rely on a Feistel network, but instead it is based on a substitution-permutation network.

It can be implemented in hardware as well as software.

The AES algorithm is more robust against hacking because it uses longer key sizes like 128, 192, and 256 bits. As a result, it is a very safe and secure protocol

PERFORMANCE ANALYSIS

PROPOSED MODEL:

As we've seen above the Diffie-Hellman algorithm is at risk of some certain attacks and in the case of AES producing a tougher secret key isn't a problem however the way to change the important thing generated is the problem though it's not a major problem it's worth making it simple. So, the concept is to first make the Diffie-Hellman more secure and to be able to try this, we'll take a primitive root to the secret obtained received and take non-public keys and perform the set of rules using the secret key obtained to generate a 2d mystery key thereby making it greater stable as now the attacker can't carry out whatever on this because he/she won't know that we're the usage of a primitive root to the secret key acquired. Now, this secret key can be exchanged and used in the AES algorithm to carry out the encryption and decryption operations.

The new proposed algorithm has the following steps:

The sender and receiver picks a prime number '**p**' and it's primitive root '**g**'.

Then both of them choose a private key each, let it '**a**' and '**b**', this private key is only known to them.

The public key of both the sender and receiver is calculated, for sender it is equal to

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

And for receiver it's

$$B=g^{b} \bmod p$$

Where A and B represent the respective public key.

The public keys generated are then exchanged by both sender and receiver.

Now both the sender and receiver calculate their secret key i.e sender's secret key is obtained by

$$S= B^{a} \bmod p$$

And similarly for receiver

$$S= A^{b} \bmod p$$

Where **S** represents the secret key generated.

We now consider an integer '**e**' which is the primitive root of the secret key '**S**'.

Both the sender and receiver now choose their second set of private keys and let those be '**c**' and '**d**' respectively.

The second public key of both the sender and receiver is calculated, for sender it is equal to

$$AA = g^{c} \ mod \ S.$$

and for receiver it's

$$BB = g^{d} \ mod \ S$$

Here **AA** and **BB** represent the sender and receiver's second set of public keys.

The public generated are again exchanged by both the sender and receiver.

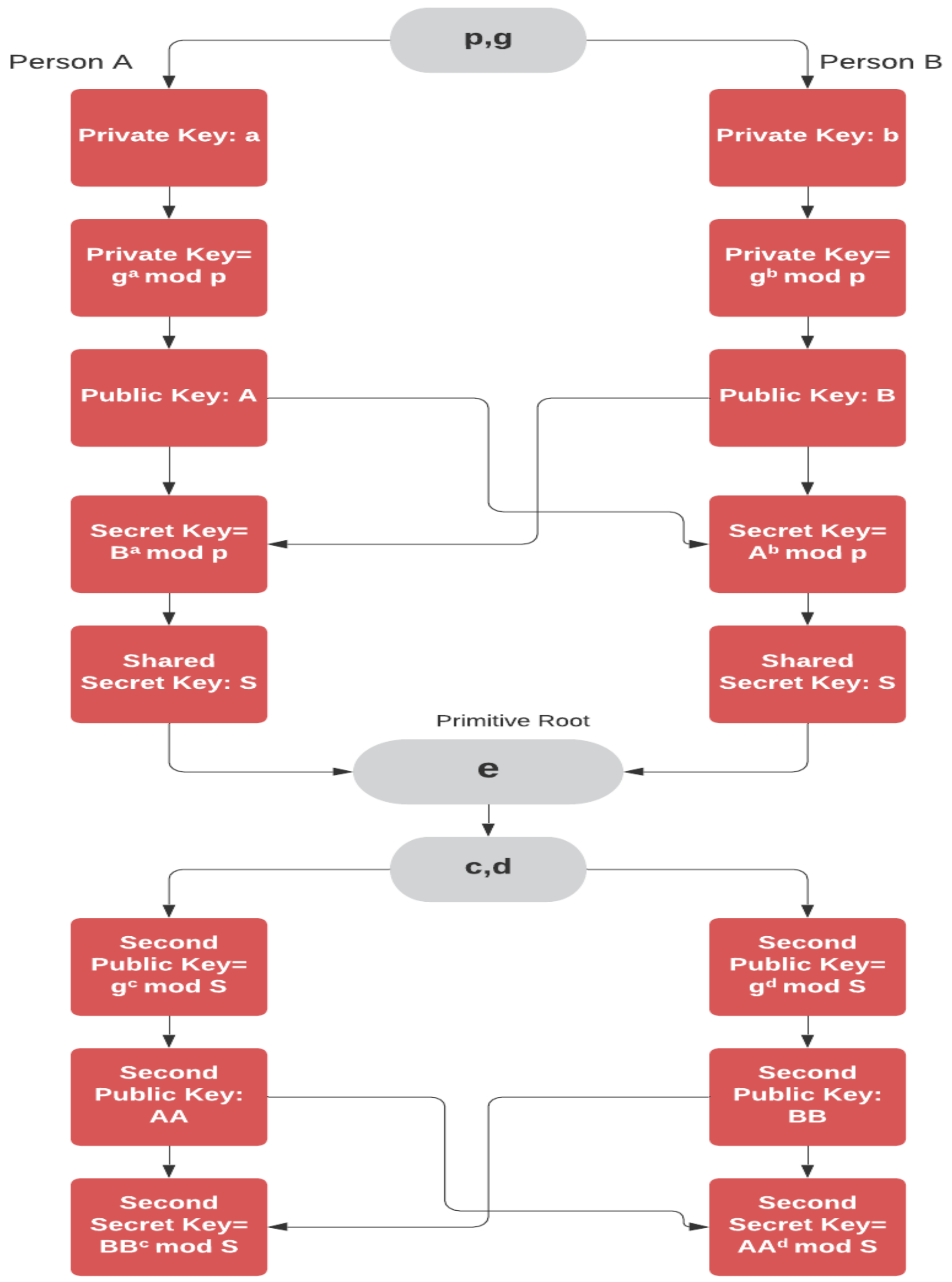
Now both the sender and receiver calculate their final secret key, for sender it's

$$S = BB^{c} \ mod \ S$$

and for receiver it's

$$S = AA^{b} \ mod \ S$$

FLOWCHART:


p,g
Person A
Person B
Private Key: a
Private Key: b
Private Key= g^a mod p
Private Key= g^b mod p
Public Key: A
Public Key: B
Secret Key= B^a mod p
Secret Key= A^b mod p
Shared Secret Key: S
Shared Secret Key: S
Primitive Root
e
c,d
Second Public Key= g^c mod S
Second Public Key= g^d mod S
Second Public Key: AA
Second Public Key: BB
Second Secret Key= BB^c mod S
Second Secret Key= AA^d mod S

LITERATURE SURVEY :

[1] "Secure authentication approach using Diffie-Hellman key exchange algorithm for WSN."

Authors : Kumar, Sanjay, and R. K. Singh.

International Journal of Communication Networks and Distributed Systems 17.2 (2016): 189-201.

Description:

Most of the sensor network applications are deployed in a hostile environment. So, to keep the confidentiality of the information over the sensor network, secure communication between the sink and the sensor nodes is required.

In this paper , the author proposed a method to establish secure communication over a sensor network using the idea of public key cryptography. They've used the Diffie-Hellman key distribution algorithm to establish a secure key between the nodes by using the secret information that was pre-loaded in every node's memory before deployment. And, for Encryption they have used the technique DEA (Data Encryption Algorithm). Additionally, they have introduced a tampering module to provide high resiliency against key compromisation and node capture attack.

[2] "Strong Diffie-Hellman-DSA Key Exchange".

Authors: Ik Rae Jeong, Jeong Ok Kwon, Dong Hoon Lee.

IEEE Communications Letters (Volume: 11, Issue: 5)

Description:

This paper suggests a strong Diffie-Hellman-DSA combined method to provide security against session state reveal attacks. In all of the previous Diffie-Hellman-DSA methods, they do not provide security against session state reveal attacks.

[DSA =(DSA.key, DSA.gen, DSA.ver)]

In general, the DSA technique deals with creating private-public key pair signatures and verifying the signature message pair using public key and returns the result.

In this paper, they discussed about the importance of three security notations of key exchange:

1. Key Independence
2. Forward Secrecy
3. Security (against session state reveal).

The paper revolves around two secure keys, one for the communication of user1 to user2 and the other for user2 to user1. Thus,the improved Diffie Hellman -DSA method/scheme approached by the paper provides all the security notions discussed for secure data transfer against session state review attacks.

[3] "A secured multiplicative Diffie Hellman key exchange routing approach for mobile ad hoc network".

Authors : Manjula, T., & Anand, B. (2019).

Journal of Ambient Intelligence and Humanized Computing, 1-11.

Description:

In order to secure the nodes from suspicious activities, an efficient routing algorithm is needed to provide a safe and secure network. Secure routing based multiplicative Diffie Hellman key exchange (MDKE) algorithm is suggested in order to enhance MANET's security in this paper.

Majority of the routing protocols concerning MANETs are built on the idea that for the data packets to be forwarded to the nodes that are assigned, cooperation amongst the nodes in a network should exist, a malicious node can take an interest so as to join the information traffic course, with the goal that information parcels can be dropped at the season of the transmission of information. In order to solve this issue, cryptography is employed for the purpose of authenticating all of the routing control packets, such that we can prevent outside attackers' participation in the process of route discovery.

[4] “Symmetric Key Generation and Distribution Using Diffie-Hellman Algorithm”.

Authors : Purohit, K., Kumar, A., Upadhyay, M. and Kumar, K., 2020.

In Soft Computing: Theories and Applications (pp. 135-141). Springer, Singapore.

Description:

Diffie–Hellman was one of the first recorded public key cryptography algorithms that was used to transfer data from one user to the other by keeping the integrity and security of the information. The existing Diffie–Hellman algorithm can be accessed easily by means of many attacks like middle man, and brute force etc. The proposed technique can replace it as it uses many complicated functions to generate the common key for all the users in the network, thus making it much better than the previous and creating a secure and safe channel.

In the proposed algorithm, each and every user has to select a random private key ’k’ on his own such that its public key is calculated by Public Key=g(k)modp where, ‘p’ is a large prime number which we select and generator ’g’ of the prime number, where ’g’ is less than ’p’ and the power of ’g’ must be varying from ’1’ to ‘p-1’. The study for the comparison between the existing Diffie–Hellman key sharing protocol and the proposed Diffie–Hellman is done basically on three parameters: the encryption time, the decryption and the total time.

[5] “Fuzzy Elliptic Curve Cryptography based Cipher Text Policy Attribute based Encryption for Cloud Security”.

Authors : Gurpreet Singh, Dr. Sushil Garg.

2020 International Conference on Intelligent Engineering and Management (ICIEM). 2020

Description:

This paper proposes a Ciphertext Policy Attribute Based Encryption which is a form of Public Key Encryption which is eminent as a Data access control scheme for data security.

Here this paper shows more easy and scalable mechanisms which helps to more secure and safe key transfer. Initially, Fuzzy Attribute based Encryption policy has been proposed in which performance can be improved by the ABE scheme by using available attributes for decryption and increasing the accuracy. Next, after encryption using an agreed access policy, the cipher text will be sent through a secure channel using the Diffie Hellman Key Exchange algorithm. Finally, using Elliptic Curve Cryptography, computational complexity can be lowered without sacrificing the security. Also, ECC algorithm is more complex to interpret and hence increases the security of the system.

[6] "A hybrid cryptosystem of image and text files using blowfish and Diffie-Hellman techniques"

Authors : T. K. Hazra, A. Mahato, A. Mandal and A. K. Chakraborty

(2017) 8th Annual Industrial Automation and Electromechanical Engineering Conference (IEMECON), Bangkok, 2017, pp. 137-141

Description:

The technique used in this paper is that initially the user encrypts a file using a secret key generated by the blowfish algorithm. Then using Diffie-Hellman protocol a shared private key will be generated for two users who are trying to communicate over an insecure channel. Now if the second user wants to decrypt the file encrypted by the first user, he has to use the shared key for permission. Once the permission is granted, the file can be decrypted using the blowfish algorithm.

When the file is being sent to another user, if it gets attacked the attacker still would not be able to see the content as the text/image will remain encrypted because the attacker doesn't have the secret key of blowfish encryption. This is how the attacks are intercepted.

[7] "Integrating Diffie-Hellman key exchange into the digital signature algorithm (DSA)"

Authors: L. Harn, M. Mehta and Wen-Jung Hsin.

IEEE Communications Letters, vol. 8, no. 3, pp. 198-200.

Description:

Most of the data transfer in the internet is done using private key concepts to secure the process. It is also very important to protect the key from intruders/ hackers/ middle-men who will try to eavesdrop/steal the key while data transmission.

This paper proposes the method of Deffie Hellman Algorithm in combination with Digital Signature Algorithm (DSA) to prevent the data transfer process from various key attacks. They have also come up with 3 alternative protocols (integrating Deffie-hellman with DSA) to deal with multiple attacks.

1. One round protocol :- for non-interactive applications where only one person will send the data to the other using one secure key.
2. Two round protocol :- for interactive applications where both the parties will share information, using two secret keys.
3. Three round protocol :- this is almost the same as a two round protocol with an additional key confirmation process.

[8] "A hybrid network security algorithm based on Diffie Hellman and Text-toImage Encryption algorithm".

Authors : Abusukhon, A., Anwar, M.N., Mohammad, Z. and Alghannam, B.

(2019) Journal of Discrete Mathematical Sciences and Cryptography, 22(1), pp.65-81.

Description:

There are some techniques like Text-to-Image Encryption algorithm –TTIE with an encryption key and also many more techniques to secure the sensitive data which is shared through the internet. These days safeguarding the data with these kinds of techniques is also becoming a big challenge.

Hence this paper came up with the added security level to the algorithms like Text-to-Image Encryption algorithm - TTIE by exchanging the encryption keys with Diffie Hellman technique.

[9] “privateDH: An Enhanced Diffie-HellmanKey-Exchange Protocol using RSA and AES AlgorithmHyperelliptic Curve Diffie–Hellman-Based Two-Server Password-Only Authenticated Key Exchange Protocol for Edge Computing Systems”

Authors: Ripon Patgiri .

Description:

With the advent of cryptography based information exchange on the internet ,the challenges to protect the data from intruders is also increasing. This paper suggests using RSA cryptography and AES encryption to protect and compensate for the failures of the Diffie-Hellman Key exchange process.

The term privateDH stands for Private Diffie-Hellman algorithm, where the sender receives the public key of the receiver through a trusted third party and encrypts the key using AES & random number generator and sends both (the random key and the encrypted key) back to the receiver. Now, the receiver decrypts the key using its private key to start Diffie-Hellman exchange by sending the two generated prime numbers to the sender after encrypting them by the key received from the sender. Thus, both of them (sender and receiver) uses a single key for communication and uses AES for the encryption.

[10] “A study on diffie-hellman key exchange protocols”

Authors : Manoj Ranjan Mishra, Jayaprakash Kar.

International Journal of Pure and Applied Mathematics, Volume 114 No. 2 2017, 179-189.

Description:

This paper provides a complete study on the purpose and variants of Diffie-Hellman protocol to protect the transmitted data over the network against unauthorized disclosure or modification of the data sent and received between the communicating parties. They have also mentioned its insecurity against man in the middle attack.

An efficient authenticated key exchange protocol has been proposed known as station-to-station (STS) protocol. In two-party authenticated key exchange, the legitimate parties can compute a secret key using Diffie-Hellman and then authenticate each other by exchanging their digital signatures. By integrating this STS protocol with the Diffie-Hellman protocol, we can be able to protect the key transfer against man in the middle attack. This method provides forward secrecy.

One pass protocol and two pass protocols are also discussed in this paper along with several attacks. The purpose of studying these attacks when such protocols are used is to test the security factors of the protocol and landing on to the possible solutions against such vulnerabilities.

IMPLEMENTATION:

How a General Diffie Hellman Key Exchange Works

Step 1: Alice and Bob get public numbers: P=23, G=9.

Step 2: Alice selects a private key a = 4 & Bob selects a private key b=3.

Step 3: Alice and Bob compute public values :

Alice: x = (9^4 mod 23) = (6561 mod 23) = 6

Bob: y = (9^3 mod 23) = (729 mod 23) = 16

Step 4: Alice and Bob exchange public numbers (x and y).

Step 5: Alice receives public key y = 16 and Bob receives public key x = 6.

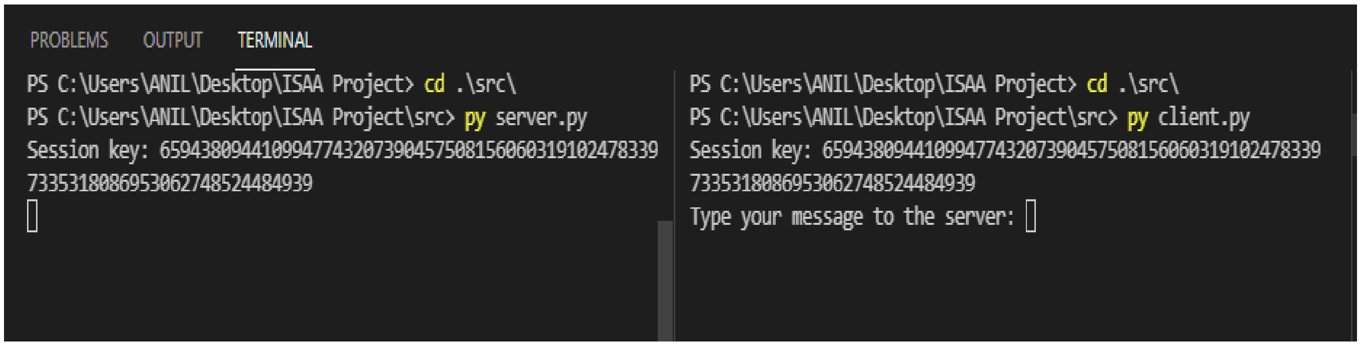
Step 6: Alice and Bob compute Symmetric keys

Alice: Ka = y^a mod p = 65536 mod 23 = 9

Bob: Kb = x^b mod p = 216 mod 23 = 9

Step 7: 9 is the shared key/secret exchanged between Alice and Bob.

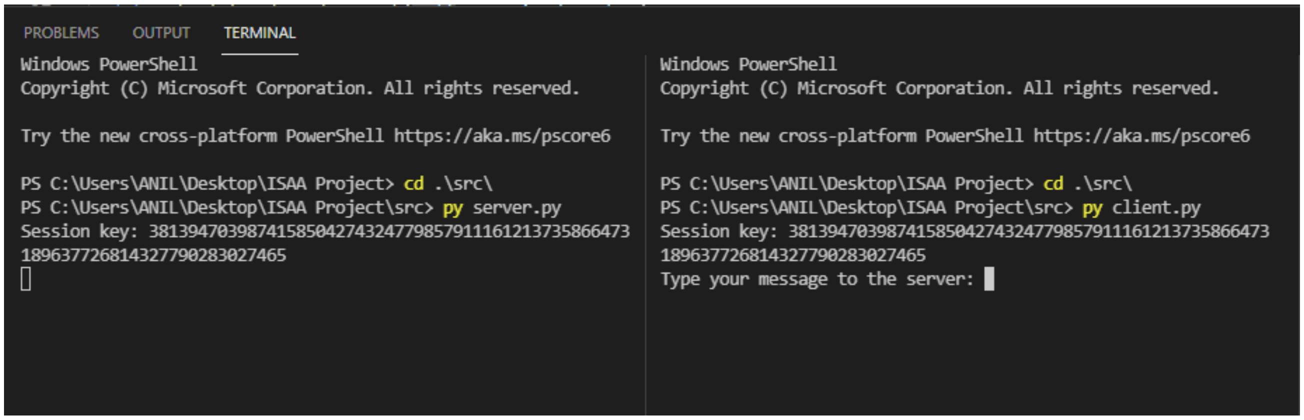
Session key generated and exchanged on client and server side


PROBLEMS OUTPUT TERMINAL

```
PS C:\Users\ANIL\Desktop\ISAA Project> cd .\src\
PS C:\Users\ANIL\Desktop\ISAA Project\src> py server.py
Session key: 6594380944109947743207390457508156060319102478339
7335318086953062748524484939
```

```
PS C:\Users\ANIL\Desktop\ISAA Project> cd .\src\
PS C:\Users\ANIL\Desktop\ISAA Project\src> py client.py
Session key: 6594380944109947743207390457508156060319102478339
7335318086953062748524484939
Type your message to the server:
```

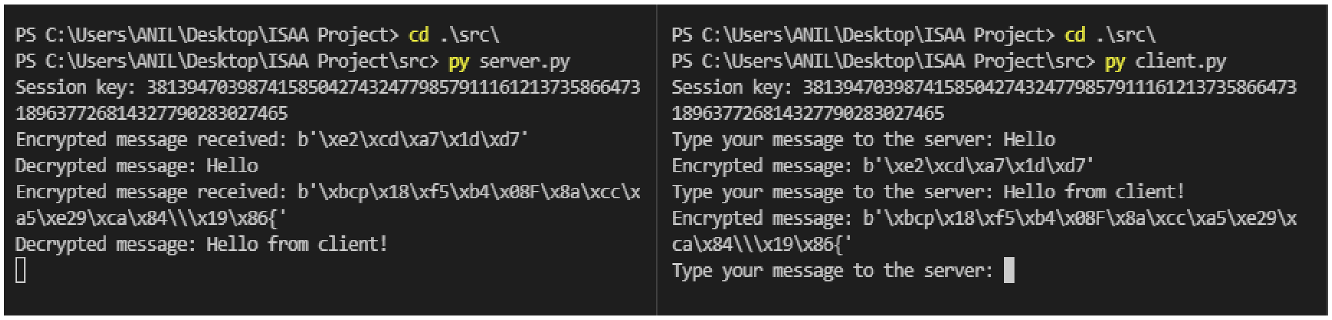
New session key is generated and exchanged every time a session is started


PROBLEMS OUTPUT TERMINAL

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\ANIL\Desktop\ISAA Project> cd .\src\
PS C:\Users\ANIL\Desktop\ISAA Project\src> py server.py
Session key: 3813947039874158504274324779857911161213735866473
189637726814327790283027465
```

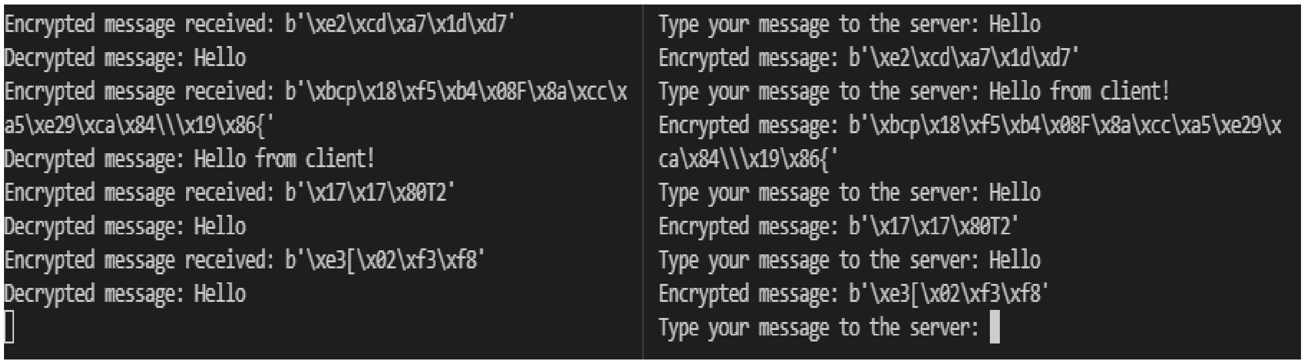
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\ANIL\Desktop\ISAA Project> cd .\src\
PS C:\Users\ANIL\Desktop\ISAA Project\src> py client.py
Session key: 3813947039874158504274324779857911161213735866473
189637726814327790283027465
Type your message to the server:
```

AES Encryption for sending encrypted message


```
PS C:\Users\ANIL\Desktop\ISAA Project> cd .\src\
PS C:\Users\ANIL\Desktop\ISAA Project\src> py server.py
Session key: 3813947039874158504274324779857911161213735866473
189637726814327790283027465
Encrypted message received: b'\xe2\xcd\xa7\x1d\xd7'
Decrypted message: Hello
Encrypted message received: b'\xbcp\x18\xf5\xb4\x08F\x8a\xcc\x
a5\xe29\xca\x84\\\x19\x86{'
Decrypted message: Hello from client!
```

```
PS C:\Users\ANIL\Desktop\ISAA Project> cd .\src\
PS C:\Users\ANIL\Desktop\ISAA Project\src> py client.py
Session key: 3813947039874158504274324779857911161213735866473
189637726814327790283027465
Type your message to the server: Hello
Encrypted message: b'\xe2\xcd\xa7\x1d\xd7'
Type your message to the server: Hello from client!
Encrypted message: b'\xbcp\x18\xf5\xb4\x08F\x8a\xcc\xa5\xe29\x
ca\x84\\\x19\x86{'
Type your message to the server:
```

Same message encrypted value differs every time



```
Encrypted message received: b'\xe2\xcd\xa7\x1d\xd7'
Decrypted message: Hello
Encrypted message received: b'\xbcp\x18\xf5\xb4\x08F\x8a\xcc\x
a5\xe29\xca\x84\\\x19\x86{'
Decrypted message: Hello from client!
Encrypted message received: b'\x17\x17\x80T2'
Decrypted message: Hello
Encrypted message received: b'\xe3[\x02\xf3\xf8'
Decrypted message: Hello
```

```
Type your message to the server: Hello
Encrypted message: b'\xe2\xcd\xa7\x1d\xd7'
Type your message to the server: Hello from client!
Encrypted message: b'\xbcp\x18\xf5\xb4\x08F\x8a\xcc\xa5\xe29\x
ca\x84\\\x19\x86{'
Type your message to the server: Hello
Encrypted message: b'\x17\x17\x80T2'
Type your message to the server: Hello
Encrypted message: b'\xe3[\x02\xf3\xf8'
Type your message to the server:
```

<u>CODE:</u>

DiffieHellman.py

```python
#!/usr/bin/env python3


import sys

import secrets

from Math import Math


class DiffieHellman:

  """

  Encapsulate methods necessary to calculate the shared secret

  between two peers.

  """

  def __init__(self, generator : int, modulus : int): # agreed numbers

    self.__generator = generator
    self.__modulus = modulus
    self.__result, self.__exponent = self.__calculate_local_number()

  def __calculate_local_number(self):
    """
    Returns the random chosen exponent and to number to send
    to the other peer.
    """
     exponent = 2 + secrets.randbelow(sys.maxsize) # range [2, sys.maxsize +
1]
           result = Math.fast_exponentiation(self.__generator, exponent,
self.__modulus)

    return result, exponent

  def calculate_shared_secret(self, received_number):
    """
    Calculates the shared secret having the received number and the exponent
    used for it's own operation
    """
       result = Math.fast_exponentiation(received_number, self.__exponent,
self.__modulus)
    return result
  def get_result(self):

    """ get method for result """

    return self.__result


  def get_exponent(self):

    """ get method for exponent """

    return self.__exponent
```

Server.py

```
import socket

from DiffieHellman import DiffieHellman

import pyaes


class Server:

  def __init__(self, host, port):

    self.__tcp_server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    orig = (host, port)

    self.__tcp_server.bind(orig)

    self.__tcp_server.listen(1)


    self.__session_estabilished = False


  def listen(self):
```

```
  while True:
   con, client = self.__tcp_server.accept()
   if not self.__session_estabilished:
    self.__establish_session(con)
   while True:
    msg = con.recv(1024)
    if not msg:
     break
    print("Encrypted message received: " + str(msg))
    message = self.__decrypt_message(msg)
    print("Decrypted message: " + message.decode('utf-8'))
   self.__session_estabilished = False

 def __receive_generator_and_prime(self, con):
  generator = int(con.recv(1024))
  prime = int(con.recv(1024))
  return generator, prime

 def __establish_session(self, con):
  generator, prime = self.__receive_generator_and_prime(con)

  diffie_hellman = DiffieHellman(generator, prime)
  result = diffie_hellman.get_result()
```

```
    result_from_client = int(con.recv(1024))

    con.send(bytes(str(result), 'utf8'))


    self.__key = diffie_hellman.calculate_shared_secret(result_from_client)

    self.__session_estabilished = True

    print("Session key: " + str(self.__key))


    self.__key = self.__key.to_bytes(32, byteorder = "big")


    self.__aes = pyaes.AESModeOfOperationCTR(self.__key)


  def __decrypt_message(self, message):

    message = self.__aes.decrypt(message)

    return message


if __name__ == "__main__":

  server = Server("", 5000)

  server.listen()
```

Client.py

```
import socket

from DiffieHellman import DiffieHellman
```

```python
import pyaes


class Server:
  def __init__(self, host, port):
    self.__tcp_server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    orig = (host, port)
    self.__tcp_server.bind(orig)
    self.__tcp_server.listen(1)

    self.__session_estabilished = False

  def listen(self):
    while True:
      con, client = self.__tcp_server.accept()
      if not self.__session_estabilished:
        self.__establish_session(con)
      while True:
        msg = con.recv(1024)
        if not msg:
          break
        print("Encrypted message received: " + str(msg))
        message = self.__decrypt_message(msg)
        print("Decrypted message: " + message.decode('utf-8'))
      self.__session_estabilished = False

  def __receive_generator_and_prime(self, con):
    generator = int(con.recv(1024))
    prime = int(con.recv(1024))
    return generator, prime

  def __establish_session(self, con):
    generator, prime = self.__receive_generator_and_prime(con)

    diffie_hellman = DiffieHellman(generator, prime)
    result = diffie_hellman.get_result()
    result_from_client = int(con.recv(1024))
    con.send(bytes(str(result), 'utf8'))

    self.__key = diffie_hellman.calculate_shared_secret(result_from_client)
    self.__session_estabilished = True
    print("Session key: " + str(self.__key))

    self.__key = self.__key.to_bytes(32, byteorder = "big")

    self.__aes = pyaes.AESModeOfOperationCTR(self.__key)
  def __decrypt_message(self, message):

    message = self.__aes.decrypt(message)

    return message


if __name__ == "__main__":

  server = Server("", 5000)

  server.listen()
```

Math.py

```
#!/usr/bin/env python3


import secrets

import queue

import threading

import os


class Math:

  """

  Math is a module with useful math operations,

  it hold methods for large probably primes generation,

  primality test and fast exponentiation.

  """

  @staticmethod
  def __generate_number(bits_size : int):

    """

    Private method to generate a number with `bits_size` bits.

    """

    #number = random.getrandbits(bits_size)

    number = secrets.randbits(bits_size)

     number |= (1 << bits_size - 1) | 1      # ensures that number is odd and
most significant byte is one.

    return number


  @staticmethod

  def fast_exponentiation(base : int, exponent : int, prime : int):

    """

    Calculate the exponantiation of integers

    (base ^ exponent) % prime in logarithmic time: O(lg(exponent)).


    Returns an integer, the result of the operation.

    """

    if base == 0 and exponent == 0:

      return None


    base %= prime # ensure that base is in the range of the answer.

      result = 1      # starts with one because it's the neutral element of
multiplication.
```

```
    while exponent > 0:
      if exponent & 1:
        result = (result * base) % prime

      # applies (base ^= 2) and divides exponent by 2.
      exponent //= 2
      base = (base * base) % prime

    return result # (base ^ exponent) % prime

  @staticmethod
  def miller_rabin_test(number : int, tests : int = 128):
    """
    Tests if `number` is a prime number.

    Returns a boolean value
    """
    if number == 3 or number == 2:
      return True      # random.randrange(2, number - 1) throws and exception when number is either 2 or 3.

    if number <= 1 or number % 2 == 0:
      if(number != 2):
          return False    # immediately discarts if number is even different from 2 or less than one.


    s = 0

    r = number - 1


    # Miller-Rabin test

    while r & 1 == 0:  # finds s and r

      s += 1

      r //= 2


    # Miller-Rabin test

    for _ in range(tests):

      a = max(2, secrets.randbelow(number - 1))

      x = Math.fast_exponentiation(a, r, number)


      if x != 1 and x != number - 1:

        for i in range(1, s):

          if x == number - 1:

            break


          x = Math.fast_exponentiation(x, 2, number)

          if x == 1:

            return False
        if x != number - 1:
          return False

    return True

  @staticmethod
  def generate_prime_number(bits_size : int = 256):
    """
     Generates a, probably, prime number with length `bits_size` in bits. By default bits_size = 2014.

    Returns, probably, a prime.
    """
    prime = Math.__generate_number(bits_size)
    while not Math.miller_rabin_test(prime):
      prime += 2

    return prime # returns a probably prime

  @staticmethod
  def generate_safe_prime_number(bits_size : int = 256):
    """
    Generates a safe prime number. A prime number p is safe is (p - 1) / 2 is
a prime.

    """

    prime = Math.generate_prime_number(bits_size)       # gets prime number

     while not Math.miller_rabin_test((prime - 1) // 2): # tests if (p - 1) /
2 is prime.

       prime = Math.generate_prime_number(bits_size)      # if not, generates
another prime number and retry.



    return prime



  @staticmethod

  def __generate_generator_and_prime(id : int, bits_size : int , result_queue
: queue):

    """

     Generates a safe prime number p and the generator of Zp, Zp = (Z, * mod
p)

    Returns both numbers

    """

     safe_prime = Math.generate_safe_prime_number(bits_size)          # Gets a
safe prime number p

     factor = (safe_prime - 1) // 2                                          #
derivate a prime from safe prime.



     generator = max(2, secrets.randbelow(safe_prime - 1)) # Choose a random
integer in the range [2, safe_prime - 2]

    exponentation = Math.fast_exponentiation(generator, factor, safe_prime)
    while exponentation == 1: # Some guy in stack overflow taugth this test
to me.
      generator = max(2, secrets.randbelow(safe_prime - 1))
      exponentation = Math.fast_exponentiation(generator, factor, safe_prime)

    result_queue.put((generator, safe_prime)) # save in the queue the result
of the operation.

  @staticmethod
  def generate_generator_and_prime(bits_size : int = 256):
    """
    Generates a safe prime number p and the generator of Zp, Zp = (Z, * mod
p)
    Returns both numbers
    """
    q = queue.Queue()
    # threads to search for prime and generator.
    threads = [threading.Thread(target=Math.__generate_generator_and_prime,
              args=(i, bits_size, q)) for i in range(os.cpu_count() // 2)]

    for thread in threads:
      thread.daemon = True
      thread.start()
    return q.get() # returns the first result of the threads
```

FUTURE SCOPE:

As the security is growing day by day attackers are also being more cognizant. Each security schema has some weak points i.e. if an attacker knows them then he can bypass security. So to make the system more secure we can work on the weakness of algorithm and can further enhance the security. Moreover since Diffie Hellman is prone to man-in-the-middle attack we can further work over enhancing and improving a more secure Diffie hellman algorithm, so as to prevent attacks like man-in-the middle attack.

CONCLUSION:

Diffie–Hellman key exchange (DH) is a method of securely exchanging cryptographic keys over a public channel and is one of the first public-key protocols. DH is one of the earliest practical examples of public key exchange implemented within the field of cryptography.

In the public key cryptosystem, enciphering and deciphering are governed by distinct keys, E and D, such that computing D from E is computationally infeasible (e.g., requiring more than 10^100 instructions). The enciphering key E can thus be publicly disclosed without compromising the deciphering key D. This was the main ideology behind the Diffie-Hellman Key Exchange Protocol. Each user of the network can, therefore, place his enciphering key in a public directory. This enables any user of the system to send a message to any other user enciphered in such a way that only the intended receiver can decipher it. As such, a public key cryptosystem is a multiple access cipher.

A private conversation can therefore be held between any two individuals regardless of whether they have ever communicated before. Each one sends messages to the other enciphered in the receiver’s public enciphering key and deciphers the messages he receives using his own secret deciphering key.

In general, we've found that the Diffie-Hellman algorithm makes it easier for exchanging the secret key generated which can then be used in symmetric algorithms like AES to perform the encryption and decryption operations.

But the general Deffie-Hellman is prone to several attacks making it unreliable for performing the secret key exchange. Our proposed methodology will be impenetrable for the above-mentioned attacks. We will use the key generated in symmetric algorithms like AES and make it even more secure. Our proposed scheme eliminates the overheads of key computation and their management. Provision of security to the user's data on the cloud empowers the Data owner to outsource the data to save.

REFERENCES:

[1] https://www.inderscienceonline.com/doi/abs/10.1504/IJCNDS.2016.079102

[2] http://ieeexplore.ieee.org.egateway.vit.ac.in/document/4202097

[3] https://link.springer.com/article/10.1007/s12652-019-01612-8

[4] https://link.springer.com/chapter/10.1007/978-981-15-4032-5_14

[5] https://ieeexplore.ieee.org/abstract/document/9159961

[6] https://ieeexplore.ieee.org/abstract/document/8079577

[7] http://ieeexplore.ieee.org.egateway.vit.ac.in/document/1278320

[8] https://www.tandfonline.com/doi/abs/10.1080/09720529.2019.1569821

[9]
https://www.researchgate.net/publication/351736537_privateDH_An_Enhanced_Diffie-Hellman_Key-Exchange_Protocol_using_RSA_and_AES_Algorithm

[10]
https://www.researchgate.net/publication/317339928_A_study_on_diffiehellman_key_exchange_protocol