Implementing Secure Data Transfer using modified Diffie-Hellman Algorithm and AES Encryption

Abstract

Encryption is one of the techniques for providing security for the data that has been sent by the sender. There are different algorithms used to achieve the encryption of the data. In symmetric key encryption algorithms, safe key exchange is a difficult problem to solve. We created software that first uses machine learning to assess network data and predict whether a medium is safe or not. We also modified the Diffie Hellman Key Exchange algorithm. This will allow the user to send the plain text to the destination using the AES algorithm. The key will be sent to the destination to decrypt the message.

Aim

We aim to create an ML model that predicts if a medium is safe and then 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.

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, coupled with AES, thus creating a Platform for Secure Data Transfer.

Scope

As security is growing day by day attackers are also being more cognizant. Each security schema has some weak points i.e. if the attackerknew them then he could bypass security. So to make the system more secure we can work on the weakness of the algorithm and can further enhance the security.

Introduction

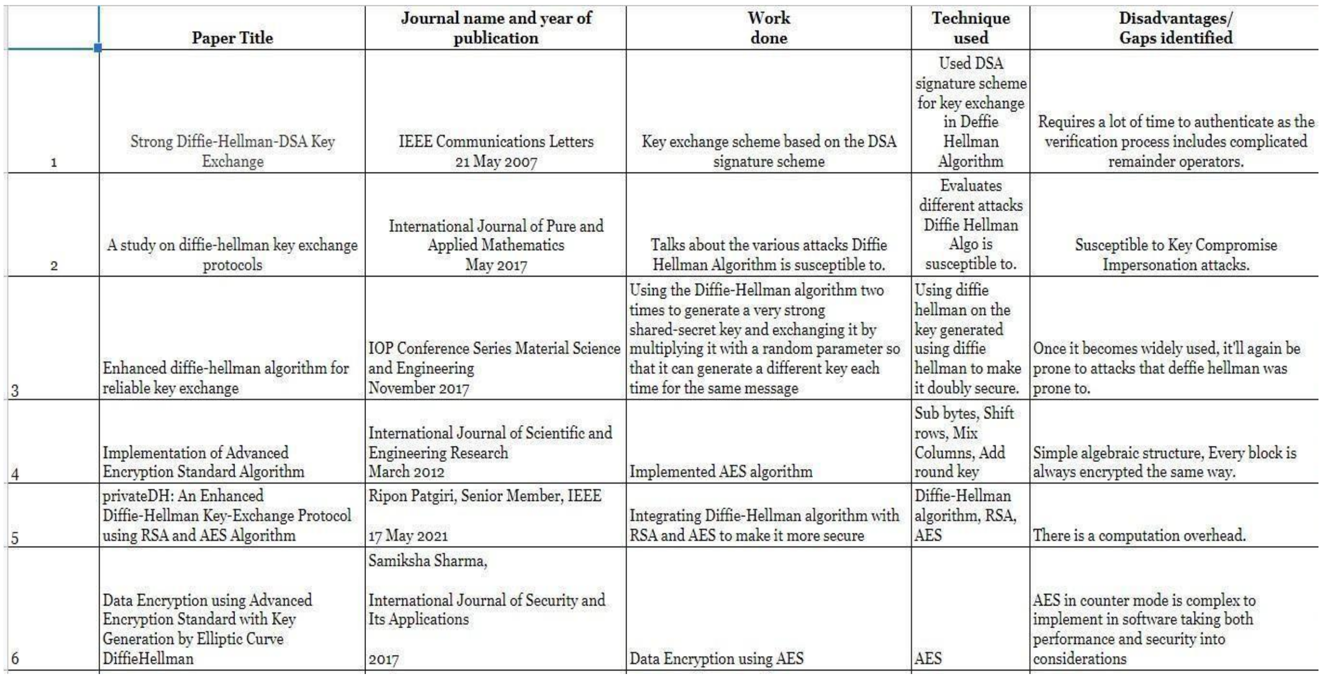
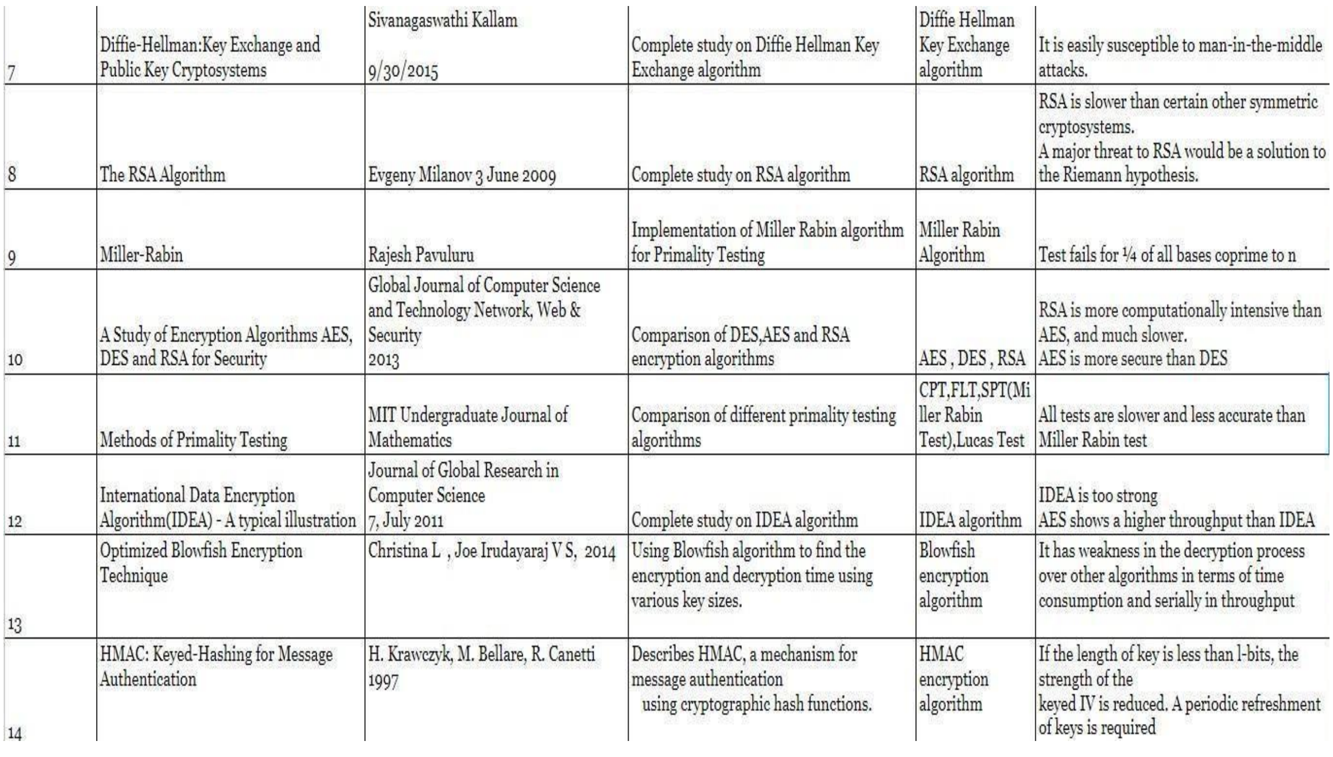
Security is one of the most vital concerns in any field, especially today in the current scenario when there is an extensive rise in the usage of the internet, people tend to require secure transmission and secure ways of having end-to-end encryption. Thus, effective software that securely transfers text from one device to another is needed for achieving an immense amount of privacy. To provide privacy, we propose a two-step method where we first run a machine learning model to check whether the medium is safe and then we use a safe text encryption algorithm by utilizing an upgraded variant of Diffie-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 include 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 utilized in an asymmetric 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:

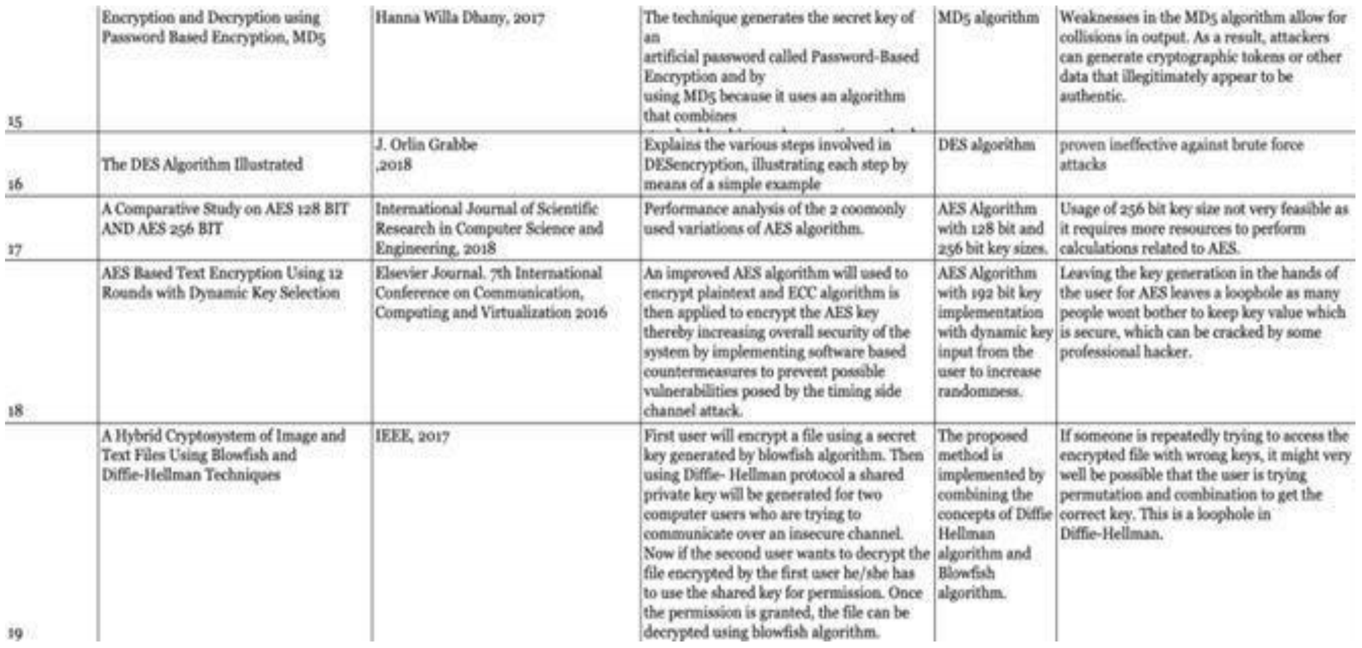
1. **Secure**: As the algorithm is used the encryption and decryption will be secure.
2. **Confidentiality**:This applicationcan help in sending the confidential message without any attacks.

3. **Attacks**: This application can help in the prevention of any type of attack with great ease.
4. **Ease**: This application is easy to use.

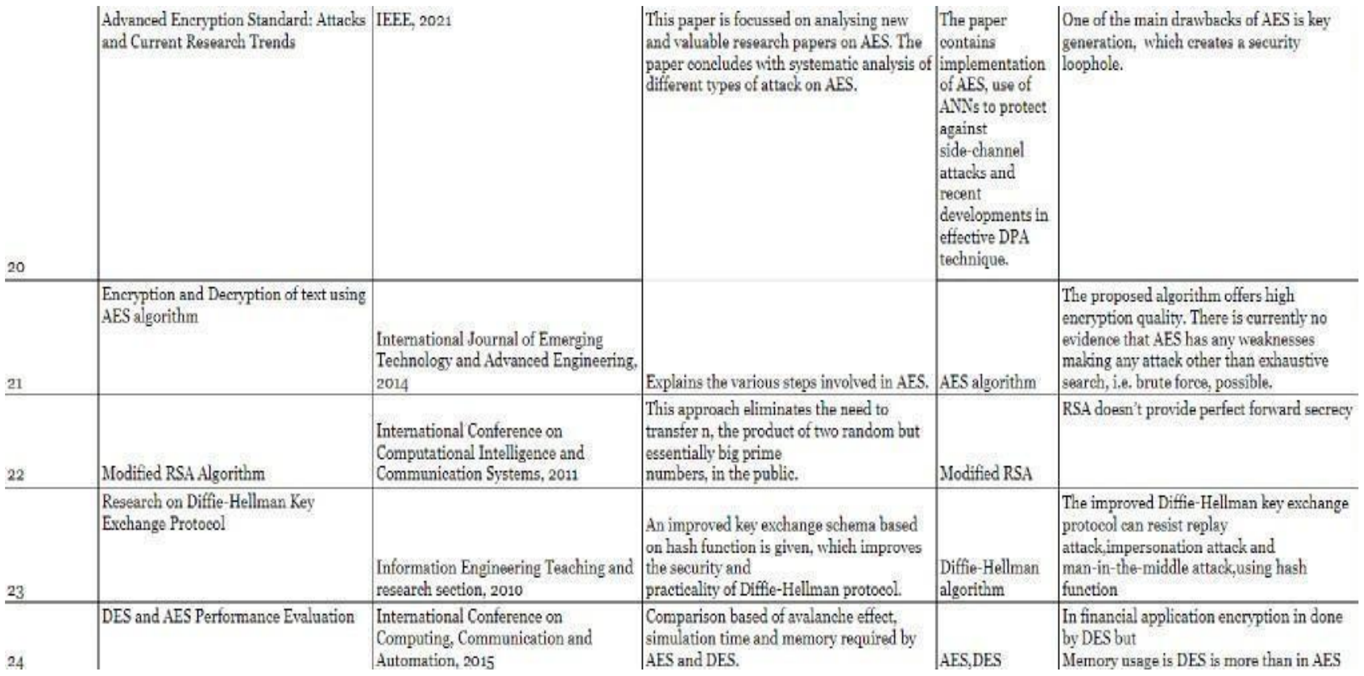
Literature Review

	Paper Title	Journal name and year of publication	Work done	Technique used	Disadvantages/ Gaps identified
1	Strong Diffie-Hellman-DSA Key Exchange	IEEE Communications Letters 21 May 2007	Key exchange scheme based on the DSA signature scheme	Used DSA signature scheme for key exchange in Deffie Hellman Algorithm	Requires a lot of time to authenticate as the verification process includes complicated remainder operators.
2	A study on diffie-hellman key exchange protocols	International Journal of Pure and Applied Mathematics May 2017	Talks about the various attacks Diffie Hellman Algorithm is susceptible to.	Evaluates different attacks Diffie Hellman Algo is susceptible to.	Susceptible to Key Compromise Impersonation attacks.
3	Enhanced diffie-hellman algorithm for reliable key exchange	IOP Conference Series Material Science and Engineering November 2017	Using the Diffie-Hellman algorithm two times to generate a very strong shared-secret key and exchanging it by multiplying it with a random parameter so that it can generate a different key each time for the same message	Using diffie hellman on the key generated using diffie hellman to make it doubly secure.	Once it becomes widely used, it'll again be prone to attacks that deffie hellman was prone to.
4	Implementation of Advanced Encryption Standard Algorithm	International Journal of Scientific and Engineering Research March 2012	Implemented AES algorithm	Sub bytes, Shift rows, Mix Columns, Add round key	Simple algebraic structure, Every block is always encrypted the same way.
5	privateDH: An Enhanced Diffie-Hellman Key-Exchange Protocol using RSA and AES Algorithm	Ripon Patgiri, Senior Member, IEEE 17 May 2021	Integrating Diffie-Hellman algorithm with RSA and AES to make it more secure	Diffie-Hellman algorithm, RSA, AES	There is a computation overhead.
6	Data Encryption using Advanced Encryption Standard with Key Generation by Elliptic Curve DiffieHellman	Samiksha Sharma, International Journal of Security and Its Applications 2017	Data Encryption using AES	AES	AES in counter mode is complex to implement in software taking both performance and security into considerations
7	Diffie-Hellman:Key Exchange and Public Key Cryptosystems	Sivanagaswathi Kallam 9/30/2015	Complete study on Diffie Hellman Key Exchange algorithm	Diffie Hellman Key Exchange algorithm	It is easily susceptible to man-in-the-middle attacks.
8	The RSA Algorithm	Evgeny Milanov 3 June 2009	Complete study on RSA algorithm	RSA algorithm	RSA is slower than certain other symmetric cryptosystems. A major threat to RSA would be a solution to the Riemann hypothesis.
9	Miller-Rabin	Rajesh Pavuluru	Implementation of Miller Rabin algorithm for Primality Testing	Miller Rabin Algorithm	Test fails for ¼ of all bases coprime to n
10	A Study of Encryption Algorithms AES, DES and RSA for Security	Global Journal of Computer Science and Technology Network, Web & Security 2013	Comparison of DES,AES and RSA encryption algorithms	AES , DES , RSA	RSA is more computationally intensive than AES, and much slower. AES is more secure than DES
11	Methods of Primality Testing	MIT Undergraduate Journal of Mathematics	Comparison of different primality testing algorithms	CPT,FLT,SPT(Miller Rabin Test),Lucas Test	All tests are slower and less accurate than Miller Rabin test
12	International Data Encryption Algorithm(IDEA) - A typical illustration	Journal of Global Research in Computer Science 7, July 2011	Complete study on IDEA algorithm	IDEA algorithm	IDEA is too strong AES shows a higher throughput than IDEA
13	Optimized Blowfish Encryption Technique	Christina L , Joe Irudayaraj V S, 2014	Using Blowfish algorithm to find the encryption and decryption time using various key sizes.	Blowfish encryption algorithm	It has weakness in the decryption process over other algorithms in terms of time consumption and serially in throughput
14	HMAC: Keyed-Hashing for Message Authentication	H. Krawczyk, M. Bellare, R. Canetti 1997	Describes HMAC, a mechanism for message authentication using cryptographic hash functions.	HMAC encryption algorithm	If the length of key is less than l-bits, the strength of the keyed IV is reduced. A periodic refreshment of keys is required



15	Encryption and Decryption using Password Based Encryption, MD5	Hanna Willa Dhany, 2017	The technique generates the secret key of an artificial password called Password-Based Encryption and by using MD5 because it uses an algorithm that combines	MD5 algorithm	Weaknesses in the MD5 algorithm allow for collisions in output. As a result, attackers can generate cryptographic tokens or other data that illegitimately appear to be authentic.
16	The DES Algorithm Illustrated	J. Orlin Grabbe ,2018	Explains the various steps involved in DESencryption, illustrating each step by means of a simple example	DES algorithm	proven ineffective against brute force attacks
17	A Comparative Study on AES 128 BIT AND AES 256 BIT	International Journal of Scientific Research in Computer Science and Engineering, 2018	Performance analysis of the 2 coomonly used variations of AES algorithm.	AES Algorithm with 128 bit and 256 bit key sizes.	Usage of 256 bit key size not very feasible as it requires more resources to perform calculations related to AES.
18	AES Based Text Encryption Using 12 Rounds with Dynamic Key Selection	Elsevier Journal. 7th International Conference on Communication, Computing and Virtualization 2016	An improved AES algorithm will used to encrypt plaintext and ECC algorithm is then applied to encrypt the AES key thereby increasing overall security of the system by implementing software based countermeasures to prevent possible vulnerabilities posed by the timing side channel attack.	AES Algorithm with 192 bit key implementation with dynamic key input from the user to increase randomness.	Leaving the key generation in the hands of the user for AES leaves a loophole as many people wont bother to keep key value which is secure, which can be cracked by some professional hacker.
19	A Hybrid Cryptosystem of Image and Text Files Using Blowfish and Diffie-Hellman Techniques	IEEE, 2017	First user will encrypt a file using a secret key generated by blowfish algorithm. Then using Diffie- Hellman protocol a shared private key will be generated for two computer 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/she has to use the shared key for permission. Once the permission is granted, the file can be decrypted using blowfish algorithm.	The proposed method is implemented by combining the concepts of Diffie Hellman algorithm and Blowfish algorithm.	If someone is repeatedly trying to access the encrypted file with wrong keys, it might very well be possible that the user is trying permutation and combination to get the correct key. This is a loophole in Diffie-Hellman.



20	Advanced Encryption Standard: Attacks and Current Research Trends	IEEE, 2021	This paper is focussed on analysing new and valuable research papers on AES. The paper concludes with systematic analysis of different types of attack on AES.	The paper contains implementation of AES, use of ANNs to protect against side-channel attacks and recent developments in effective DPA technique.	One of the main drawbacks of AES is key generation, which creates a security loophole.
21	Encryption and Decryption of text using AES algorithm	International Journal of Emerging Technology and Advanced Engineering, 2014	Explains the various steps involved in AES.	AES algorithm	The proposed algorithm offers high encryption quality. There is currently no evidence that AES has any weaknesses making any attack other than exhaustive search, i.e. brute force, possible.
22	Modified RSA Algorithm	International Conference on Computational Intelligence and Communication Systems, 2011	This approach eliminates the need to transfer n, the product of two random but essentially big prime numbers, in the public.	Modified RSA	RSA doesn't provide perfect forward secrecy
23	Research on Diffie-Hellman Key Exchange Protocol	Information Engineering Teaching and research section, 2010	An improved key exchange schema based on hash function is given, which improves the security and practicality of Diffie-Hellman protocol.	Diffie-Hellman algorithm	The improved Diffie-Hellman key exchange protocol can resist replay attack,impersonation attack and man-in-the-middle attack,using hash function
24	DES and AES Performance Evaluation	International Conference on Computing, Communication and Automation, 2015	Comparison based of avalanche effect, simulation time and memory required by AES and DES.	AES,DES	In financial application encryption in done by DES but Memory usage is DES is more than in AES

Observations made

- Diffie-Hellman is prone to attacks: It is prone to several attacks making it unreliable for key exchange.

- Most of the time, an insecure channel can be detected initially without losing any crucial data.

- A man in the middle (MITM) attack is a general term for when a perpetrator positions himself in a conversation between a user and an application—either to eavesdrop or to impersonate one of the parties, making it appear as if a normal exchange of information is underway.

- The goal of an attack is to steal personal information, such as login credentials, account details, and credit card numbers. Targets are typically the users of financial applications, SaaS businesses, e-commerce sites, and other websites where logging in is required.

- AES+Diffie-Hellman gives more security: We found that using Diffie hellman for key exchange gives symmetric algorithms like AES more security.
- Our Solution: We will be modifying the Diffie Hellman algorithm by adding an extra layer of security using a Machine Learning model.

Techniques Identified

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:

Machine Learning Model

A machine learning model can be created based on network characteristics that predict whether a medium is safe or not. This will help analyze at an initial stage whether to send data or not.

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 that 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 the AES encryption technique.

Neither the sender nor the receiver needs any prior knowledge of each other is 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.

AES Encryption

It is an encryption technique that 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 safety by encrypting using the 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.

Proposed model

As the Diffie-Hellman algorithm is at risk of certain attacks and in the case of AES producing a tougher secret key isn't a problem. So, the concept is to first analyze whether the medium is a safe channel and then make the Diffie-Hellman algorithm more secure by generating two secret keys using two sets of prime numbers and then performing an XOR operation on the two secret keys to generate xorkey. Then 2 new secret keys are generated by taking the modulus of original secret keys with xorkey to get NewKey1 and Newkey2. Then XOR is done again on these NewKey1 and NewKey2 to get Final Shared Secret Key.

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:
First, we run the model on the network parameters and determine whether it is a safe channel with an accuracy of X.
Then we run the Modified Diffie Hellman Algorithm where:
The sender and receiver pick a prime number 'p' and its 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 the sender, it is equal to
And for the receiver it's
$A=g^a \bmod p$

$B = g^b \bmod p$

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

And similarly for the receiver
$S1 = B^a \bmod p$
$S1 = A^b \bmod p$

Where S1 represents the first secret key generated.

Similarly, S2 is generated using another prime number and its primitive root and following the same steps.

Now,

We calculate xorkey by performing XOR operation on S1 and S2 xorkey = S1 ^ S2
Now we calculate NewKey1 and NewKey2 by taking S1 modulus with xorkey and S2 modulus with xorkey

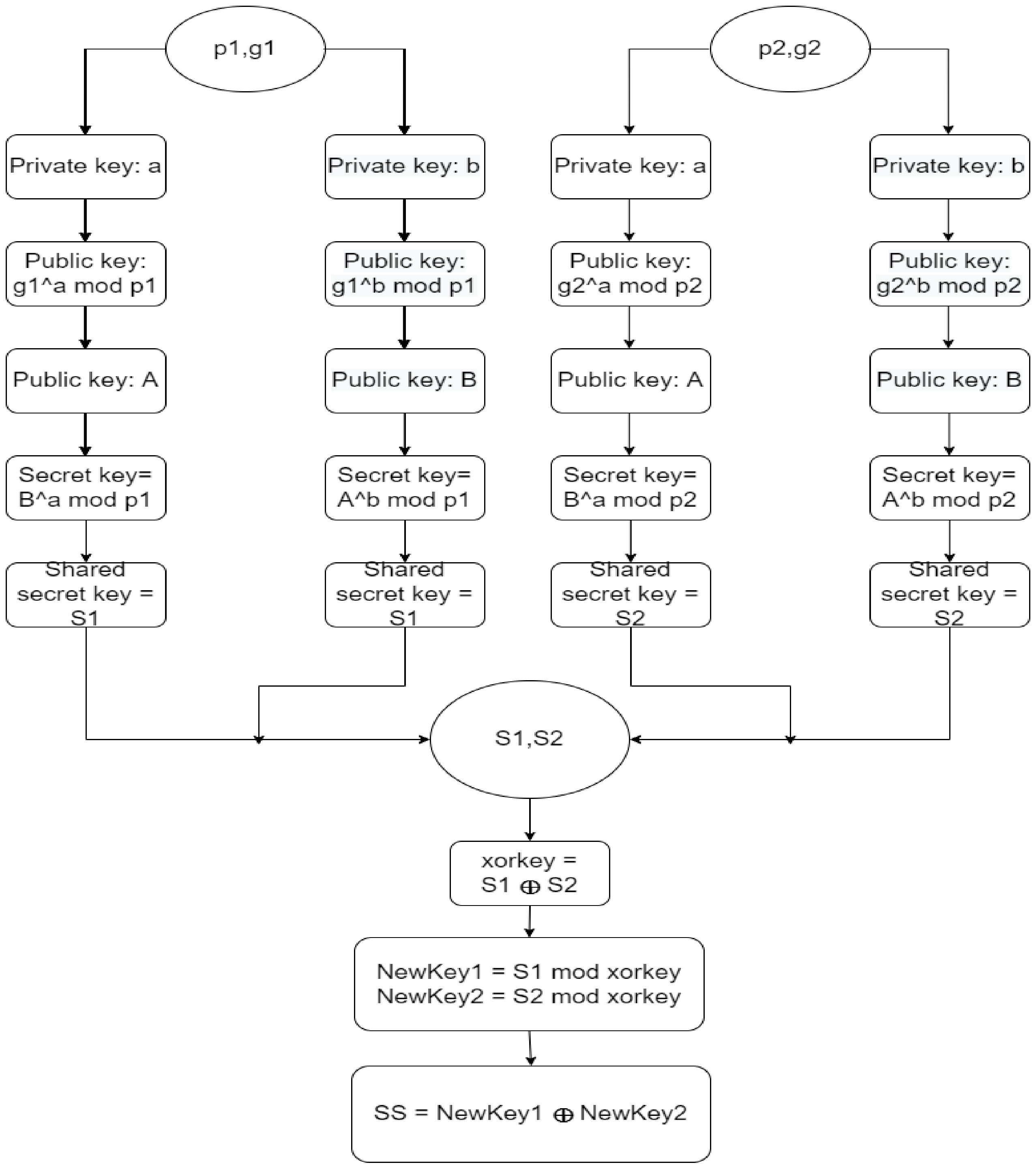
NewKey1 = S1 mod xorkey
NewKey2 = S2 mod xorkey

Now finally we generate the Final Secret key by taking XOR of the new keys

SS = NewKey1 ^ NewKey2

This key is used for AES encryption.

Work Flowchart



Key Words

Diffie Hellman

Encryption

Secret key AES

Encryption

Codes

1. DiffieHellman.py

```
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 the public 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
```

2. 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):
        generator1, prime1 = self.__receive_generator_and_prime(con)
        diffie_hellman1 = DiffieHellman(generator1, prime1)
        result1 = diffie_hellman1.get_result()
        result_from_client1 = int(con.recv(1024))
        con.send(bytes(str(result1), 'utf8'))
        self.__key1 = diffie_hellman1.calculate_shared_secret(result_from_client1)
        generator2, prime2 = self.__receive_generator_and_prime(con)
```

```
            diffie_hellman2 = DiffieHellman(generator2, prime2)
            result2 = diffie_hellman2.get_result()
            result_from_client2 = int(con.recv(1024))
            con.send(bytes(str(result2), 'utf8'))
                                        self__ key2          =
diffie_hellman2.calculate_shared_secret(result_from_client2)
            xorkey = self.__key1 ^ self___key2
            newk1 = self.__key1%xorkey
            newk2 = self.__key2%xorkey
            self.__key = newk1^newk2
            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()
```

3. client.py

```
import socket
from Math import Math
from DiffieHellman import DiffieHellman
import pyaes
import tkinter
class Client:
    def __init__(self, host, port):
        destination = (host, port)
        self.__tcp_client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.__tcp_client.connect(destination)
    def establish_session(self):
        generator1, prime1 = Math.generate_generator_and_prime(256)
        self.__tcp_client.send(bytes(str(generator1), 'utf8'))
        self.__tcp_client.send(bytes(str(prime1), 'utf8'))
        diffie_hellman1 = DiffieHellman(generator1, prime1)
        result1 = diffie_hellman1.get_result()
        self.__tcp_client.send(bytes(str(result1), 'utf8'))
        result_from_server1 = int(self.__tcp_client.recv(1024))
```

```
                                                           self.__key1             =
diffie_hellman1.calculate_shared_secret(result_from_server1)
        generator2, prime2 = Math.generate_generator_and_prime(256)
        self.__tcp_client.send(bytes(str(generator2), 'utf8'))
        self.__tcp_client.send(bytes(str(prime2), 'utf8'))
        diffie_hellman2 = DiffieHellman(generator2, prime2)
        result2 = diffie_hellman2.get_result()
        self.__tcp_client.send(bytes(str(result2), 'utf8'))
        result_from_server2 = int(self.__tcp_client.recv(1024))
                                                           self.__key2             =
diffie_hellman2.calculate_shared_secret(result_from_server2)
        xorkey = self.__key1 ^ self __ key2
        newk1 = self.__key1%xorkey
        newk2 = self.__key2%xorkey
        self.__key = newk1^newk2
        print("Session key: " + str(self.__key))
        self.__key = self.__key.to_bytes(32, byteorder = "big")
        self.__aes = pyaes.AESModeOfOperationCTR(self.__key)
    def send_message(self, message):
        message = self.__aes.encrypt(message)
        print("Encrypted message: " + str(message))
        self.__tcp_client.send(message)
if __name___== "__main__":
    def send(event=None): # event is passed by binders.
#Handles sending of messages
        msg = my_msg.get()
        my_msg.set("") # Clears input field.
        msg_list.insert("end",msg)
        client.send_message(msg)
        if msg == "{quit}":
             top.quit()
    def on_closing(event=None):
#This function is to be called when the window is closed.
        my_msg.set("{quit}")
        send()
    top = tkinter.Tk()
    top.title("Chatter")
    messages_frame = tkinter.Frame(top)
    my_msg = tkinter.StringVar() # For the messages to be sent.
    my_msg.set("Type your messages here.")
      scrollbar  =  tkinter.Scrollbar(messages_frame)  #  To  navigate  through
pastmessages.
# Following will contain the messages.
    msg_list = tkinter.Listbox(messages_frame, height=15, width=50,
    yscrollcommand=scrollbar.set)
    scrollbar.pack(side=tkinter.RIGHT, fill=tkinter.Y)
    msg_list.pack(side=tkinter.LEFT, fill=tkinter.BOTH)
    msg_list.pack()
    messages_frame.pack()
    entry_field = tkinter.Entry(top, textvariable=my_msg)
    entry_field.bind("<Return>",  send)
    entry_field.pack()
    send_button = tkinter.Button(top, text="Send", command=send)
    send_button.pack()
    top.protocol("WM_DELETE_WINDOW", on_closing)
    client = Client("localhost", 5000)
    client.establish_session()
    tkinter.mainloop()
```

4. Math.py

```
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  exponentiation  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
            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
```

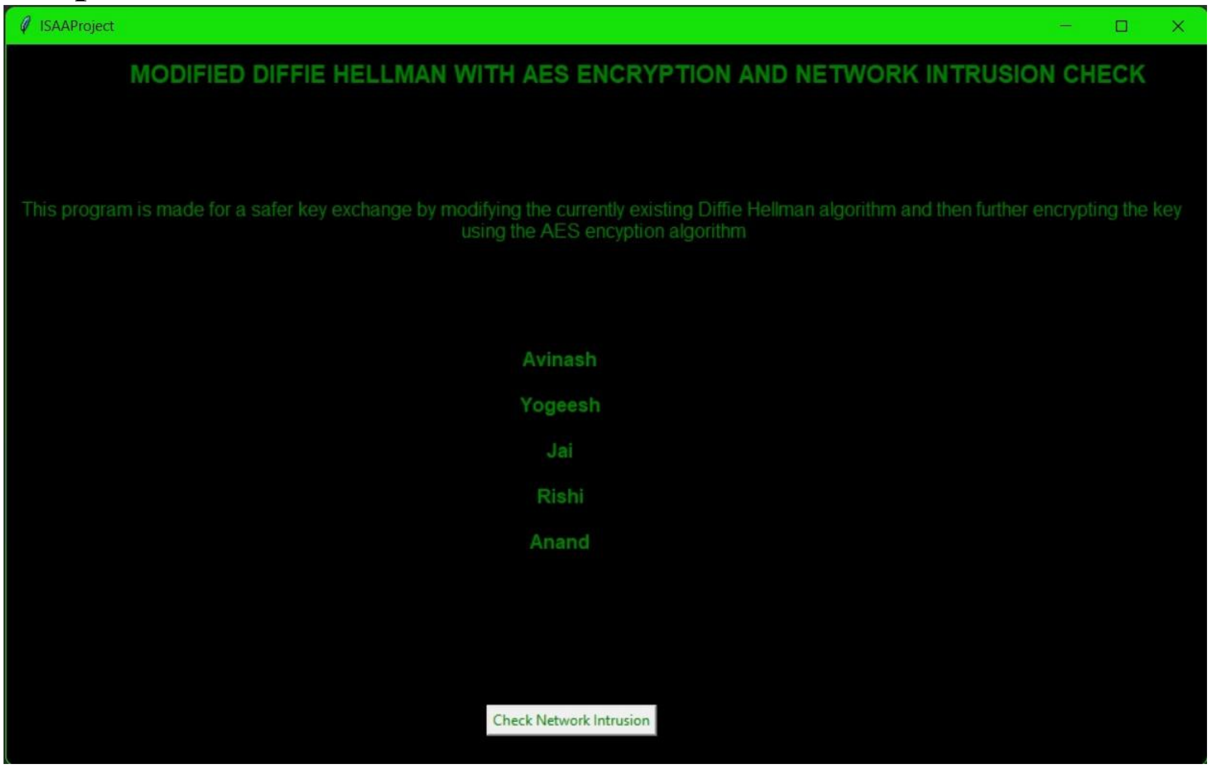
```
        if number == 3 or number == 2:
            return True # random.randrange(2, number - 1) throws and exception
when number is either 2 or 3.
        if number <= 1 or number % 2 == 0:
            if(number != 2):
                return False # immediately discarts if number is even different
from 2 orless 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
defaultbits_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()
```

5. UI.py

```
from tkinter import *
from tkinter import messagebox as mb
from client import Client
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
import tensorflow
import pickle
from xgboost import XGBClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
import random
a=Tk()
a.title("ISAAProject")
```

```
a.geometry("1000x600")
a.configure(bg="black")
#Heading
head=Label(a,text="MODIFIED DIFFIE HELLMAN WITH AES ENCRYPTION AND NETWORK
INTRUSION CHECK")
head.config(font=("Helvetica bold",15,"bold"),fg="green",bg="black")
head.place(x=100,y=10)
#Explanation'
s="This program is made for a safer key exchange by modifying the currently
existing Diffie Hellman algorithm and then further encrypting the key \nusing
the AES encyption algorithm"
exp=Label(a,text=s)
exp.config(font=('Helvetica bold',12),fg="green",bg="black")
exp.place(x=10,y=125)
#Our names
names=Label(a,text="Avinash\n\nYogeesh\n\nJai\n\nRishi\n\nAnand")
names.config(font=("Helvetica bold",12,"bold"),fg="green",bg="black")
names.place(x=425,y=250)
def net():
    print("Inside network check")
    a.destroy()
    b=Tk()
    b.title("Network Intrusion Check")
    b.geometry("500x500")
    b.configure(bg="black")
    b.focus_force()
    #Heading
    head=Label(b,text="NETWORK INTRUSION CHECK")
    head.config(font=("Helvetica bold",15,"bold"),fg="green",bg="black")
    head.place(x=110,y=10)
    #Inputting name of user
    lname=Label(b,text="Name")
    lname.config(font=("Helvetica bold",12),fg="green",bg="black")
    lname.place(x=125,y=150)
    ename=Entry(b)
    ename.place(x=190,y=150)
    def netcheck():
        print("Inside network check function")
        name=ename.get()
        dataset = pd.read_csv('Train_data.csv')
        y = dataset['class']
        dataset.drop(columns=['class','src_bytes','land','num_failed_logins','
urgent','srv_count'], inplace=True)
        y = pd.DataFrame(y)
        columns = ['protocol_type', 'service', 'flag']
        lb = LabelEncoder()
        for i in columns:
            dataset[i] = lb.fit_transform(dataset[i])
        y = lb.fit_transform(y)
        y=pd.DataFrame(y)
           X_train,  X_test,  y_train,  y_test  =  train_test_split(dataset,  y,
test_size =0.25, random_state = 0)
        model = XGBClassifier()
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        cm = confusion_matrix(y_test, y_pred)
        print(cm)
        print(accuracy_score(y_test, y_pred))
        num = random.randint(0, 6298)
        print(y_pred)
#print(np.array(X_test)[num])
        x = y_pred[num];
        if(x):
            mb.showinfo("Network Intrusion Check","Network is Safe")
            b.destroy()
#We call client.py here
            def send(event=None): # event is passed by binders.
#Handles sending of messages.
                msg = my_msg.get()
                my_msg.set("") # Clears input field.
                msg_list.insert("end",name+" : "+msg)
                client.send_message(msg)
                if msg == "{quit}":
                     top.quit()
            def on_closing(event=None):
#This function is to be called when the window is closed.
                my_msg.set("{quit}")
                send()
            top = Tk()
            top.title("Chatter")
            top.configure(bg="white")
            messages_frame = Frame(top)
            my_msg = StringVar() # For the messages to be sent.
            my_msg.set("Type your messages here.")
             scrollbar  =  Scrollbar(messages_frame)  #  To  navigate  through  past
messages.
# Following will contain the messages.
            msg_list = Listbox(messages_frame, height=15, width=50,
            yscrollcommand=scrollbar.set)
            scrollbar.pack(side=RIGHT, fill=Y)
            msg_list.pack(side=LEFT, fill=BOTH)
            msg_list.pack()
            msg_list.configure(bg="black",fg="green")
            messages_frame.pack()
            entry_field = Entry(top, textvariable=my_msg)
            entry_field.bind("<Return>", send)
            entry_field.pack()
            entry_field.configure(fg="green")
            send_button = Button(top, text="Send", command=send)
            send_button.pack()
            send_button.configure(fg="green")
            s="Welcome "+name
            msg_list.insert("end",s)
            top.protocol("WM_DELETE_WINDOW", on_closing)
            print("Generating key...")
            client = Client("localhost", 5000)
            client.establish_session()
            top.mainloop()
        else:
            mb.showerror("Network Intrusion Check","Network is not safe")
            b.destroy()
                         butnetintcheck=Button(b,text="Check            Network
Intrusion",command=netcheck).place(x=185,y=400)
butnetcheck=Button(a,text="Check Network Intrusion",command=net)
butnetcheck.config(fg="green")
butnetcheck.place(x=400,y=550)
a.mainloop()
```

Output Screenshots



MODIFIED DIFFIE HELLMAN WITH AES ENCRYPTION AND NETWORK INTRUSION CHECK

This program is made for a safer key exchange by modifying the currently existing Diffie Hellman algorithm and then further encrypting the key using the AES encyption algorithm

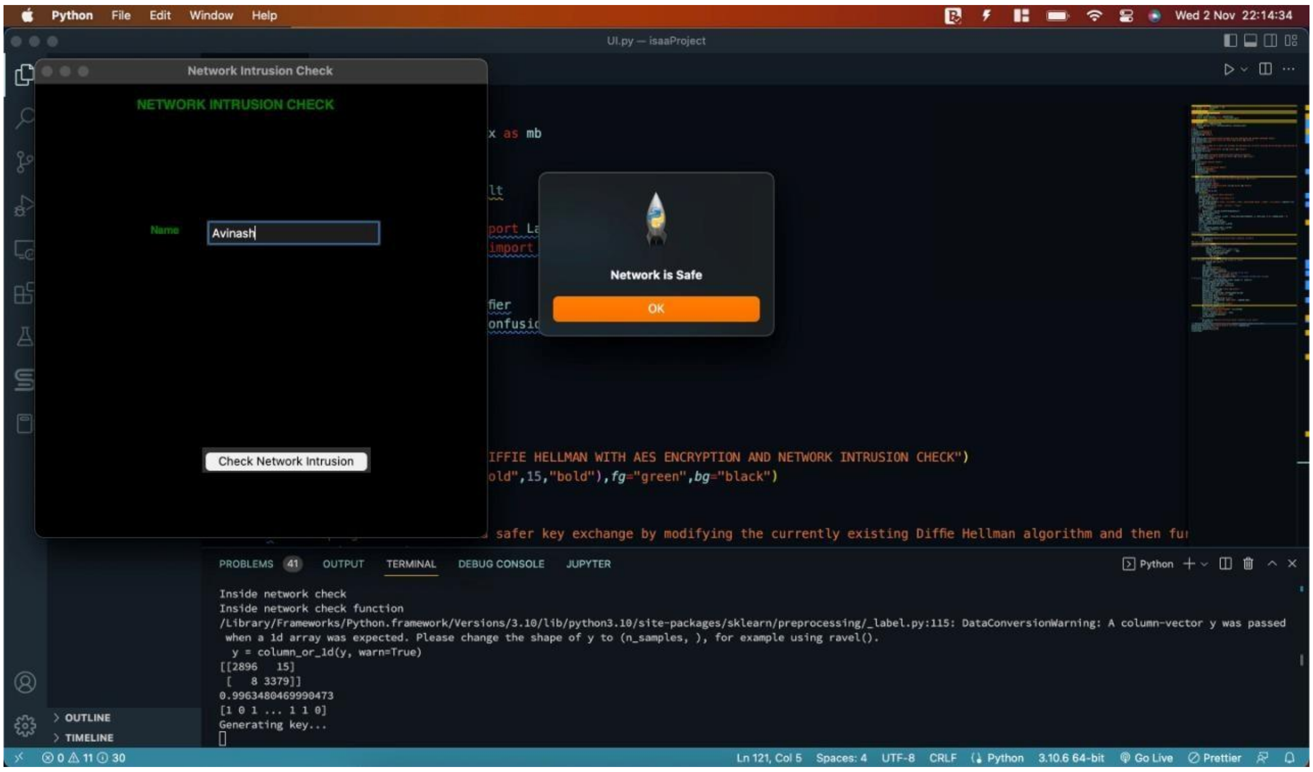
Avinash

Yogeesh

Jai

Rishi

Anand

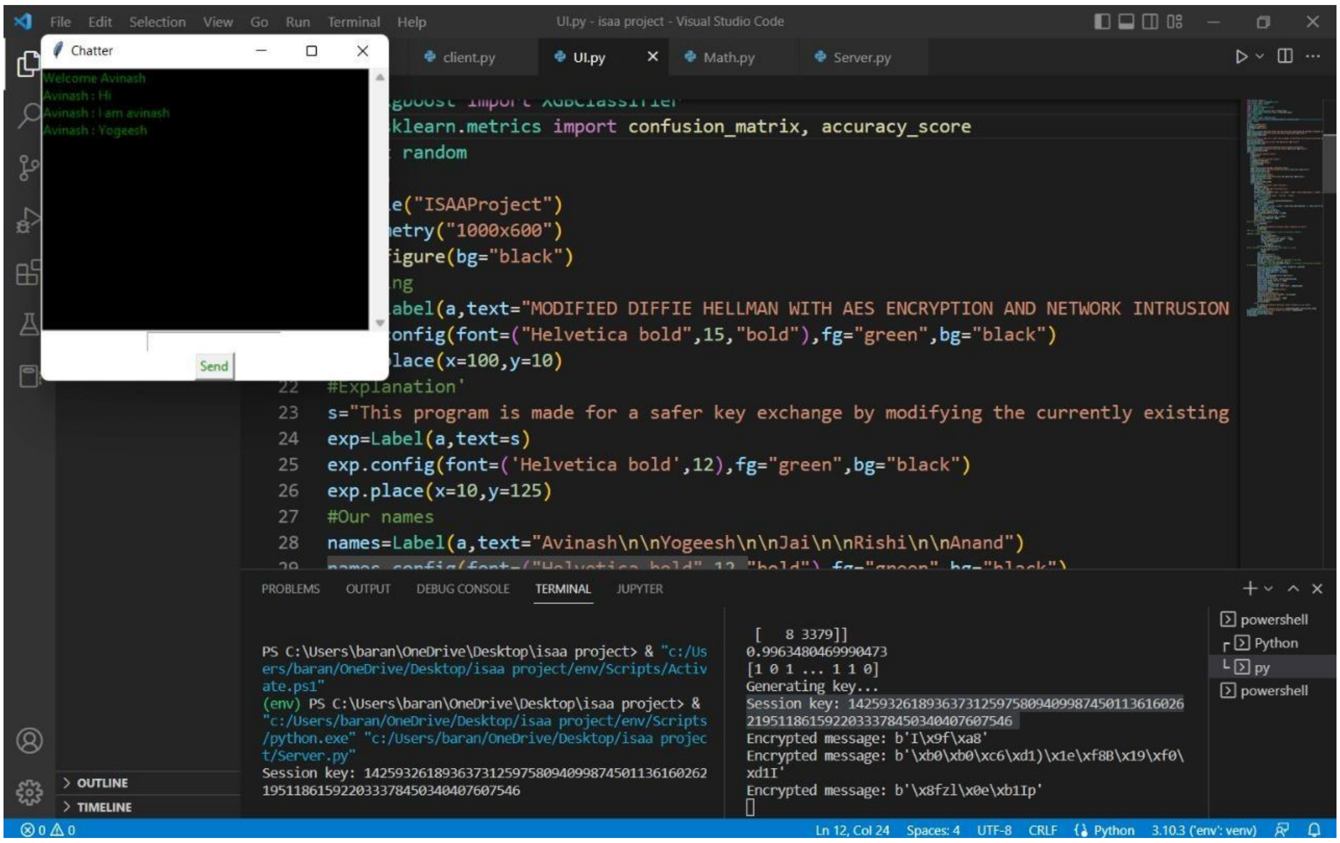


Name: Avinash

Network is Safe

```
x as mb

lt

port La
import

fier
onfusio

IFFIE HELLMAN WITH AES ENCRYPTION AND NETWORK INTRUSION CHECK")
old",15,"bold"),fg="green",bg="black")

 safer key exchange by modifying the currently existing Diffie Hellman algorithm and then fu
```

```
Inside network check
Inside network check function
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/sklearn/preprocessing/_label.py:115: DataConversionWarning: A column-vector y was passed
 when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
[[2896   15]
 [   8 3379]]
0.9963480469990473
[1 0 1 ... 1 1 0]
Generating key...
```



```
Welcome Avinash
Avinash : Hi
Avinash : I am avinash
Avinash : Yogeesh
```

```
gboost import XGBClassifier
klearn.metrics import confusion_matrix, accuracy_score
 random

e("ISAAProject")
etry("1000x600")
figure(bg="black")
ng
abel(a,text="MODIFIED DIFFIE HELLMAN WITH AES ENCRYPTION AND NETWORK INTRUSION
onfig(font=("Helvetica bold",15,"bold"),fg="green",bg="black")
lace(x=100,y=10)
#Explanation'
s="This program is made for a safer key exchange by modifying the currently existing
exp=Label(a,text=s)
exp.config(font=('Helvetica bold',12),fg="green",bg="black")
exp.place(x=10,y=125)
#Our names
names=Label(a,text="Avinash\n\nYogeesh\n\nJai\n\nRishi\n\nAnand")
names.config(font=("Helvetica bold",12,"bold"),fg="green",bg="black")
```

```
PS C:\Users\baran\OneDrive\Desktop\isaa project> & "c:/Us
ers/baran/OneDrive/Desktop/isaa project/env/Scripts/Activ
ate.ps1"
(env) PS C:\Users\baran\OneDrive\Desktop\isaa project> &
"c:/Users/baran/OneDrive/Desktop/isaa project/env/Scripts
/python.exe" "c:/Users/baran/OneDrive/Desktop/isaa projec
t/Server.py"
Session key: 142593261893637312597580940998745011361602621
95118615922033378450340407607546
```

```
 [   8 3379]]
0.9963480469990473
[1 0 1 ... 1 1 0]
Generating key...
Session key: 14259326189363731259758094099874501136160262
195118615922033378450340407607546
Encrypted message: b'I\x9f\xa8'
Encrypted message: b'\xb0\xb0\xc6\xd1)\x1e\xf8B\x19\xf0\
xd1I'
Encrypted message: b'\x8fzl\x0e\xb1Ip'
```

Future Scope

As security is growing day by day attackers are also being more cognizant. Each security schema has some weak points i.e. if the attacker knew them then he can bypass security. So to make system more secure we can work on the weakness of algorithm and can further enhance the security

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 its management. Provision of security to the user's data on the cloud empowers the Data owner to outsource the data to save.

References

1. Jeong, I. R., Kwon, J. O., & Lee, D. H. (2007). Strong diffie-hellman-DSA key exchange. IEEE communications letters, 11(5), 432-433.

2. Mishra, M. R., & Kar, J. (2017). A study on diffie-hellman key exchange protocols. International Journal of Pure and Applied Mathematics, 114(2), 179-189.

3. Kumar, C., & Vincent, P. D. R. (2017, November). Enhanced diffie-hellman algorithm for reliable key exchange. In IOP conference series: materials science and engineering (Vol. 263, No. 4, p. 042015). IOP Publishing.

4. Pitchaiah, M., & Daniel, P. (2012). Implementation of advanced encryption standard algorithm.

5.Patgiri, R. (2021). privateDH: An Enhanced Diffie-Hellman Key-Exchange Protocol using RSA and AES Algorithm. Cryptology ePrint Archive.

6. Kallam, S. (2015). Diffie-hellman: key exchange and public key cryptosystems. Master degree of Science, Math and Computer Science, Department of India State University, USA, 5-6.

7. Sharma, S., & Chopra, V. (2017). Data encryption using advanced encryption standard with key generation by elliptic curve

diffie-hellman. International Journal of Security and Its Applications, 11(3), 17-28.

8.Milanov, E. (2009). The RSA algorithm. RSA laboratories, 1-11.

9. Christina, L., & Joe Irudayaraj, V. S. (2014). Optimized Blowfish encryption technique. International Journal of Innovative Research in Computer and Communication Engineering, 2(7), 5009-5015.

10. Krawczyk, H., Bellare, M., & Canetti, R. (1997). HMAC: Keyed-hashing for message authentication.

11. Dhany, H. W., Izhari, F., Fahmi, H., Tulus, M., & Sutarman, M. (2017, October). Encryption and decryption using password based encryption, MD5, and DES. In International conference on public policy, social computing and development (Vol. 2018).

12. Grabbe, J. O. (2010). The DES algorithm illustrated.

13. Hazra, T. K., Mahato, A., Mandal, A., & Chakraborty, A. K. (2017, August). A hybrid cryptosystem of image and text files using blowfish and Diffie-Hellman techniques. In 2017 8th Annual Industrial Automation and Electromechanical Engineering Conference (IEMECON) (pp. 137-141). IEEE.

14. Kaur, J., Lamba, S., & Saini, P. (2021, March). Advanced Encryption Standard: Attacks and Current Research Trends. In 2021 International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE) (pp. 112-116). IEEE.

15. Mathur, N., & Bansode, R. (2016). AES based text encryption using 12 rounds with dynamic key selection. Procedia Computer Science, 79, 1036-1043.

16. Guy-Cedric, T. B. I. (2018). SR,"A Comparative Study on AES 128 BIT AND AES 256 BIT,". Int. J. Sci. Res. Comput. Sci. Eng, 6(4), 30-33.

17. Pavuluru, R. (2015). Miller-Rabin.

18. Mahajan, P., & Sachdeva, A. (2013). A study of encryption algorithms AES, DES and RSA for security. Global Journal of Computer Science and Technology.

19. Mcgregor-Dorsey, Zachary. (1999). Methods of Primality Testing.

20. Basu, Sandipan. (2011). INTERNATIONAL DATA ENCRYPTION ALGORITHM (IDEA) – A TYPICAL ILLUSTRATION.

21. Padate, R., & Patel, A. (2014). Encryption and decryption of text using AES algorithm. International Journal of Emerging Technology and Advanced Engineering, 4(5), 54-9.

22. Chhabra, A., & Mathur, S. (2011, October). Modified RSA algorithm: a secure approach. In 2011 International Conference on

Computational Intelligence and Communication Networks (pp.545-548). IEEE.

23. Li, N. (2010, April). Research on Diffie-Hellman key exchange protocol. In 2010 2nd International Conference on Computer Engineering and Technology (Vol. 4, pp. V4-634). IEEE.

24. Bhat, B., Ali, A. W., & Gupta, A. (2015, May). DES and AES performance evaluation. In International Conference on Computing, Communication & Automation (pp. 887-890). IEEE.

25. Mahajan, P., & Sachdeva, A. (2013). A study of encryption algorithms AES, DES and RSA for security. Global Journal of Computer Science and Technology.

26. Mcgregor-Dorsey, Zachary. (1999). Methods of Primality Testing.

27. Basu, Sandipan. (2011). INTERNATIONAL DATA

ENCRYPTION ALGORITHM (IDEA) – A TYPICAL ILLUSTRATION.

28. Padate, R., & Patel, A. (2014). Encryption and decryption of text using AES algorithm. International Journal of Emerging

Technology and Advanced Engineering, 4(5), 54-9.

29. Chhabra, A., & Mathur, S. (2011, October). Modified RSA algorithm: a secure approach. In 2011 International Conference on Computational Intelligence and Communication Networks (pp. 545-548). IEEE.

30. Li, N. (2010, April). Research on Diffie-Hellman key exchange protocol. In 2010 2nd International Conference on Computer Engineering and Technology (Vol. 4, pp. V4-634). IEEE.

31. Bhat, B., Ali, A. W., & Gupta, A. (2015, May). DES and AES performance evaluation. In International Conference on

Computing, Communication & Automation (pp. 887-890). IEEE.