e-ISJN: A4372-3114
p-ISJN: A4372-3115

ISSN: 2321-7782 (Online)
ISSN: 2347-1778 (Print)
Impact Factor: 7.327

Volume 8, Issue 6, June 2020

International Journal of Advance Research in Computer Science and Management Studies

Research Article / Survey Paper / Case Study

Available online at: www.ijarcsms.com

# A Novel Text Encryption Algorithm using enhanced Diffie Hellman and AES

**K. Nanda Kishore**[1]
B.Tech III year Student,
Department of IST SET,
Jain Univeristy,
Bengaluru, Karnataka, India

**Sujan Chhetri**[2]
B.Tech III year Student,
Department of IST SET,
Jain University,
Bengaluru, Karnataka, India

***Abstract: Security is one of the vital concerns especially in the current period with an extensive rise in the usage of the internet. So, an effective text encryption algorithm is of greater need for achieving an immense amount of privacy. In this context, we propose a secure text encryption algorithm using an enhanced version of Diffe-Hellman and AES. By improving the Diffie-Hellman key exchange Algorithm can help in generating a secret key that is highly secure and this key is used in the AES algorithm to perform the encryption and decryption operations on a text. The proposed algorithm is implemented in Java and the results are discussed along with a possible explanation.***

## I. INTRODUCTION

One of the main problems that Cryptography tries to solve involves in making communication taking place over an insecure channel safe from unauthorized access. This problem gives rise to the need for a key exchange algorithm that can be used to exchange keys securely over an insecure channel. But sharing the key to the other party over a large teleprocessing network can be costly and time-consuming [1]. This is where Diffie-Hellman came into the picture with the purpose of making two parties exchange a session key[2] which will be then be used in other symmetric encryption algorithms, in our case AES. Despite being widely used Diffie-Hellman is prone to various attacks like Man-in-middle, plain-text, and others. So, we've proposed an algorithm which is an update to the Diffie-Hellman algorithm for generating a highly secure key which can defend against the mentioned attacks above and we then use this key to perform encryption and decryption of the text using AES.

## II. THE DIFFIE-HELLAMAN KEY EXCHANGE ALGORITHM

The Diffie-Hellman key exchange is the most widely used method of generating and exchanging keys via an insecure channel[3]. The Diffie–Hellman key exchange creates a shared secret between the two parties which is used for secret communication for exchanging the data over public networks. If you want to exchange some secret information to another person then the most effective way to do so is encrypting the message with a code and prearrange the type of code and key that you are planning to use in advance or over a safe communication channel. Say, you are sending a message encoded with shift-cipher in which every 'a' becomes 'b' and 'b' becomes 'c' and so on. So let the message be 'node is beautiful' which becomes 'opef jt cfbvjgvm'. Suppose the code is uncracked by the Man in the Middle and is safely received by the next person. But what if you couldn't arrange code in advance with the receiver? Though the message will be secure enough to be decoded by the Man in the Middle but also the receiver won't be able to decode. This problem was solved by the Diffie-Hellman key exchange. The Diffie-Hellman key exchange algorithm allows those who have never met in advance to create a shared key safety, even via an insecure channel.

## 2.1 Algorithm

The Diffie-Hellman Key Exchange algorithm involves the following steps:

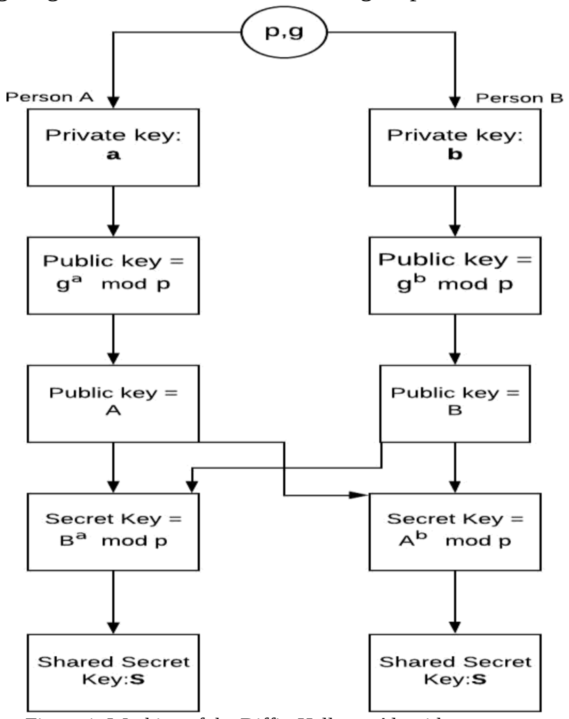


Figure 1: Working of the Diffie-Hellman Algorithm.

- 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 be a and b, this private is only known to them.
- The public key of both the sender and receiver is calculated for sender it is equal to

$$A = g^a mod p.$$

and for receive it's

$$B = g^b mod p.$$

where A and B represent the respective public key.

- The public keys generated are then exchanged by both the sender and receiver.
- Now both the sender and receiver calculate their secret key i.e sender's secret key is be obtained by

$$S = B^a mod p.$$

and similarly for receiver

$$S = A^b mod p.$$

where S represents the secret key generated.

- This 'S' is the shared secret key which can be used in symmetric algorithms for encryption and decryption.

## 2.2 Security overview of Diffie-Hellman algorithm

Diffie-Hellman key exchange is complex to execute and is an important part of a secure mechanism for transferring data online[4]. It is not vulnerable to attacks until it is implemented with the proper authentication method. Security depends on the implementation and the number that is selected for the exchange. Though it's harder for the attacker to find the secret key obtained as the only way is to find out the private keys of both sender and receiver but it’s not impossible. Known attacks like man-in-the- middle, plain-text, and others have proved that the algorithm can be cracked wide open.

## III. Advanced Encryption Standard (AES)

Advanced Encryption Standard(AES) is a symmetric key block cipher technique[5]. It is based on a substitution permutation network. It includes a series of linked operations that involve replacing inputs by specific outputs and some others involve shuffling of bits around or permutations. AES performs the computations on bytes. Hence, it serves the 128 bits of plain-text as 16 bytes. These bytes are arranged into four columns and four rows to be processed as a matrix. The number of rounds is variable and it depends on the length of the key provided. AES uses 10 rounds for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys. These rounds use a different 128-bit round key, which will be calculated from the original AES key.

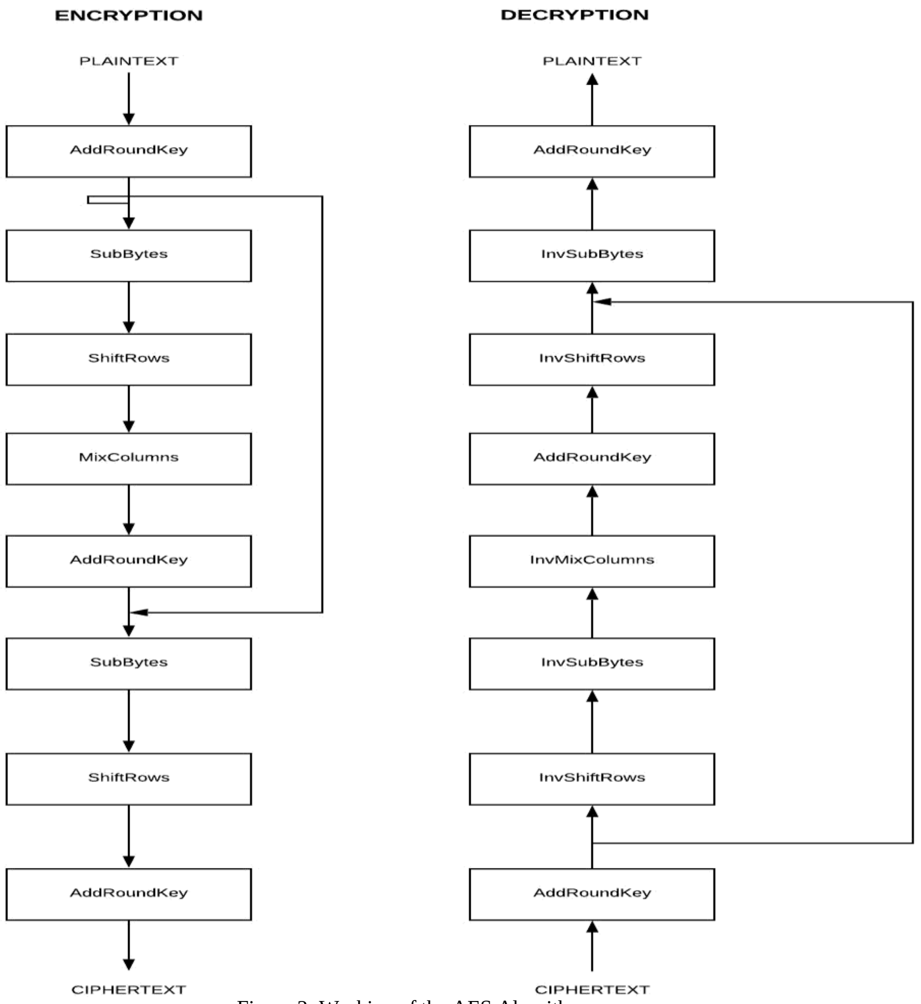


Figure 2: Working of the AES Algorithm.

### 3.1 Algorithm

The AES algorithm has the following steps for encryption and decryption:

- First step involves deriving the set of round keys from the cipher key.
- Then we’ll initialize the state array with the plain text.
- Next step involves adding the initial round key to the starting state array.
- We’ll perform nine rounds of state manipulation.
- We’ll continue until the tenth and final round of state manipulation.
- Then we’ll take the final state array out as the cipher text.

The decryption is similar to the encryption process as the process is just reversed.

### 3.2 Security overview of the AES algorithm

AES uses a key expansion process in which the initial key comes up with a series of newly generated keys called round keys[6]. These round keys are generated over multiple rounds of changes which makes it harder to crack the encryption. The AES encryption is more complex with the increase in rounds of encryption. The 256 bit key gives the highest level of encryption.

## IV. Proposed Algorithm

As we've seen above the Diffie-Hellman algorithm is vulnerable to certain attacks and in the case of AES generating a harder secret key isn't a problem but how to exchange the key generated is the problem though it's not a major issue it's worth making it simple. So, the idea is to first make the Diffie-Hellman more secure and in order to do that, we'll take a primitive root to the secret obtained and take two private keys and perform the algorithm using the secret key obtained to generate a second secret key thereby making it more secure as now the attacker can't perform anything on this because he/she won't know that we're using a primitive root to the secret key obtained. Now, this secret key can be exchanged and used in the AES algorithm to perform the encryption and decryption operations.

### 4.1 Algorithm

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 be a and b, this private is only known to them.
- The public key of both the sender and receiver is calculated, for sender it is equal to

$$A = g^a modp.$$

  and for receiver it's

$$B = g^b modp.$$

  where A and B represent the respective public key.
- The public keys generated are then exchanged by both the sender and receiver.
- Now both the sender and receiver calculate their secret key i.e sender's secret key is be obtained by

$$S = B^a modp.$$

  and similarly for receiver

$$S = A^b modp.$$

  where S represents the secret key generated.

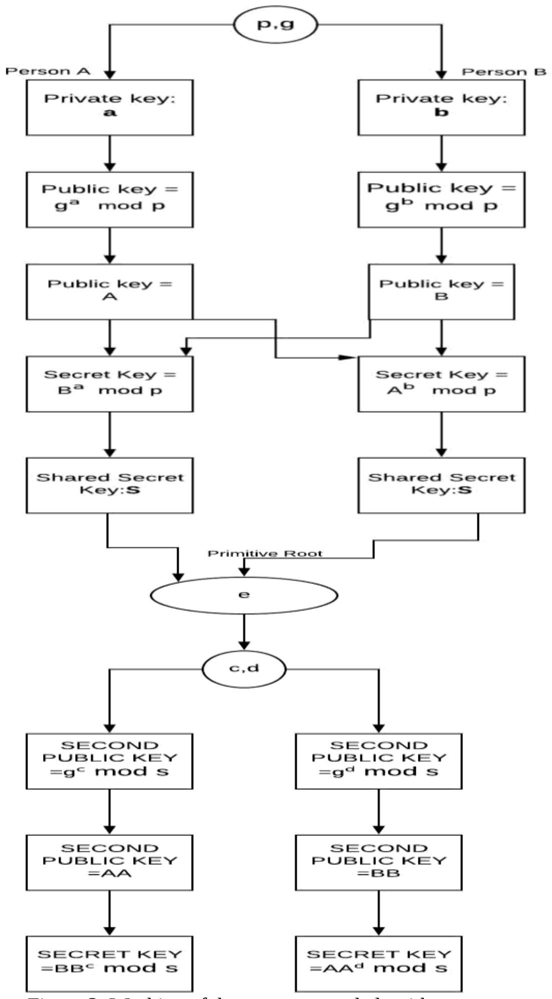


Figure 3: Working of the new proposed algorithm.

- 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

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

- Next step involves deriving the set of round keys from the final secret key obtained above.
- Then we’ll initialize the state array with the plain text.
- Next step involves adding the initial round key to the starting state array.

- We'll perform nine rounds of state manipulation.
- We'll continue until the tenth and final round of state manipulation.
- Then we'll take the final state array out as the cipher text.

The decryption is similar to the encryption process as the process is just reversed.

### 4.2 Security overview of the new proposed algorithm

As mentioned above the newer algorithm i.e, the enhanced version of Diffe-Hellman is much secure and can't be broken using the attacks like man-in-the-middle, plain-text, etc. As here the attacker doesn't have any clue that we're taking a primitive root of the secret key and then using both to generate new public keys and thereby newer secret keys thus making it most secure. The only downside of this algorithm, for now, is that it's execution time is more though the run time is scalable enough.

## V. Implementation

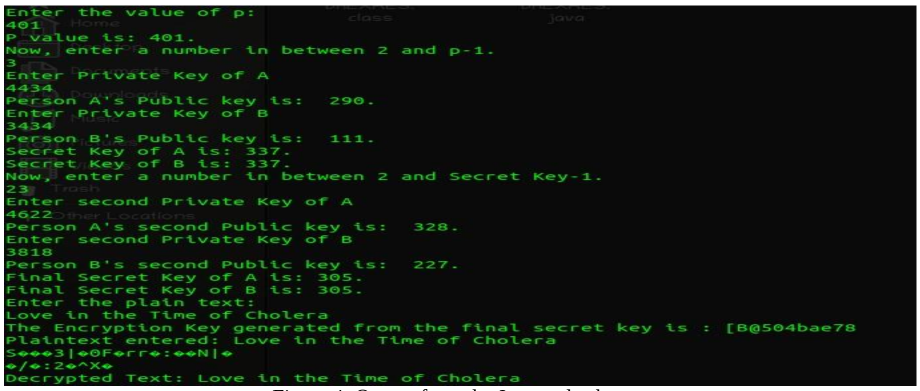
### 5.1 Code

```
public class DHEXAES {
            final static BigInteger one = new BigInteger("1");
            private byte[] key;
      private static final String ALGORITHM = "AES";
      public DHEXAES(byte[] key)
         {
            this.key = key;
         }
      public byte[] encrypt(byte[] plainText) throws Exception
         {
               SecretKeySpec secretKey = new SecretKeySpec(key, ALGORITHM); Cipher cipher =
            Cipher.getInstance(ALGORITHM); cipher.init(Cipher.ENCRYPT_MODE, secretKey);


            return cipher.doFinal(plainText);
         }
      public byte[] decrypt(byte[] cipherText) throws Exception
         {
               SecretKeySpec secretKey = new SecretKeySpec(key, ALGORITHM); Cipher cipher =
            Cipher.getInstance(ALGORITHM); cipher.init(Cipher.DECRYPT_MODE, secretKey);


            return cipher.doFinal(cipherText);
         }
```

```
                    public static void main(String args[]) throws Exception {

                Scanner sc = newScanner(System.in); BigInteger pVal;

            // Prime Number Input from User(p). System.out.println("Enter the value of p:");
            String ans = sc.next();

                pVal = getNextPrime(ans); System.out.println("P value is: "+pVal+".");

            // Primitive Number Input from User(g).

            System.out.println("Now, enter a number in between2 and p-1.");

                BigInteger gVal = new BigInteger(sc.next());

            // Private key for A from user. System.out.println("Enter Private Key of A");
            BigInteger prA = new BigInteger(sc.next());

            // Public key of A generated.

                BigInteger puA = gVal.modPow(prA,pVal);

            // This is the value that will get sent from A toB.

            System.out.println("Person A's Publickey is:                "+puA+ ".");

            // Private key for B from user. System.out.println("Enter Private Key of B");
            BigInteger prB = new BigInteger(sc.next());

            // Public key of B generated.

                BigInteger puB = gVal.modPow(prB,pVal);

            // This is the value that will get sent from B toA.

            System.out.println("Person B's Publickey is:                "+puB+ ".");

            //Calulating A and B's secret keys. BigInteger skA =puB.modPow(prA,pVal);
            BigInteger skB =puA.modPow(prB,pVal);

                System.out.println("Secret Key of A is:"+skA+".");

                System.out.println("Secret Key of B is:"+skB+".");

                        //Taking primitive root of the secret key obtained above.

            System.out.println("Now, enter a number in between2 and Secret Key-1.");

                BigInteger e = new BigInteger(sc.next());

    System.out.println("Enter second Private Key ofA"); BigInteger prAA = newBigInteger(sc.next());

            // Second Public key of A generated.

                BigInteger puAA = gVal.modPow(prAA,skA);

            // This is the value that will get sent from A toB.

            System.out.println("Person A's second Public keyis: "+puAA+".");

            // Second Private key for B from user. System.out.println("Enter second Private Key ofB");
            BigInteger prBB = new BigInteger(sc.next());
```

```
                    //Second Public key of B generated.

                        BigInteger puBB = gVal.modPow(prBB,skA);

                    // This is the value that will get sent from B toA.

                    System.out.println("Person B's second Public keyis: "+puBB+".");

                    //Calulating A and B's final secret keys. BigInteger skAA =
                    puBB.modPow(prAA,skA); BigInteger skBB = puAA.modPow(prBB,skA);

                    System.out.println("Final Secret Key of A is:"+skAA+ ".");

                    System.out.println("Final Secret Key of B is:"+skBB+ ".");

    //converting the final secret key into encryption keywith SHA-256

        MessageDigest digest = MessageDigest.getInstance("SHA-256"); byte[] encodedhash = digest.digest(
    String.valueOf(skAA).getBytes(StandardCharsets.UTF_8));        DataInputStream        in=new
    DataInputStream(System.in);

    //plaintext

        String question ;

        System.out.println("Enter the plaintext:"); question=in.readLine();

                        byte[] plainText = question.getBytes(StandardCharsets

                              .UTF_8);

                    DHEXAES advancedEncryptionStandard = newDHEXAES( encodedhash);

                    byte[] cipherText =advancedEncryptionStandard. encrypt(plainText);

                    byte[] decryptedCipherText = advancedEncryptionStandard.decrypt(cipherText);

    System.out.println("The Encryption Key generated fromthe final secret key is : " + encodedhash);

                    System.out.println("Plaintext entered: " + newString (plainText));

                        System.out.println(new String(cipherText)); System.out.println("Decrypted Text: " + newString(

                              decryptedCipherText));

        }

    public static BigInteger getNextPrime(String ans) {

                        BigInteger test = new BigInteger(ans);

                    while (!test.isProbablePrime(99)) test = test.add(one);

                    return test;

        }


}
```

## 5.2 Explanation and output



```
Enter the value of p:
401
P value is: 401.
Now, enter a number in between 2 and p-1.
3
Enter Private Key of A
4434
Person A's Public key is:  290.
Enter Private Key of B
3434
Person B's Public key is:  111.
Secret Key of A is: 337.
Secret Key of B is: 337.
Now, enter a number in between 2 and Secret Key-1.
23
Enter second Private Key of A
4622
Person A's second Public key is:  328.
Enter second Private Key of B
3818
Person B's second Public key is:  227.
Final Secret Key of A is: 305.
Final Secret Key of B is: 305.
Enter the plain text:
Love in the Time of Cholera
The Encryption Key generated from the final secret key is : [B@504bae78
Plaintext entered: Love in the Time of Cholera
S���3|�0F�rr�:��N|�
�/�:2�^X�
Decrypted Text: Love in the Time of Cholera
```

Figure 4: Output from the Java code above

As per the algorithm mentioned above, we take the required values and perform the operation to find the secret key and then take a primitive root to it and required values and again perform the operation to generate the final secret key. So, it suffices to say that we are just using the Diffie-Hellman algorithm for two times for generating a very strong shared-secret key. This final key is then used in the AES  and as AES needs a 256-bit key we've converted the int obtained from Diffie-Hellman to 256 bit using SHA256. The text to be encrypted is taken from the user and then encrypted and decrypted using the AES algorithm. The output obtained from running the code above is shown below:

## VI. CONCLUSION

So, 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 Diffie-Hellman is prone to several attacks[7] making it not so reliable for performing the secret key exchange. The methodology proposed in this paper has been tested on different lengths of the text and the result obtained is found to be much appealing. The algorithm is tested for various security issues and found to be impenetrable for the above-mentioned attacks. Thus we can use the key generated from this in the symmetric algorithms like AES and make it even more secure.

## References

1. W.Diffie and M. E. Hellman, New directions in cryptography, IEEE Trans. Inform. Theory, Nov. 1976,22: 644-654.
2. Manoj Ranjan Mishra, and Jayaprakash Kar, A STUDY ON DIFFIE-HELLMAN KEY EXCHANGE PROTO- COLS, in International Journal of Pure and Applied Mathematics, 2017, Volume 114 No. 2: 179-189.
3. Diffie–Hellman key exchange-https://www.wikiwand.com/en/DiffieE28093Hellman-key-exchange.
4. Advanced Encryption Standard-https://www.wikiwand.com/en/Advanced-Encryption-Standard.
5. Security issues of the Diffie-Hellman key exchange-https://www.comparitech.com/blog/information- security/diffie-hellman-key-exchange/Security-issues-of-the-Diffie-Hellman-key-exchange.
6. Understanding AES 256 Encryption-https://www.solarwindsmsp.com/blog/aes-256-encryption- algorithm: :text=AES20brings20additional20security20because,harder20to20break20the20encryption.
7. Gowda and Shreyank N 2016 An advanced Diffie-Hellman approach to image steganography Advanced Networks and Telecommunications Systems (ANTS), International Conference on IEEE.

## AUTHOR(S) PROFILE

**K. Nanda Kishore**, is a student currently pursuing bachelors in Information Science and Technology from the School of Engineering and Technology, Jain University. He's interested in building secure and scalable backend platforms and he's into academic research particularly in the field of computer security and cryptography.

**Sujan Chhetri,** is a student currently pursuing bachelors in Information Science and Technology from the School of Engineering and Technology, Jain University. He's highly interested in software development and web technologies mainly the backend and he's into academic research mainly in the field of Network security, Systems and Web Technology.