International Journal of
Advances in Scientific Research and Engineering (ijasre)

E-ISSN : 2454-8006

DOI: 10.31695/IJASRE.2019.33560

Volume 5, Issue 10
October - 2019

Prevention of Man-In-The-Middle Attack in Diffie-Hellman Key Exchange Algorithm using Proposed Hash Function

Phyu Phyu Thwe[1] and May Htet[2]

Research Scholar [1] and Associate Professor[2]

[1-2]Department of Computer Engineering and Information Technology,

[1-2]Mandalay Technological University, Mandalay

Myanmar

Abstract

The key exchange process is a very important feature for many parts in cryptography. In data encryption, symmetric encryption techniques need to share the same secret key securely between two parties before encryption and this is a challenging task for secure data transmission. Most symmetric key encryptions and key management systems widely use Diffie-Hellman Key Exchange (DHKE) algorithm for the purpose of key distribution because it has simple computation and supports forward security. However, there is no key authentication and Man-In-The-Middle (MITM) attack has occurred during the key generation process. To overcome this problem, a new hash function is proposed to get the public key integrity during the public key sharing process of DHKE algorithm. This hash function is created by using six bitwise operators and operated in a variable length of the rounds depending on message length. Thus, the proposed system improves the security of DHKE and grantees the user authentication requirements.

Key Words: *Bitwise Operation, Diffie-Hellman Key Exchange, Hash Function, Key Integrity, Man-In-The-Middle Attack.*

1. INTRODUCTION

Nowadays, daily activities are accomplished through internet connectivity and communication technology. A secure communication channel is required to protect transmitted data and information from adversaries. All of the protection systems afford to get the fundamental security objectives such as confidentiality, integrity and availability for both data and information. Cryptography can fulfil these security requirements and provide trusted data transmission. In the enterprise networks, the most commonly used cryptographic protocols such as Secure Sockets Layer (SSL)/Transport Layer Security (TLS), Interner Protocol Security (IPSec), Secure Shell (SSH) are used for encryption of the communication channels to prevent compromising data. Web servers use cipher suites to secure data over Hyper Text Transfer Protocols (HTTPs) and web traffics. Cipher suites are a combination of various cryptographic algorithms such as key exchange algorithms, encryption algorithms and hash functions to give security for HTTPs traffic. DHKE algorithm is widely used to secure various internet services [1]. Besides, modern cryptography generally contains symmetric encryption techniques and asymmetric encryption techniques. Although the symmetric encryption technique uses the single same key for both encryption and decryption processes, asymmetric encryption techniques use two keys in which public key is used for encryption process and private key related with the public key is used for the decryption process. Therefore, key exchange process between two parties becomes the main difficult problem for encryption techniques, especially symmetric encryption. Among key exchange mechanisms, DHKE algorithm is commonly used in many modern cryptographic applications since this algorithm does not take much key pair generation time and not only is easy to implement but also support strong security. However, DHKE cannot support authentication of two communicating parties and suffers from a MITM attack. This is an important reason not being enough avoidance from compromise by an attacker. To overcome this venerability, a method to support authentication for DHKE algorithm is needed.

In this research, it is intended to use DHKE approach for secure key distribution. The main idea of the system is to modify DHKE algorithm to support secure key exchange feature. In the proposed system, the proposed hash function is attached to the original DHKE algorithm for preventing the MITM attack to generate the session key more securely. The hash code generated from the proposed hash function is also evaluated with the randomness test of the National Institute of Standards and Technology (NIST).

The rest of the paper is systematized as follows: related work is described in Section 2. The design of the proposed system is presented in Section 3. Section 4 explains the research methods. The experimental results of this system are shown in Section 5. Finally, discussion and conclusion are presented in Section 6.

2. RELATED WORKS

In 2010, Nan, L. proposed key exchange scheme based on the existing hash function to enhance the security of DHKE protocol. The author proposed a method to verify the communicating parties to each other needed to prevent MITM attack. The author uses an authentication server to store the identifiers of users and generate a one-time random number. The one-time random number is the secret information only between user A and user B and difficult for others to know. Therefore it gives the identity authentication of DHKE public keys on both sides with the help of hash function. Various cryptographic algorithms were tested to compare the computational efficiency and recommended to use Secure Hash Algorithm 1 (SHA-1) as a hash function in DHKE protocol. The improved DHKE protocol can resist replay attack, impersonation attack and MITM attack. The simulation results are tested in the Local Area Network (LAN). According to the results, authentication using the hash function can reduce the computational steps and is faster than the other public key and symmetric key encryption algorithm in execution time. The improved method for DHKE was not possible to prove the identification of users without Authentication Server (AS) due to maintaining the record of user identities in communication from AS [2].

Then, Preetika, J., Manju, V. and Pushpendra, R. V. proposed a node to node authentication protocol for wireless sensor network. In their proposed method, the concept of DHKE protocol is used to solve the key distribution problem among senor nodes. Then, the weakness of DHKE protocol such as MITM attack is resolved by combining cryptography and cluster head. A MITM attack is eliminated because the key in transmission cannot be used without the user's private key confined only to the valid user. For encryption of common numbers pre-assigned to each node manually before deployment, Data Encryption Algorithm (DEA) was used. By eliminating the man-in-middle attack, the network attacks of eavesdropping or captured nodes compromise are reduced significantly. But, the computational overhead is greater due to encryption and decryption processes of pre-assign numbers. [3].

Aqeel, S. K. and David, L. proposed improvements which can help to prevent the MITM attack. Firstly, the authors studied the problem of MITM attack in DHKE protocol and considered some of the shortcomings of current defences. The proposed method involved the use of Geffe generation to generate a longer random sequence from the password. To give user authentication requirement, the private key was generated from usernames and passwords that used to connect two authorized users with their systems. The proposed method ensured that the private keys will not be sent through the channels, and will be saved as hashes in the server. As a result, this method provided non-repudiation property and prevents MITM attack by identifying the sender and the receiver from their user information. However, the method will have expensive cost and complexity because an authentication server is required to maintain securely usernames and passwords and to encrypt and hash all of the data on the communication channel between server and users [4].

Aryan, C. K. and Durai, R. V. P. M. considered reducing the possibility of attacks on DHKE algorithm. Many web services use DHKE for reliable communication and for securing purpose. MITM attack, plaintext attack, logjam attack, etc were found on the DHKE protocol which is used in TLS. Therefore, the authors restructured the DHKE algorithm to get a stronger secret key by calculating two times of original computation with a new private key. The second secret key was generated by taking the primitive root of the first secret key and exchanged by multiplying with a random parameter. Although the upgraded algorithm reduced the probability of known-plaintext attack, it is still susceptible to a MITM attack. Also, the execution time of the upgraded algorithm is much greater than the original algorithm because of generating double session keys [5].

According to the literature and concepts of the enhancements which are pointed out from the previous researches, the main objective of this work is to propose authenticated DHKE algorithm with the help of proposed hash function to support public key integrity with acceptable execution time.

3. DESIGN OF THE PROPOSED SYSTEM

Although DHKE algorithm is a useful key exchange algorithm in symmetric key cryptography, the main problem is that DHKE suffers from MITM attack during public key sending process between sender and receiver. In a MITM attack, a third party who does not both sender and receiver negotiates the communication channel and public keys are received and altered. Then, the altered public keys are resent to both real users and shared secret key same as both real users can be easily calculated. The proposed work modifies DHKE algorithm to prevent MITM attack by supporting authenticated key exchange process between sender and receiver. This research also proposes a new hash function using various bitwise operators to provide the integrity of the public key in DHKE algorithm.

In the proposed system design, proposed Variable Round Hash (VRH) function is added to verify whether the public key has not been altered by an attacker. In the system, a private key is chosen and the public key is calculated by using prime *(p)*, generator *(g)* and private key like the same process of original DHKE algorithm. Before exchanging the public key to each other, the hash codes of public keys in both sites are generated with the help of proposed VRH function and interchanged with each other. In the verification process, the hash code is produced from receiving public key again in both sites. If both receiving hash code and calculating hash code is matched, it is ensured that the key has not been changed. Therefore, the secret key is calculated using this public key. Otherwise, the receiving public key is rejected and the key exchange process is repeated. The design of the proposed system is shown in Figure 1.

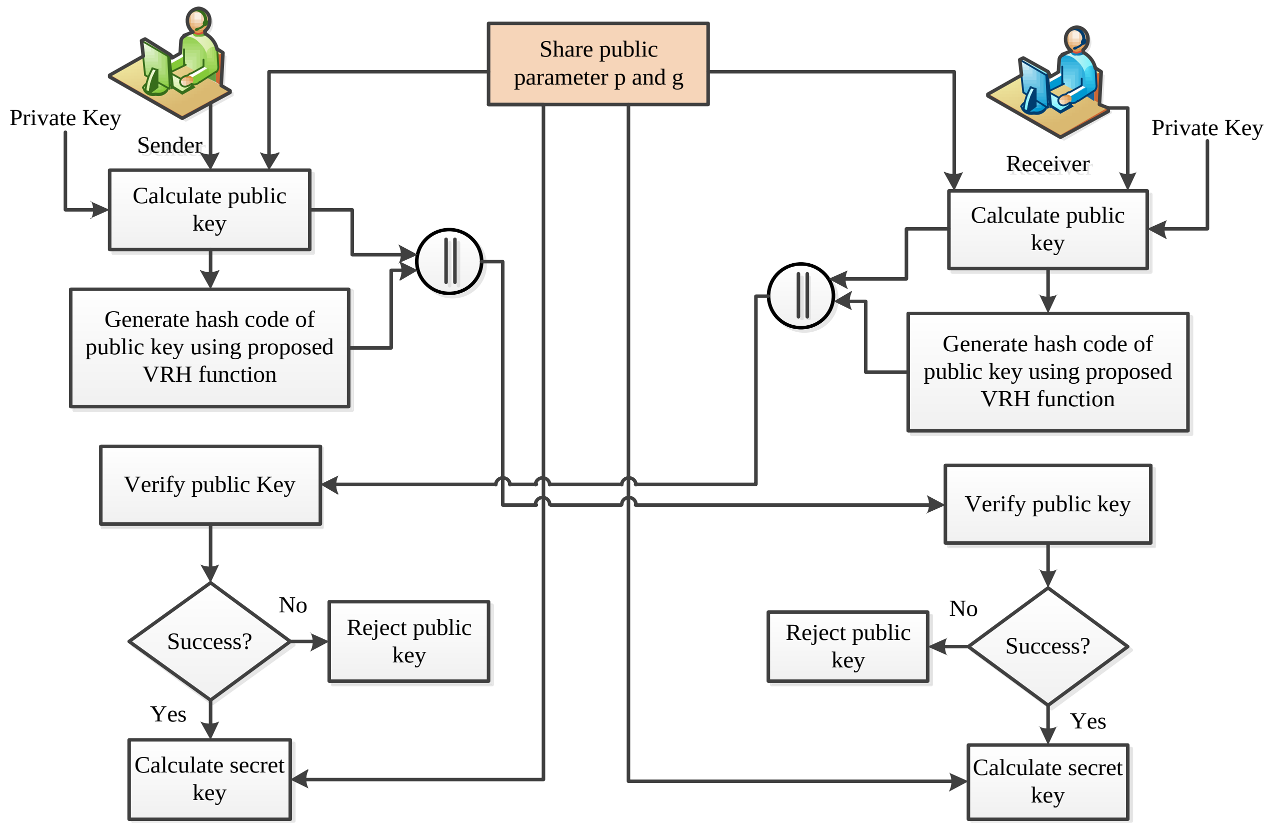

Figure1. Design of the proposed system

In the modified DHKE algorithm, as time complexity of proposed VRH function is O(n), the modified algorithm has O(n)+O(n). Therefore, the time complexity of the modified DHKE algorithm is competitive equivalent to that of the original DHKE algorithm.

4. RESEARCH METHODS

The proposed system combined original DHKE algorithm and proposed hash function namely Variable Round Hash (VRH) function for generating hash code. VRH function is mostly based on the bitwise operations to be simple without exceeding time complexity of original DHKE algorithm.

4.1 Diffie-Hellman Key Exchange Algorithm

DHKE algorithm is the first public key algorithm proposed by Whitfield Diffie and Martin Hellman in 1976. The DHKE algorithm aims to securely exchange a key over a public channel between both users (user A and user B). This algorithm is based on the difficulty of computing logarithms over a finite field (Galois Field) GF(q) with a prime number (p) of elements. There is a primitive root of prime (p) called generator (g).

User A must choose a small number (a), as his private key. Then his public key (Y) can be calculated by using equation (1).

$$Y = g^{a} \bmod p, \quad \text{for } 1 \leq a \leq p-1, \qquad \text{--------------------- (1)}$$

On the other hand, User B must also choose her private key (b) and then calculate his public key (X) by using equation (2).

$$X = g^b \bmod p, \text{ for } 1 \leq b \leq p-1, \qquad \text{-------------------- (2)}$$

Although the public key (Y) and (X) can be easily calculated from private keys *(a)* and *(b)* respectively, computing the values of (a) and (b) from the public keys (Y) and (X) is much more difficult. After calculating public keys, they are interchanged between the two parties and the secret key is calculated independently using received public keys by equation (3) [6].

$$Key = Y^b \bmod p = X^a \bmod p \qquad \text{-------------------- (3)}$$

The key from equation (3) is the same secret key which can be used securely in symmetric key encryption systems for both sites. The process of the DHKE algorithm is demonstrated in Figure 2.

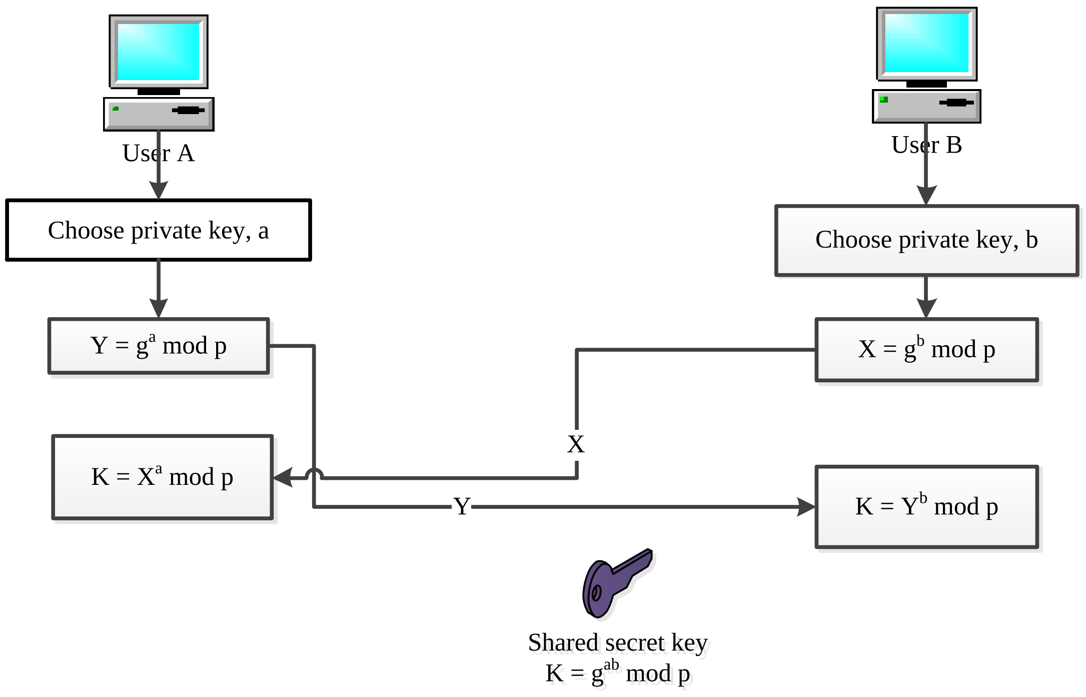

Figure2. Process of DHKE Algorithm

Time complexity of DHKE algorithm can be O(n) because calculating both public keys and the shared key depends on the bit length of private key (n).

4.2 Proposed Variable Round Hash Function

The proposed Variable Round Hashing (VRH) function uses four 32-bit variables and bitwise functions such as AND, OR, XOR, NOT, Left-shift, Right-shift for all hash function operations. The proposed VRH function can protect the invalid public key modification from a third party and provide the public key integrity as shown in Figure 3.

DOI: 10.31695/IJASRE.2019.33560

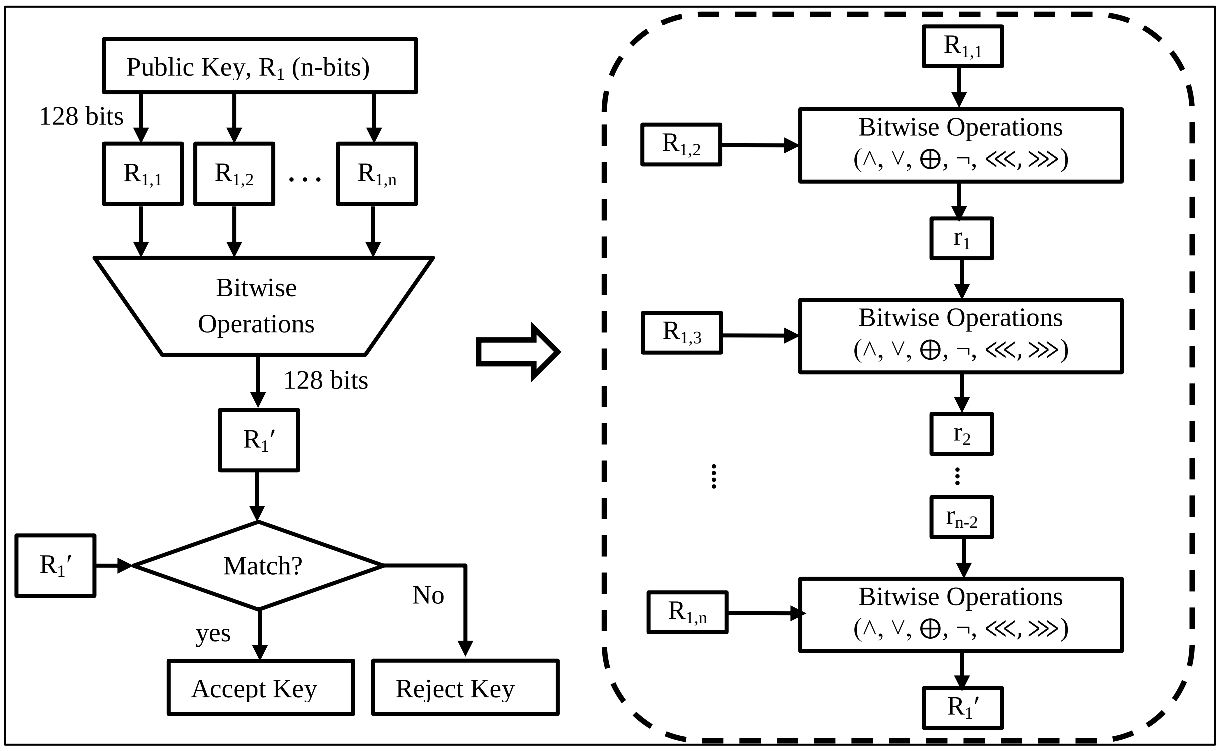
Public Key, R_1 (n-bits)
128 bits
$R_{1,1}$
$R_{1,2}$
...
$R_{1,n}$
Bitwise Operations
128 bits
R_1'
R_1'
Match?
No
yes
Accept Key
Reject Key
$R_{1,1}$
$R_{1,2}$
Bitwise Operations ($\wedge, \vee, \oplus, \neg, \lll, \ggg$)
r_1
$R_{1,3}$
Bitwise Operations ($\wedge, \vee, \oplus, \neg, \lll, \ggg$)
r_2
r_{n-2}
$R_{1,n}$
Bitwise Operations ($\wedge, \vee, \oplus, \neg, \lll, \ggg$)
R_1'

Figure3. Process of VRH Function for Public Key Integrity and User Authentication

The proposed VRH function generates 128-bit fixed-length output. The proposed VRH consists of three parts:

Padding message: The message length is padded to ensure that the length is multiples of four because of using four 32-bit working registers: A, B, C, D. The message is appended with zero bits followed by the message to get the multiple of four.

Initialization: The padding message is scheduled as 32 bits word W0, W1, W2, ..., Wj to process for each of the variable rounds. The words Wj are split into 16 words for four 32-bit variables to operate ith hash functions. Then four working registers are initialized with the first 16 words of Wj.

Compression function: It consists of four hash functions for 32-bit registers: Mix Function 1 for register A, four block functions for register B, Mix Function 2 for register C and Mix Function 3 for register D.

In the proposed hash function, the variable number of rounds is used depending on the length of the message. The example of one round hash function is displayed in Figure 4 and all rounds have the same structure.

DOI: 10.31695/IJASRE.2019.33560

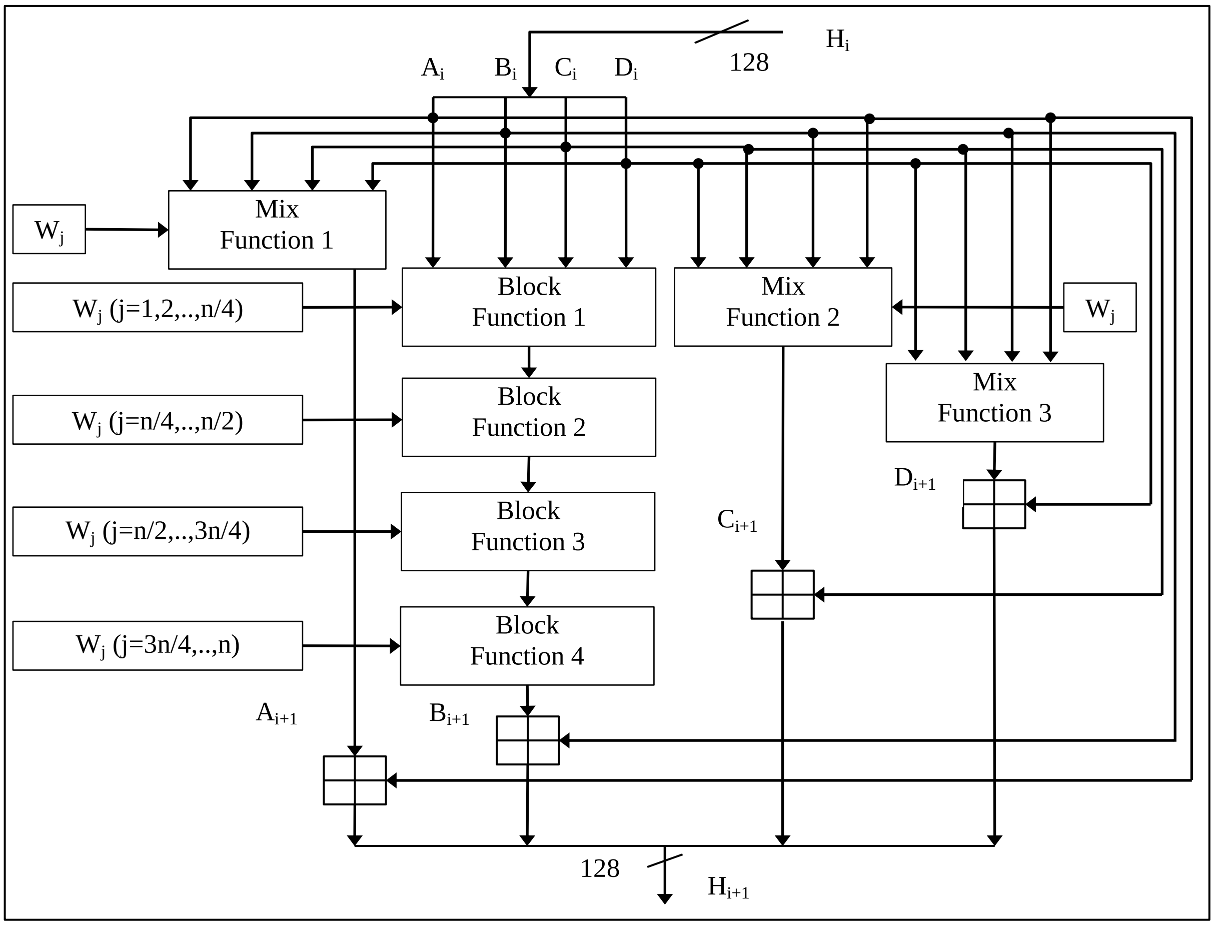

Figure 4. One Round function of the proposed VRH algorithm

In Figure 4, each 32-bit word of the message (x) is computed with four working registers in four different hash functions. The initial value of registers is replaced by a new hash value after each round. The final hash value, H_n is equal to the output h(x) of the proposed hash function.

Different internal structures of Mix Functions are built to be one-way function and provide a good avalanche effect. Mix Functions (MF) within round j are defined as mentioned in Figure 5.

$$MF_1\,(A, B, C, D, w_j) = (A + (w_j)_{<<<20}) \oplus (B + (w_{j+1} \wedge w_{n-(j+1)})) \oplus (C + ((w_{j+2})_{<<16} \vee (w_{j+3})_{<<16})) + D$$

$$MF_2\,(A, B, C, D, w_j) = (A_{<<<16} \vee \sim w_j) + (B \vee \sim w_{j+1}) + (C \oplus \sim w_{j+2}) + (D \wedge \sim w_{j+3})$$

$$MF_3\,(A, B, C, D, w_j) = (\neg A \vee B) \oplus (\neg\, C \wedge (D_{<<<16}))$$

Figure 5. Mix Functions for each round

In Block functions, padded message is divided into four blocks. Each block processes on different internal structure depending on wj as illustrated in Figure 6 and works for four rounds in every j rounds.

$$\mathrm{BF}(A,B,C,D,w_j) = \begin{cases} D + \left(\neg w_{j+k} \vee w_{\left(\frac{n}{4}\right)-(k+j-2)}\right), & 0 \le j < n/4 \\ C + \left(\neg w_{j+k} \wedge w_{(3n/4)-(j+k+1)}\right), & n/4 \le j < n/2 \\ B + \left(\neg w_{j+k} \oplus w_{\left(\frac{5n}{4}\right)-(k+j+1)}\right), & n/2 \le j < 3n/4 \\ A + \left(\neg w_{j+k} \oplus w_{\left(\frac{7n}{4}\right)-(k+j+1)}\right), & 3n/4 \le j < n \end{cases}$$

Figure 6. Internal structures for four block functions

In block functions and mix functions, the words wj are processed together with four registers (Ai, Bi, Ci, Di). Then, the new register values (Ai+1, Bi+1, Ci+1, Di+1) and old register values are added and modulo 2^{32} operation is performed. The output of mod 2^{32} operation Hi+1 is used as the input for the next round. The final output after variable length rounds is the hash value of message.

In the proposed VRH function, padding message and initialization steps have constant time individually. Compression function requires time complexity O(yn), where n is the number of blocks required to fit input message and y is the overhead per input block. Accordingly, time Complexity of Hash function is C1 + C2 + O(yn). Since C1+C2 is the constant value and y is constant time, time Complexity of Hash function is estimated as O(n) by neglecting C1+C2. In view of that, the proposed VRH function has the acceptable time complexity to supplement into other algorithms.

5. EXPERIMENTAL RESULTS

In this section, the performance of proposed algorithms will be evaluated in terms of security strength and execution time. In this research, the security of proposed key exchange algorithm with hash function is tested with MITM attack and the performance of proposed hash function is evaluated in comparison with some existing hash functions in terms of execution time, Avalanche effect, Entropy, Frequency and Collision resistance. The proposed system is developed by java 1.8 on eclipse software.

5.1 Security Strength

The security strength of proposed hash function was measured based on permutation formulation, avalanche effect, entropy test, frequency (monobit) test and collision resistance.

Permutation formulation: If the attackers know the exact bitwise operations, it is possible to manipulate the hash code. Otherwise, the attackers must try 1230 bitwise operations according to permutation formula to generate the correct hash code with the help of equation (4).

$$p_k^n = \frac{n!}{(n-k)!} \qquad \text{---------------------- (4)}$$

Where, n = number of bitwise operators and k = number of used bitwise operators [7]. Example calculations for using six bitwise operators to calculate hash code are as following:

If two bitwise operators are used in hash function, $P_2^6 = \frac{6!}{(6-2)!} = 30$ operations must be tried to correct hash code

If three bitwise operators are used in hash function, $P_3^6 = \frac{6!}{(6-3)!} = 120$ operations must be tried.

If four bitwise operators are used in hash function, $P_4^6 = \frac{6!}{(6-4)!} = 360$ operations must be tried.

If five/six bitwise operators are used in hash function, $P_5^6 = \frac{6!}{(6-5)!} = 720$ operations must be tried.

Avalanche effect: A good avalanche effect is achieved when a small change in the input results in a significant change (nearly 50%) of the encrypted text. The avalanche effect can be calculated by using equation (5) [8].

$$\text{Avalanche effect} = \frac{\text{Number of fliped bits}}{\text{Total number of bits}} \times 100 \qquad \text{---------------------- (5)}$$

, where the number of flipped bit is equal to the number of bit changed in binary values of original message and modified message. Procedure to calculate avalanche effect of a hash function is shown in Figure7.

Procedure:

1) Get initial hash value
2) Get modified hash value
3) Compare the two hash values
    - count the number of flipped bits (out of total bits)
    - count the length of longest identical bit sequence
    - count the length of longest flipped bit sequence
4) Calculate the avalanche effect

Figure7. Procedure to calculate avalanche effect

Comparison results of avalanche effects which analyzed on various message sizes for the proposed VRH and some existing hash functions as mentioned in Table1.

Table1. Avalanche effects for hash functions on various message sizes

Algorithm	**Message sizes**				
	1 KB	**1.5 KB**	**2 KB**	**2.5 KB**	**3 KB**
SHA-1	51.56	49.64	48.39	47.71	51.51
SHA-256	50.38	49.75	49.77	50.03	49.12
MD5	50.91	50	49.87	51.43	46.42
Proposed VRH	50.39	50.97	51.37	51.63	51.24

As revealed in Table1, the avalanche effect of proposed hash function is slightly smaller than MD5 for 1KB and SHA-1 for 1KB and 3KB depending on the position and binary value of character in text message. For all test messages, avalanche effect of proposed VRH function is greater than 50% and it is acceptable level for hash function which can detect even small modification of text message. The comparison results of avalanche effect during running time on 1KB message size for the proposed hash function and some existing hash functions such as SHA-1, SHA-256 and MD5 is shown in Figure7.

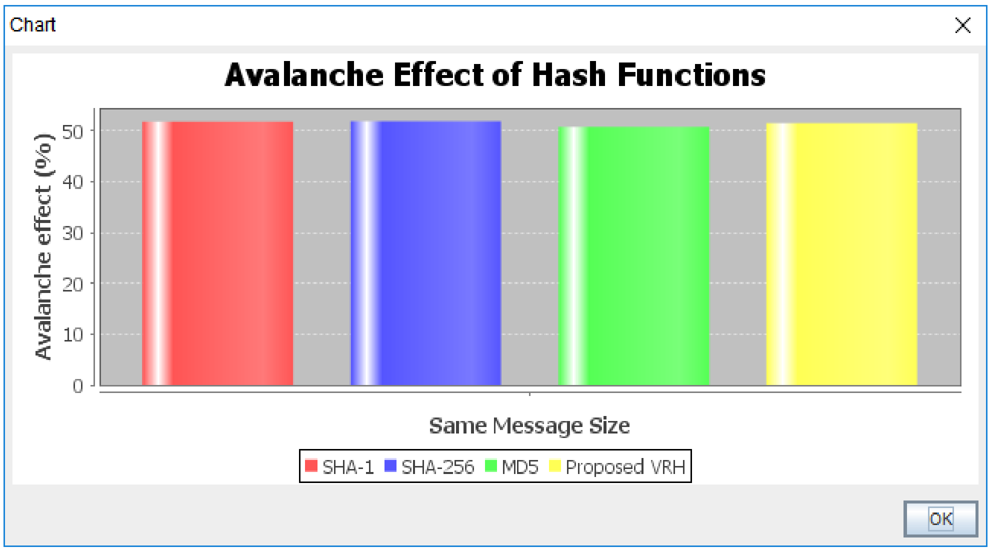

Figure8. Comparison result of avalanche effect

Entropy test: Entropy is a measure of randomness or uncertainty of the information. Hash algorithms should yield high randomness in hashed message to be less or no dependency between input message and hashed code [8]. Entropy value $H_2(X)$ can be calculate by using equation (6)

$$H_2(X) = -\sum_{i=1}^{n} \frac{\text{count}_i}{N} \log_2 \frac{\text{count}_i}{N} \qquad \text{--------------------- (6)}$$

Where, X = Random Variable,

N = Total characters consisting of n different characters (n = 2 for binary)

Count_i = the count of character n_i [10].

Comparison results of entropy test during running time on 1KB message size for existing hash functions and proposed VRH function is displayed in Figure 9.

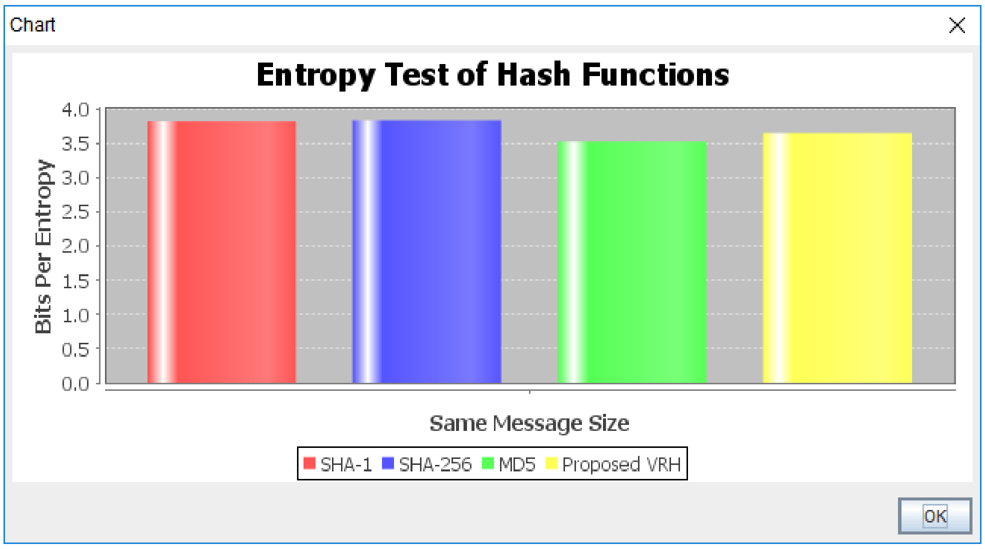

Figure9. Entropy test for hash functions

Comparison results of entropy test on various message sizes for the proposed VRH function and some existing hash functions are described in Table2.

Table2. Entropy test for hash functions on various message sizes

Algorithm	Message sizes				
	1 KB	**1.5 KB**	**2 KB**	**2.5 KB**	**3 KB**
SHA-1	3.68	3.67	3.67	3.67	3.66
SHA-256	3.92	3.81	3.75	3.79	3.86
MD5	3.51	3.77	3.81	3.67	3.69
Proposed VRH	3.54	3.7	3.64	3.74	3.63

According to the results in Table2, entropy of the proposed VRH function has nearly same level with that of SHA-1 for all message sizes. The entropy value of proposed function is higher than that of MD5 only for 1KB and 2.5KB message sizes and SHA-256 is the highest for all messages. Nevertheless, in the proposed VRH, the larger message size, the more increase the randomness of hash value.

Frequency Test: Frequency test analyses the proportion of number of ones and zeros in the entire sequence. If p-value > 0.01, the sequence is random otherwise the sequence is not random. The distance of one and zero provides the nonlinearity and confusion for the attacker [7]. P-value can be computed in equation (7)

$$\text{P-value} = \text{erfc}(z) = \frac{2}{\sqrt{\pi}} \int_{z}^{\infty} e^{-u^2}\, du \qquad \text{------------------------ (7)}$$

Where, n = The length of the bit string

ε = The sequence of bits: $\varepsilon = \varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n$.

$S_n = X_1 + X_2 + \ldots + X_n$, where $Xi = 2\varepsilon_i - 1$.

$S_{obs} = \frac{|Sn|}{\sqrt{n}}$, The absolute value of the sum of the Xi in the sequence z = The reference distribution, $\frac{S_{obs}}{\sqrt{2}}$

erfc = complementary error function [11]

The comparison results of frequency test which analyzed on various message sizes for the proposed VRH and some existing hash functions is displayed in Figure10.

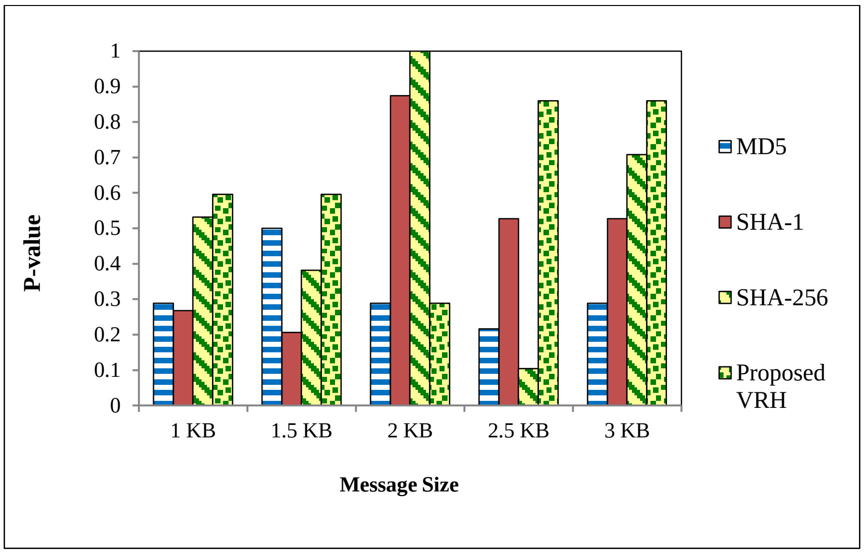

Figure10. Frequency test for hash functions

In this case, the proposed VRH function can produce the better random sequence than other existing hash function for all message size except form SHA-256 in 2KB message size. Consequently, the proposed function can provide robustly nonlinear dependency and confusion properties.

Collision resistance: A good hash function tends to generate different hash values when given different inputs [12]. The probability of hash collision depends constantly on the output length of hash code. The probability of a hash collision can be computed in equation (8).

$$1 - e^{\frac{-k(k-1)}{2N}} \quad \text{----------------------- (8)}$$

Where, N = Possible hash values

k = number of hash values generating from random messages

For that reason, the longer output length of hash code can be more resistance for collision than shorter lengths as shown in Figure 11.

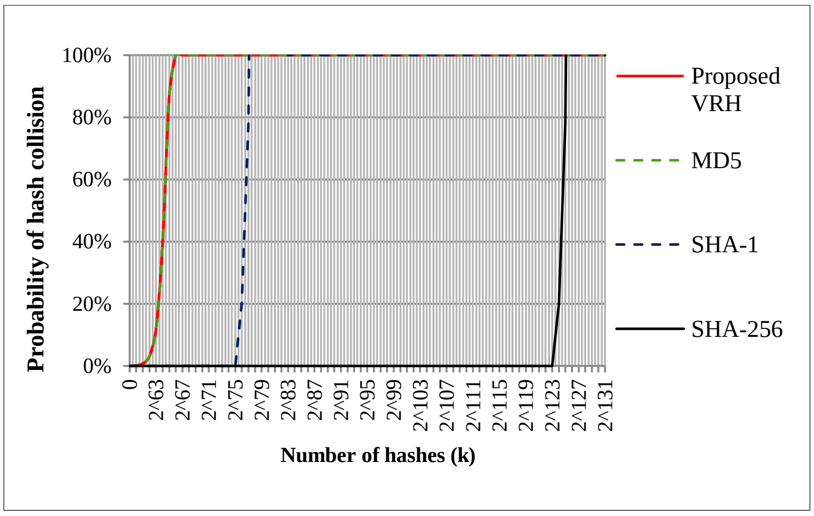

Figure11. Probability of hash collision for hash functions

The proposed VRH and MD5 have the same length of hash code in 128-bit length therefore the probability of hash collision is same at approximately number of hashes (2^{67}).

5.2 Execution Time

For performance consideration, the running time of the proposed algorithms was analyzed by implementing in core i5 processors; RAM 4GB, window 10, 64-bit operating system. The running times of the proposed VRH function and some existing algorithm: SHA-1, SHA-256 and MD5 are compared as presented in Figure12.

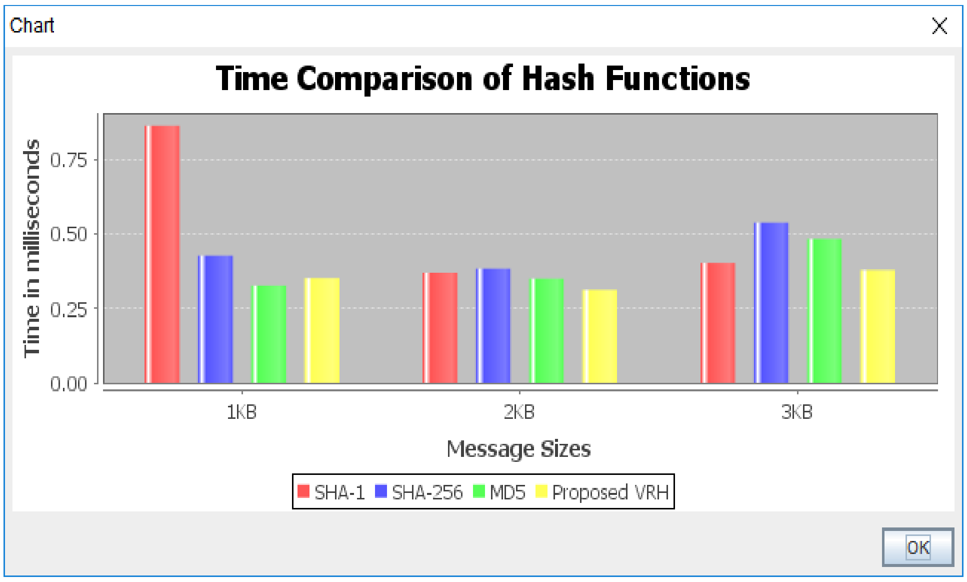

Figure12. Comparison results of execution time for hash functions

According to the experimental results, the execution times of existing hash functions are not different since message expansion, block size and compression functions of these functions are identical and only the initial hash values, rounds and output sizes are different. For above 1KB message size, the proposed VRH provides faster execution time than that of the existing hash functions due to using only four 32-bits working variable and smaller block size to operate and variable rounds depending on the message size.

6. DISCUSSION AND CONCLUSIONS

This research focused on security issues of DHKE algorithm and developed an authenticated key exchange approach. The proposed key exchange approach can generate a more secure session key for the encryption process with the help of proposed VRH function. In the proposed VRH function, the original message cannot be recovered from the hash code for the reason that number of round operations of the internal functions depends on the length of the message. According to the experimental results, the proposed hash function achieves the good security level. In addition, the total execution time of the proposed VRH function is faster than that of some existing hash functions. The DHKE algorithm with VRH hash function can prevent MITM attack since an attacker cannot create a pair of hash code and public key without knowing the internal bitwise operations of both sites. This algorithm can be used together with many cryptographic algorithms such as symmetric encryption algorithms for secure key transportation.

REFERENCES

1. C. Paar, and J. Pelz, *Understanding Cryptography, A textbook for students and practitioners*, London New York: Springer, 2010.
2. N. Li, “Research on DHKE Protocol”, *2nd International Conference on Computer Engineering and Technology, IEEE*, Vol. 4, 2010, pp. 634-637.
3. J. Preetika, V. Manju and R. V. Pushpendra, “Secure Authentication Approach Using DHKE Algorithm for WSN”, *International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT), IEEE publisher*, 2015, pp. 527-532.
4. S. K. Aqeel, and L. David, “Preventing Man-In-The-Middle Attack in DHKE Protocol”, *22nd International Conference on Telecommunications (ICT), IEEE publisher*, 06 August 2015, pp. 204-208.

DOI: 10.31695/IJASRE.2019.33560

5. Aryan, K. Chaithanya, and P. M. Durai Raj Vincent, "Enhanced DHKE Algorithm for Reliable Key Exchange", *14th ICSET, IOP Conference of Series: material Science and Engineering*, 2017, Vol. 263, pp. 1-8.
6. W. Diffie, and M. E. Hellman, "New Directions in Cryptography", *IEEE Transactions on Information Theory*, Vol. IT-22, No. 6, 1976.
7. Anonymous, Permutations and combinations: Discrete mathematics and probability, *https://www.mathplanet.com/ education/algebra-2/discrete-mathematics-and-probability/ permutations-and-combinations*, 2019.
8. K. S. Gurpreet, and S. G. Gurjot, "An Efficient Hash Algorithm to Preserve Data Integrity", *Journal of Engineering Science and Technology*, Vol. 13, No.3, 2018, pp. 778-789.
9. G. Karthi, and M. Exhilarasan, "Enhanced VSDL Hash Algorithm for Data Integrity and Protection", *Data Management, Analytic and Innovation, Advances in Intelligent Systems and Computing, Springer*, 2019, pp. 527-539.
10. Anonymous, Entropy, *https://rosettacode.org/wiki/Entropy*, 2019
11. A. Rukhin, J. Soto, S. Leigh, M. Levenson: *A Statistical Test Suite for Random and Pseudorandom Number generators for Cryptographic Applications*, National Institute of Standard and Technology (NIST), Special Publication 800-22 Revision 1a, April 2010.
12. J. Preshing, Hash Collision Probabilities, *https://preshing.com/20110504/hash-collision-probabilities/*, 2011.

DOI: 10.31695/IJASRE.2019.33560