Key Management Using Combination of Diffie–Hellman Key Exchange with AES Encryption

Yusfrizal Yusfrizal[1], Abdul Meizar[2], Helmi Kurniawan[3], Fhery Agustin[4]
Faculty of Engineering and Computer Science, Universitas Potensi Utama
Jl. K.L. Yos Sudarso Km. 6.5 No. 3A – Medan 20241, Indonesia

***Abstract*- The cryptographic application for data security by using the Diffie-Hellman key exchange and the AES algorithm was to secure data when sending, the first sender of the recipient exchanges the key or generated the key for the encrypted file by using the Diffie-Hellman key exchange, after obtaining the key file would be encrypted by using AES algorithm for encryption and decryption file, sender would send file to receiver party through server first to save data, then recipient to decrypt file to read the file so that could be read back by receiver. In this assignment, the application that combined the Diffie-Hellman Key Exchange with AES encryption algorithm. This application that first used Diffie-Hellman Key Exchange to achieve the goal of exchanging shared secret key. The results indicated that by using Diffie-Hellman with 1024 bit keys the security was equivalent to 3DES 2 key algorithm, if Diffie-Hellman with 2048 bit key then its security equal to 3DES 3 key, if Diffie-Hellman with 3072 bit key then equal security with AES-128 bits, if Diffie-Hellman with 7680 bit keys then the security was equivalent to AES-182 bits, and if Diffie-Hellman with 15360 bit keys then the security was equivalent to AES-256 bits.**

Keywords— data security; key exchange; Diffie Hellman; AES algorithm

I. INTRODUCTION

In the world of cryptography, the key to encrypt and decrypt a secret message is one of the most important elements. The key that determines a chipper text can be read or not. Key secrecy becomes something more crucial and important than the secrecy of the chipper text itself, in terms of messages (chipper text may leak but keys should not leak).

Many ways are done to keep key secrecy. A public key of cryptography technique or algorithm is one developed way of overcoming it. Public key of cryptography requires two keys, public keys that are freely informed and used without confidentiality, as well as private keys that are only used exclusively by one person and never informed to anyone, so that their secrecy is well preserved.

Android key management is the management of cryptographic keys in Android. It deals with several aspects, including key generation, key exchange, key storage, key use, and replacement of keys. Firstly, user has to generate secured personal keys to ensure a high quality start of key management. Then, if two users want to create a shared key between them, they may use secure key exchange to exchange a shared secret key for their communication. Key management also handles key storage that keys must be kept in a safe place. Key usage is how to use key to avoid leaking personal secret keys. Finally, it goes to replacement of keys. User has to replace their key after a period of time.

In this research, Key Management application implements two aspects of Android key management, including key generation and key exchange.

In this assignment, the application that combines the Diffie-Hellman Key Exchange with AES encryption algorithm. Like many of the real world cryptography supported applications, this application that first uses Diffie-Hellman Key Exchange to achieve the goal of exchanging shared secret key. Then it would use the private-key encryption mechanism, AES in this research, to encrypt the rest of the communication.

This application would first use Diffie-Hellman Key Exchange to create a shared secret key between Alice and Bob by using the user entered p, g, Alice's private key and Bob's private key. Then, the application would continue using AES encryption algorithm with the generated shared secret key to encrypt the user's message when user clicks on the "Encrypt" button. At last, the application would also decrypt the cipher text and show the user about the decrypted result.

II. RELATED WORK

Eun-Jun Yoon et al. [1] proposed an efficient Diffie-Hellman-MAC key exchange scheme providing same securities as proposed by Jeong et al. who proposed a strong Diffie-Hellman DSA key exchange scheme providing security against session state reveal attacks as well as forward secrecy and key independence. The proposed scheme is based on the keyed MAC hash function to provide efficiency. Then, they proposed a strong Diffie-Hellman-DSA key exchange scheme providing security against session state reveal attacks as well as forward secrecy and key independence.

Om Pal et al. [2] proposed as there is no provision of entity authentication in Diffie-Hellman key exchange protocol Diffie-Hellman protocol is vulnerable to man-in-middle attack and impersonation attack.

Nanli [3] presented a research paper on Diffie-Hellman key exchange protocol to eliminate the man-in-middle attack. After analyzing the approach suggested by Nanli [3] for eliminating man-in-middle attack in Diffie – Hellman Key exchange protocol, it is found that impersonation attack still exists. Therefore an improved approach is proposed in this paper. By doing two comparisons of hash values in proposed approach, impersonation attack which is present in Nandi's approach is successfully eliminated. With inclusion of authentication mechanism along with two hash comparisons at

the destination side, it eliminates the replay attack, man-in middle attack and impersonation attack successfully.

Vinothini et al. [4] also proposed an efficient Diffie-Hellman-MAC key exchange scheme providing security against session state reveal attacks as well as forward secrecy and key independence.

III. Description of Diffie–Hellman Key Exchange

Stallings [5] explains that public key algorithms are first published in Diffie and Hellman papers that are defined as public key of cryptography and are commonly referred to as Diffie-Hellman Key Exchange or Diffie-Hellman Protocol. The purpose of this algorithm is to enable two users to exchange keys securely, and then it can be used for encryption and decryption of subsequent messages.

The Diffie-Hellman protocol provided the first practical solution to the key distribution problem, allowing two parties, never having met in advance or shared keying material, to establish a shared secret by exchanging messages over an open channel. The key can then be used to encrypt subsequent communications using a symmetric key cipher. The security rests on the intractability of the Diffie-Hellman problem and computing discrete logarithms. We would call the two parties conducting the key exchange "Alice" and "Bob". [6]

The protocol steps: [6]

1. A prime number p and generator α of Z* p ($2 \leq \alpha \leq p - 2$) are selected and published.
2. Alice chooses a random secret x, $1 \leq x \leq p - 2$, and sends Bob α^x mod p.
   A→B: α^x mod p
3. Bob chooses a random secret y, $1 \leq y \leq p - 2$, and sends Alice α^y mod p.
   B→A: α^y mod p
4. Bob receives α^x and computes the shared key as K= $(\alpha^x)^y$ mod p.
5. Alice receives αy and computes the shared key as K= $(\alpha^y)^x$ mod p.

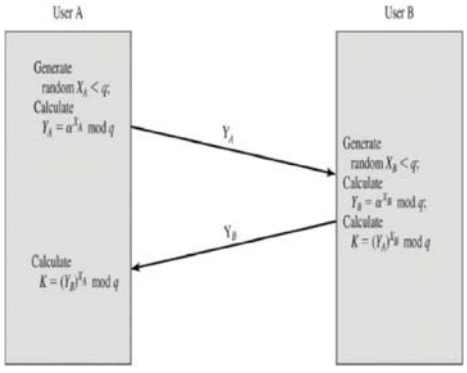

Fig. 1. Diffie Hellman Algorithm [8]

The necessary public values q and α would need to be known ahead of time. Alternatively, user A could pick values for q and α and include those in the first message. The protocol depicted in Figure 1. [7]. Asymmetric key or public key cryptographic algorithm is far more superior compared to the symmetric key cryptography when the security of the confidential data is concerned. Asymmetric key includes large number of cryptographic algorithms. [9]

IV. AES Encryption

Asymmetric key or public key cryptographic algorithm is far more superior compared to the symmetric key cryptography when the security of the confidential data is concerned. Asymmetric key includes large number of cryptographic algorithms. [10]

AES is the new encryption standard recommended by NIST to replace DES in 2001. AES algorithm can support any combination of data (128 bits) and key length of 128, 192, and 256 bits. The algorithm is referred to as AES-128, AES-192, or AES-256, depending on the key length. During encryption decryption process, AES system goes through 10 rounds for I28-bit keys, 12 rounds for I92-bit keys, and 14 rounds for 256-bit keys in order to deliver final cipher-text or to retrieve the original plain-text. [11]

Rijndael Algorithm: The Advanced Encryption Standard comprises three block ciphers, AES-128, AES-192 and AES-256. AES has a fixed block size of 128 bits and a key size of 128, 192, or 256 bits. The block-size has a maximum of 256 bits, but the key-size has no theoretical maximum. The cipher uses number of encryption rounds which converts plain text to cipher text. The output of each round is the input to the next round. The output of the final round is the encrypted plain text known as cipher text. The input given by the user is entered in a matrix known as State Matrix. Following are the four steps. [12]

A. SubBytes Step

Sub Bytes, also known as Byte substitution are the first iterative step of the algorithm in each round. In the Sub Bytes step, each byte in the matrix is reorganized using an 8-bit substitution box. This substitution box is called the Rijndael S-box. This operation provides the non-linearity in the cipher. The S-box used is derived from the multiplicative inverse over GF (2^8), known to have good non-linearity properties. To avoid attacks based on simple algebraic properties, the S-box I constructed by combining the inverse function with an invertible affine transformation. The S-box is also chosen to avoid any fixed points (and so is a derangement), and also any opposite fixed points. [13]

B. Shift Rows Step

The Shift Rows step is performed on the rows of the state matrix. It cyclically shifts the bytes in each row by a certain offset. The first row remains unchanged. Each byte of the second row is shifted one position to the left. Similarly, the third and fourth rows are shifted by two positions and three positions respectively. The shifting pattern for block of size 128 bits and 192 bits is the same. [14]

C. Mix Columns Step

In the Mix Columns step, the four bytes of each column of the state matrix are combined using an invertible linear transformation. Randomly generated polynomials are arranged in a 4*4 matrix. The same polynomial is used during decryption. Each column of the state matrix is XOR-ed with the corresponding column of the polynomial matrix. The result is updated in the same column. The output matrix is the input to Add Round Key. [15]

D. Add RoundKey

A round key is generated by performing various operations on the cipher key. This round key is XOR-ed with each byte of the state matrix. For every round a new round key is generated using some operations on the cipher key. [16]

V. AES Decryption

Decryption involves reversing all the steps taken in encryption using inverse functions like a) Inverse shift rows, b) Inverse substitute bytes, c) Add round key, and d) Inverse mix columns. The third step consists of XO Ring the output of the previous two steps with four words from the key schedule. And the last round for decryption does not involve the "Inverse mix columns" step. [17]

The Cipher Text which is formed of 256-bit 4*8 Matrix is the input for the decryption process. [15]

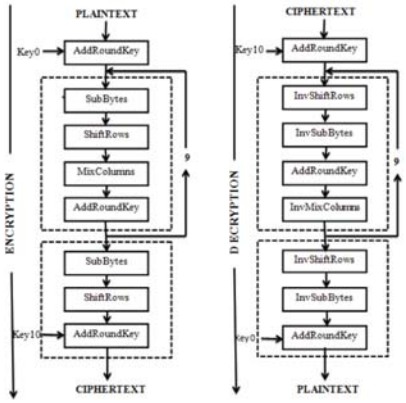

Fig. 2. AES Encryption and Decryption [18]

VI. Man-in-the-Middle Attacks

In this attack, a malicious third party, commonly referred to as "Eve" (for "eavesdropper") retrieves Alice's public key and sends her own public key to Bob. When Bob transmits his public key, Eve interrupts and substitutes the value with her own public key and then sends it to Alice. By now, Alice would have come to an agreement on a common secret key with Eve instead of Bob. This exchange can be done in reverse, and it is possible for Eve to decrypt any messages sent out by Alice or Bob, and then read and possibly modify them before the re-encryption with the appropriate key and transmitting them to the other party.[15]

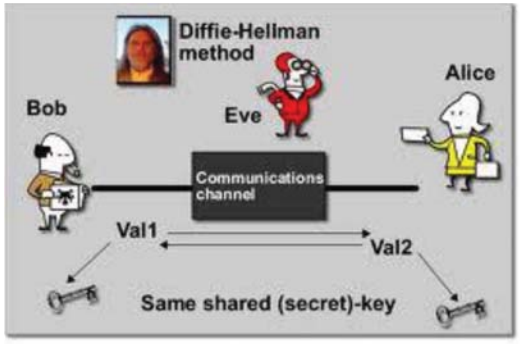

Fig. 3. Man-In-The-Middle Attack on A D-H Key Exchange [15]

To address this problem, generally a process of authentication would be needed to ensure that, whenever Alice wishes to send a message to Bob, the recipient can only be Bob and not an Eve, and vice versa. It is also important and generally the normal to discard the keys after usage, so that there would be no long-term keys that can be disclosed to cause problems in the future [16]. Other concerns typically revolve around optimizing the mathematics involved. That is, to properly generate the randomly selected values so that they are (1) large enough to achieve computational infeasibility for attackers, and (2) random enough, as pseudorandom numbers can greatly ease Eve due to their eventual predictability.

VII. Analysis and Result

Man-in-the-middle attack is done by keeping the public keys of sender and receiver by the attacker and sending fake public keys to them respectively. Here, the enhanced Diffie-Hellman algorithm secures the key as it's generating the second shared secret key and attacker is unaware of the idea of taking the primitive root of the first secret key. The known-plaintext attack uses plaintext and cipher text for retrieving the secret keys. So, this attack is very much possible on Diffie-Hellman algorithm. So, Known-plaintext attack is one of the most probable attacks in the original Diffie-Hellman algorithm. But in the upgraded Diffie-Hellman algorithm, the shared secret key is being generated for the second time i.e. second shared-secret key and attacker here is completely unaware of the idea, we are using of taking 'e' as the primitive root of first shared-secret key 'S' and then multiplying respective random number to the second shared-secret key and then exchanging it for generating key each time for every message or even for the same message. So, this would make our system very much secured. We are basically using the Diffie-Hellman algorithm for two times for generating 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 even.be very strong. [19]

The execution time of the upgraded algorithm is much greater than the original Diffie-Hellman algorithm. The difference between the two runtimes is very small. So, it is possible to introduce second-shared secret key and random parameters in the original Diffie-Hellman algorithm.

There are 6 main processes where 2 processes are the application of Diffie Hellman's Generation and Distribution algorithm. The rest of the process is Storage, usage, change and Destruction.

The steps are:

1. Generation, this process is an implementation of Diffie Hellman's algorithm. The generated variables are: variables g and n, $X = g^x \bmod n$, $Y = g^y \bmod n$, $K = Y^x \bmod n$ and $K' = X^y \bmod n$.
2. Distribution, this process is part of the implementation of the Diffie Hellman algorithm. Variables that are distributed are variables X and Y.
3. Storage, This process is the stage where after the private key is obtained. The private key is then stored in the server database and secured.
4. Usage, this process is the key usage stage taken from the database for the process of encryption and decryption.
5. Change, this process is a process to perform a key replacement if the client finds it necessary to replace the key by doing the Generation process again.
6. Destruction, this process runs when the client has changed. The keys that are in the database are deleted.
7.

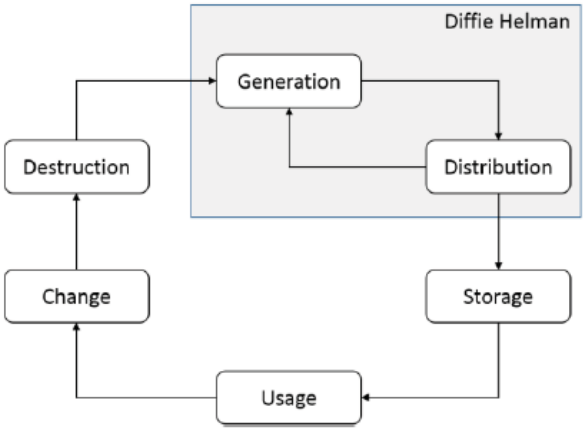

Fig. 4. Key Management System [20]

The stages contained in the System overview are divided into 3 main processes, namely. The stages of each process are as follows:

a. Generate Key Process:
- The server generates the numbers g and n.
- Client generates the number x while taking the g and n numbers of the server. Next compute $X = g^x \bmod n$. The resulting X value is sent to the server.
- Client (receiver) generates the y number while taking the numbers g and n from the server. Next calculate $Y = g^y \bmod n$. The resulting Y value is sent to the server.
- The Client takes the Y value sent by the receiving client, and calculates $K = Y^x \bmod n$ and stores K in the database as the private key to encrypt.
- The Client takes the X value sent by the sending client, and calculates $K' = X^y \bmod n$ and stores K' in the database as the private key to decrypt.

b. Encryption Process:
- Client specifies the client address (Receiver) (IP / Hostname).
- The Client selects the file on the computer and retrieves the private key.
- The Client encrypts the selected file with the private key.
- The Client sends the file to the server.

c. Decryption Process:
- The Client selects files on the server.
- Client takes the private key.
- The Client decrypts the selected file with the private key.

VIII. Implementation

Android key management is the management of cryptographic keys in Android. It deals with several aspects, including key generation, key exchange, key storage, key use, and replacement of keys. Firstly, user has to generate secured personal keys to ensure a high quality start of key management. Then, if two users want to create a shared key between them, they may use secure key exchange to exchange a shared secret key for their communication. Key management also handles key storage that keys must be kept in a safe place. Key usage is how to use key to avoid leaking personal secret keys. Finally, it goes to replacement of keys. User has to replace their key after a period of time. This application implements two aspects of Android key management, including key generation and key exchange.

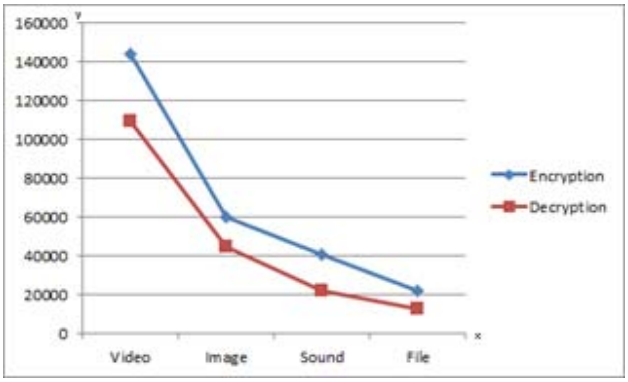

Fig. 5. Testing of encryption and decryption time

In this assignment, the application that combines the Diffie-Hellman Key Exchange with AES encryption algorithm. Like many of the real world cryptography supported applications, we would build an application that first uses Diffie-Hellman Key Exchange to achieve the goal of exchanging shared secret

key. Then we would use the private-key encryption mechanism, AES in this lab, to encrypt the rest of the communication.

The application would first use Diffie-Hellman Key Exchange to create a shared secret key between Alice and Bob by using the user entered p, g, Alice's private key and Bob's private key. Then, the application would continue using AES encryption algorithm with the generated shared secret key to encrypt the user's message when user clicks on the "Encrypt" button. At last, the application would also decrypt the cipher text and show the user about the decrypted result.

For the testing time of encryption and decryption process got result that size of file tested have influence in long time of encryption process and length of time process of decryption. For video.avi has the longest encryption time of 144,067 ms and also the longest decryption time of 109,700 ms. While file.doc has the fastest encryption time of 22,200 ms and the fastest decryption time is 12,600 ms.

In testing these resources is done by testing the usage power by using Heap Size and Used Heap. Heap Size is the application of some memory for Java Virtual Machine applications. Used Heap is the amount of memory actually used by the application process. From the test results obtained the lowest HS encryption value is file.docx and sound.mp3 with 162,500 MB and the lowest HS decryption value is file.docx with 59,500 MB. For the lowest UH encryption value is video.avi with 25,167 MB and the lowest UH decryption value is file.docx with 12.574MB. The comparison of HS values and UH values in the test shows the level of power consumption when the application is run. The more the value of both HS and UH low then the power used for processing is getting smaller.

TABLE I.
SYSTEM SECURITY TESTING

No	Kind of File	Number of Different Bits	Avalanche Effect %
1	File.docx	405665	50.0199%
2	Video.avi	4117510	50.0040%
3	Sound.mp3	741683	49.9819%
4	Image.jpg	1306524	49.9448%
Average of Avalanche Effect			49.9877%

In system security testing or Avalanche Effect obtained result that the highest AE value obtained in file.docx with value 50.0199% and the lowest AE value obtained in image.jpg with value 49.9448%. The average AE total is 49.9877%, quite good as it is close to half of the hundred percent of the total bits tested.

TABLE II.
RESOURCES TESTING HEAP SIZE

No	Kind of File	HS Encryption	HS Decryption
1	File.docx	162.500 MB	59.500 MB
2	Video.avi	263.950 MB	182.417 MB
3	Sound.mp3	162.500 MB	74.483 MB
4	Image.jpg	232.033 MB	103.933 MB
Average Resouces		205.246 MB	105.083 MB

TABLE III.
RESOURCES TESTING USED SIZE

No	Kind of File	U Encryption	UH Decryption
1	File.docx	25.167 MB	12.574 MB
2	Video.avi	25.167 MB	115.740 MB
3	Sound.mp3	56.709 MB	23.422 MB
4	Image.jpg	65.133 MB	41.593 MB
Average Resouces		43.044 MB	48.332 MB

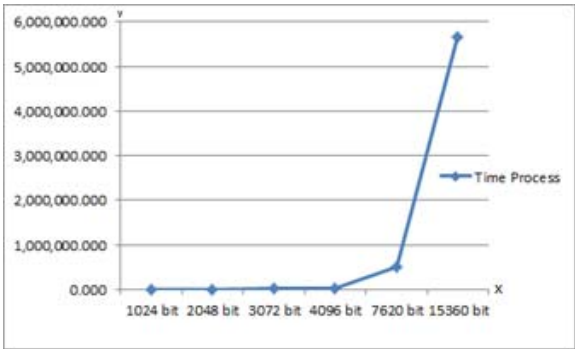

Fig. 6. Diffie-Hellmann Time Process

In the Diffie-Hellmann process time test, the average total processing time of 497,567 ms for 1024 bits, 3175,567 ms for 2048 bits, 12343,767 ms for 3072 bits, and 32447.267 ms for 4096 bits and 515274.8 ms for 7620 bits is estimated. The above results indicate that by using Diffie-Hellman with 1024 bit keys the security is equivalent to 3DES 2 key algorithm, if Diffie-Hellman with 2048 bit key then its security equal to 3DES 3 key, if Diffie-Hellman with 3072 bit key then equal security with AES-128 bits, if Diffie-Hellman with 7680 bit keys then the security is equivalent to AES-182 bits, and if Diffie-Hellman with 15360 bit keys then the security is equivalent to AES-256 bits.

IX. CONCLUSION

This paper represents the security methods, which secure the data of malicious users at the cloud is completely avoided by Diffie Hellman key exchange algorithm. This paper also addresses the problems of the access control using proper authentication mechanism by two factors. D-H protocol fits better in this scenario as number of users on cloud is very large and key management is very difficult. Our proposed scheme eliminates the overheads of key computation and their management. Provision of security to the users data on the cloud would defiantly empowers the Data owner to outsource the data to save.

REFERENCES

[1] Eun-Jun Yoon and Kee-Young Yoo, “An Efficient Diffie- Hellman-MAC Key Exchange Scheme”, Fourth International Conference on Innovative Computing, Information and Control, 2012.

[2] Om Pal, et al, “Diffie-Hellman Key Exchange Protocol with Entities Authentication”, International Journal Of Engineering And Computer Science, Volume 6 Issue 4, 2017

[3] Nan Li, “Research on Diffie – Hellman Key, Exchange Protocol”, IEEE 2nd International Conference, on Computer Engineering and Technology, Vol. No 4, pp 634 – 637, 2010.
[4] Vinothini, Saranya, Vasumathi, “A Study On Diffie-Hellman Algorithm in Network Security”, International Journal Of Engineering And Computer Science, Volume 3 Issue 7 July, 2014 Page No. 7346-7349.
[5] Navpreet Kaur et al, “Authenticated Diffie-Hellman Key Exchange Algorithm”, International Journal of Computer Science and Information Technologies, Vol. 5 (4) , 2014, 5404-5407
[6] Stallings, W., “Cryptography and Network Security Principles and Practices Fourth Edition”, Prentice Hall, New Jersey, 2005.
[7] Dripto Chatterjee, Joyshree Nath, Sankar Das, Shalabh Agarwal, Asoke Nath, “Symmetric Key Cryptography Using Modified DJSSA Symmetric Key Algorithm”, Proceedings of International conference Worldcomp 2011 held at Las Vegas 18-21 July 2011, Page-306-311, Vol-1(2011).
[8] Khan, Shakeeba S., and R. R. Tuteja. "Security in cloud computing using cryptographic algorithms." International Journal of Innovative Research in Computer and Communication Engineering 3.1 (2015): 148-155.
[9] Ayan Roy, “Brief Comparison Of RSA and Diffie-Hellman (Public Key) Algorithm”, Transactions on Information Security, Vol 1, 2016.
[10] Mr. Gurjeevan Singh, Mr. Ashwani Singla and Mr. K S Sandha, "Cryptography Algorithm Comparison for Security Enhancement in Wireless Intrusion Detection System", International Journal of Multidisciplinary Research, Vol.1 Issue 4, pp. 143-151, August 2011
[11] V. Sumathy & C. Navaneethan, “Enhanced AES Algorithm For Strong Encryption”, International Journal of Advances in Engineering & Technology, Sept 2012, Vol. 4, Issue 2, pp. 547-553
[12] Dr. (Mrs) Pushpa R. Suri, “A Cipher based on 3D Array Block Rotation”, International Journal of Computer Science and Network Security, VOL.10 No.2, February 2010, pp. 186-191.
[13] Das Debasis, Misra Rajiv. “Programmable Cellular Automata Based Efficient Parallel AES Encryption Algorithm”. International Journal of Network Security & Its Applications (IJNSA), Vol.3, No.6, November 2011, pp. 204.
[14] Dr. Prerna Mahajan & Abhishek Sachdeva, “A Study of Encryption Algorithms AES, DES and RSA for Security”, Global Journal of Computer Science and Technology Network, Web & Security, Volume 13 Issue 15 Version 1.0, 2013.
[15] Priyanka Pimpale, Rohan Rayarikar and Sanket Upadhyay, “Modifications to AES Algorithm for Complex Encryption”, IJCSNS International Journal of Computer Science and Network Security, VOL.11 No.10, October 2011
[16] Maryam Ahmed, et al, “Diffie-Hellman and Its Application in Security Protocols”, International Journal of Engineering Science and Innovative Technology (IJESIT) Volume 1, Issue 2, November 2012
[17] EMC Corporation (2012) “What Is Diffie-Hellman?”, RSA Laboratories [Online] Available at http://www.rsa.com/rsalabs/node.asp?id=2248 Accessed 20 December 2017]
[18] Aleisa, Noura. "A Comparison of the 3DES and AES Encryption Standards." International Journal of Security and Its Applications 9.7 (2015): 241-246.
[19] Conti, Mauro, Nicola Dragoni, and Viktor Lesyk. "A survey of man in the middle attacks." IEEE Communications Surveys & Tutorials 18.3 (2016): 2027-2051.
[20] Sumimol, L., and A. Janisha. "Security in Wireless Adhoc Networks Based on Trust and Encryption." International Journal of Advanced Research in Computer and Communication Engineering 4.6 (2015).