Preventing Man-In-The-Middle Attack in Diffie-Hellman Key Exchange Protocol

Aqeel Sahi Khader
Department of Mathematics and Computing
University of Southern Queensland
Toowoomba, Queensland, Australia
akeel_sahy@yahoo.co.uk

David Lai
Department of Mathematics and Computing
University of Southern Queensland
Toowoomba, Queensland, Australia
david.lai@usq.edu.au

Abstract— **The acceleration in developments in communication technology has led to a consequent increase in the vulnerability of data due to penetration attacks. These attacks often came from outside where non-qualified companies develop IT projects. Cryptography can offer high levels of security but has recently shown vulnerabilities such as the man-in-the-middle (MITM) attack in areas of key exchange protocols, especially in the Diffie-Hellman (DH) protocol. Firstly, this paper presents an overview of MITM attacks targeted at the DH protocol then discusses some of the shortcomings of current defenses. A proposed method to secure DH, which helps secure systems against MITM attacks, is then presented. This method involves the use of Geffe generation of binary sequences. The use of Geffe generator offers high levels of randomness. Data hashed and encrypted using this proposed method will be so difficult to intercept and decrypt without the appropriate keys. This offers high levels of security and helps prevent MITM attacks.**

Keywords*— *public key; random number generation; data security; cryptography; message authentication; digital signatures.

I. INTRODUCTION

Nowadays, data security and privacy are a controversial issue. Researchers are trying to do their best to find out the perfect way to secure the data efficiently. One of these solutions is encrypting the data using cryptography algorithms. Cryptography is the science of converting the readable information into unreadable or hidden, and only the authorized persons or machines can retrieve or obtain the original texts. It is divided into two major types (symmetric and asymmetric) regarding their keys. (1) Symmetric cryptosystems require the users to have the same key to be used for encryption and decryption processes. (2) Asymmetric cryptosystems require the users to have two different keys (encryption key and decryption key). Symmetric key should be changed from time to time to make it more secure and unbreakable to prevent other users from obtaining the plain text. Therefore, the security of any symmetric cryptography system depends on key exchange protocol used by the system. Key exchange protocol is the way of distributing the keys in a secure manner among the users [1]. There are so many ways to exchange the keys between Alice and Bob, Alice can choose a key then to send to Bob physically (mail or person). Alternatively, if they have an old key so Alice can choose new key and encrypt it by the old one and send to Bob. Unfortunately, those techniques are not secure in wide distributed systems nowadays for many reasons. Consequently, the secrecy of their messages will rely heavily on the chosen key exchange protocol, for example, Diffie-Hellman. Furthermore, asymmetric cryptosystems process is slower than symmetric one [1]. Symmetric cryptosystems uses a shared secret key to be used for encrypt and decrypt processes. While asymmetric cryptosystems does not use a single shared secret key, as an alternative it uses mathematical key pairs: a private and public key. In this cryptosystems the communications are decrypted with the private key and are encrypted with the public key. As a result asymmetric cryptosystems have consuming too much computing power to produce the two keys (private and public) and that is why it's slower than symmetric cryptosystems.

Diffie-Hellman key exchange protocol was brought in 1976 by Whitfield Diffie and Martin Hellman [2], this protocol is widely used for secure key exchange. The process of this protocol supposes that Alice and Bob have different private keys and they have to agree upon two relatively prime numbers p, g then each of them uses the obtained information to calculate the public keys. After that they share their public keys between each other and use it with the private one, p and g to get the same shared key. As a result, both of Alice and Bob obtained the shared key without sending their private keys through the channel [3].

In this paper, we will explain the problem of Man-In-The-Middle (MITM) attack in Diffie-Hellman protocol and the current defenses. This paper proposes improvements which can help preventing MITM attack. These can verify and secure the communications between Alice and Bob.

Users have usernames and passwords, and these will be used to connect them with their systems. Passwords as known are a mixed string, and it can contain numbers, symbols, lowercase, and uppercase. These mixtures can be represented in binary using ASCII code. The binary can be used to generate a different binary sequence using random number generators and extract new information, which can be helpful to overcome the problem.

In section II discuss the related work, section III present the proposed work and section IV describe the conclusion and future work.

II. RELATED WORKS

As mentioned, the scheme of DH was brought in 1976 by Whitfield Diffie and Martin Hellman [2], Diffie-Hellman Key Exchange (DHKE) is one of the best protocols to exchange the keys safely. It has so many advantages that made most of the cryptographers feel confidante when they use it in their cryptosystems. However, DHKE has a serious issue called Man-In-The_Middle attack (MITM); this algorithm is vulnerable to this kind of attack in which Eve can attack (active attack, she can modify) all communications between Alice and Bob [4]. Because of this type of attack, researchers tried to find the best defense for that, as a result they came out with several answers. The most famous solutions that the researchers have found are: Digital Signatures (DS) and Message Authentication Code (MAC). Even though they are convenient for many systems, but they still have some weaknesses we need to think about.

Digital Signatures (DS): Generally, DS used the private key to sign the message and uses the public key to verify the message in the other party. DS faces difficulties with securing the private keys, verification and secrecy. In terms of securing the private keys, if they are exposed, all security promises are gone [5]. Therefore, most of developers and cryptosystem designers believe that we should keep the private keys in a safe way (like never send or store the private keys in plain text). If we lose them, we could have so many damages. Subsequently, anyone who gets the private key can sign the message and send to the other party (the person who has the public key, Bob), he will recognize it as a valid message (the private key holder signed the message, Alice), hence Bob will believe that the message was sent by Alice.

Every person who holds the private key can sign a document. In other words, the DS does not automatically ensure that Alice signed the document. It does ensure the Alice document was signed by someone who had access to the private key [6]. So it is like stamps, it can be stolen and utilize by other people. In relation to verification and secrecy, when the DS failed to be verified by Bob (the PuK holder) the system will flag the message as invalid because it cannot be determined whether the message was corrupted by Eve or Alice used a false private key. This means that Alice has to be responsible for the security of each of her private keys. The DS can provide authenticity but cannot provide security. Therefore encryption and decryption are required to add security. Without this, DS cannot prevent the message from being intercepted changed by MITM [6].

Message Authentication Code (MAC): MACs differ from DS as in MAC both side (Alice and Bob) need to use the

same key for generating the keyed MAC tag and verification.

Commonly, Alice generates MAC tag using MAC algorithm and sends the tag with the message to Bob. Bob uses the same MAC algorithm with the same key to generate the keyed MAC tag. Then Bob check his MAC tag with Alice MAC tag if they are identical, then he will mark the message as a valid message, else he will mark the message as an invalid message.

Therefore, both of sender and receiver must agree on the same shared key and MAC algorithm before they start their communications, just like the case with symmetric-key systems. Because of that, Bob cannot prove that the message coming from Alice (unlike digital signature), which mean MAC does not provide a non-repudiation property. Anyone who receives the message and can confirm a MAC, for example Bob, can produce MACs for different messages. Thus, 'MACs are symmetric-key schemes, and they do not provide a non-repudiation' [7].

As a result, Eve can attack the communication channels and record the messages that would be sent from Alice to Bob, and later Eve sends a copy of the messages to Bob. This will make Bob feels that those messages coming from Alice. Eve might send the message back to Alice, who would trust that it came from Bob [8].

III. PROPOSED WORK

The proposed work aims to distribute the keys between Alice and Bob without being compromised by Eve. Figure 1 explains the way that Alice and Bob communicates with the server and the data that would be sending through the channels to retrieve the shared key.

Assume that the password of Alice contains eight characters (password = abcdefgh (a character string), 64 bits). The binary representation of the ASCII code for "abcdefgh" is: a=97 (01100001) + b=98 (01100010) + c=99 (01100011) + d=100 (01100100) + e=101 (01100101) + f=102 (01100110) + g=103 (01100111) + h=104 (01101000). As a result we have obtained the password in binary numbers. The sequence will be: 0110000101100010011000110110010001100101011001100110011101101000 (64 bits)

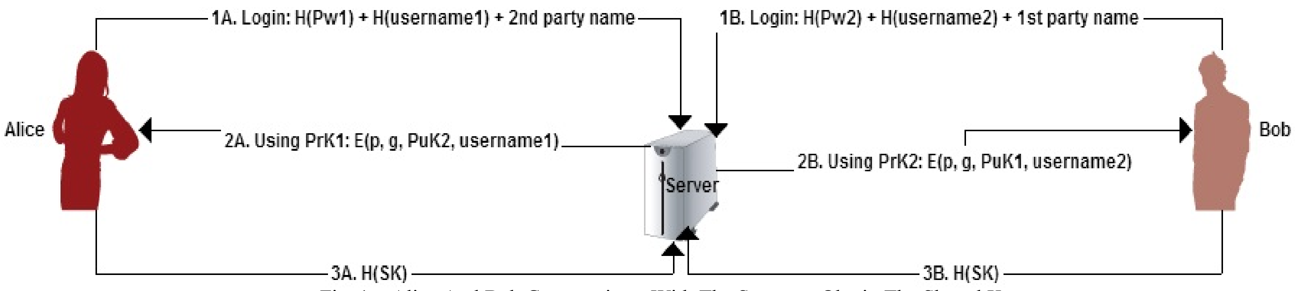
1A. Login: H(Pw1) + H(username1) + 2nd party name
1B. Login: H(Pw2) + H(username2) + 1st party name
Alice
2A. Using PrK1: E(p, g, PuK2, username1)
Server
2B. Using PrK2: E(p, g, PuK1, username2)
Bob
3A. H(SK)
3B. H(SK)

Fig. 1. Alice And Bob Communicate With The Server to Obtain The Shared Key

To make it more secure we used a pseudo random sequence generator to generate longer sequence.

A. Geffe Generator

Geffe generator is a pseudo random sequence generator. It uses three registers to generate one random sequence. The Geffe generator chose as it is in stream cipher cryptography to generate the new sequence. This generator used for several reasons: Firstly, the complexity of this generator could be superior in a different configuration of the steps. Secondly, the device does have some necessary features; for example, it has a balanced distribution of 0's and 1's in its output. It also gives the benefit of being useful as a module of a building of similar arrangements. The whole device could play the role of LFSR1 (linear feedback shift registers) in the same arrangement with like generators, and this complexity would escalate accordingly [9]. The Geffe generator needs three (LFSRs). The length of those LFSRs should be relatively prime, which mean the greatest common divisor (GCD) of the length of the initial values that would be input into the three registers LFSR1 (Len1), LFSR2 (Len2) and LFSR3 (Len3) is one:

$$\text{GCD (Len1, Len2, Len3)} = 1 \tag{1}$$

Suppose the first one will have 20 bits as input for LFSR1, the second will have 21 bits as input for LFSR2 and the last one will have 23 bits as input for LFSR3, all together will be 64 bits (it is possible to choose any three numbers relatively primes and give us 64 bits).

20 bits	21 bits	23 bits
01100001011000100110,	001101100100011001010,	11001100110011101101000

And they should be connected as shown below:

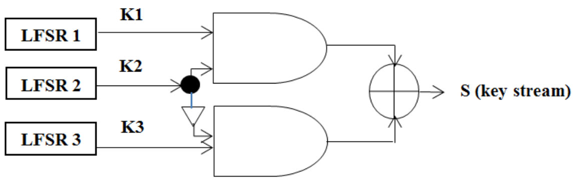

Fig. 1. Geffe Generator

The feedback functions are: K1= S10+S19, K2= S10+S20 and K3= S10+S22, successively. As an output of those registers, we will have three long binary sequences K1, K2 and K3, and each of them have the length 2^n-1 without repetition, which mean K1=1048575, K2=2097151 and K3=8388607. Then Geffe generator will mix them together and generate one binary sequence S have the length $(2^{K1}-1)$ $(2^{K2}-1)$ $(2^{K3}-1)$ = 18446715486418763775 without repetition. As shown in table I:

TABLE I. REGISTER'S PERIOD AND THEIR FEEDBACK FUNCTIONS

Bits	Feedback Function	Period
n		2^n-1
20	K1= $S_{10}+S_{19}$	1048575
21	K2= $S_{10}+S_{20}$	2097151
23	K3= $S_{10}+S_{22}$	8388607
64	S = K_1 K_2+K_2 K_3+K_3	$(2^{K1}-1)(2^{K2}-1)(2^{K3}-1)$ =18446715486418763775

Definitely if we generate this extreme large number (S) our system will slow down if not stop. So, we assume that we can generate sequences (S) with 1024, 512, 256 or 128 bits long, and that would be convenient to deal with. And after obtaining the binary sequence from Geffe generator, we will use three basic kinds of statistical test to check whether the randomness of the sequence is good or not. If the test successes we will continue with the next steps, else the system will ask the user to change his/her password. Finally, when the tests finished will decide which length could give us better results (1024, 512, 256 or 128).

B. Statistical tests

- Frequency test

For every binary random sequence, we expect that half of the sequence is 0's, and the other half is 1's, the purpose of this test relies on the number of 0's (n0) and the number of 1's (n1) in the sequence (n), which is we need to test. The static used is:

$$X1 = \frac{(n0-n1)^2}{n} \tag{2}$$

To check whether the sequence passes this test or not, for one degree of freedom, the value of X_1 should be less than the acceptance threshold values of the test (X1 < 3.8415) [10].

- Serial test

This test depends upon the repetition of (n00, n01, n10, and n11) which denotes the numbers (00, 01, 10, and 11) in S respectively. And we expect that each of them represents nearly a quarter of n. The static used is:

$$X2 = \frac{4}{n-1}\sum_{i=0}^{1} \sum_{j=0}^{1}(\text{nij})^2 - \frac{2}{n}\sum_{i=0}^{1}(ni)^2 + 1 \tag{3}$$

To check whether the sequence passes this test or not, for two degree of freedom, the value of X_2 should be less than the acceptance threshold values of the test (X2 < 5.9915) [10].

- Poker test

This test is divide the sequence S into a number of blocks (K) have the length (M), and then check the repetition of those blocks and determine whether they appear approximately the same number of times as would be expected for a random sequence, the number of block's K=n/M (without fractions). The static used is:

$$X3 = \frac{2^m}{k}\left(\sum_{i=1}^{2^m} ni^2\right) - k \tag{4}$$

To check whether the sequence passes this test or not, for 2^m-1 degree of freedom, the value of X_3 should be less than the acceptance threshold values of the test (X3 < 14.0671) [10]. The poker test is a generalization of frequency test when we use m=1 we obtain results as frequency tests results.

In our work, we used those three tests to check the sequence S with 1024, 512, 256 and 128 bits long (with the current password), and the results are shown in the table below:

TABLE II. STATISTICAL RESULTS

Length of Pseudo	Statistical test	Value of statistical test	D. of freedom	Acceptance threshold
1024 its	Frequency test	0.03515625	1	3.8415
	Serial test	2.04011256720423	2	5.9915
	Poker test	9.19354838709677	7	14.0671
512 bits	Frequency test	0.0703125	1	3.8415
	Serial test	0.452192392367863	2	5.9915
	Poker test	7.2235294117647	7	14.0671
256 bits	Frequency test	1	1	3.8415
	Serial test	1.42745098039217	2	5.9915
	Poker test	7.51764705882353	7	14.0671
128 bits	Frequency test	5.28125	1	3.8415
	Serial test	4.56914370078741	2	5.9915
	Poker test	7.90476190476191	7	14.0671

As shown in Table II, three statistical tests have been used to check the randomness os the sequences. When the sequence S was 128 bits, it failed with frequency test (the sequence generated by password = abcdefgh does not pass the test). S equal to a 256 bit have been chosen, because it's the shortest successful length, and still can ensure that the sequence has a good randomness and small size. As illustrated in Table II, the frequency test increased every time when the bits' size decreased, and that was the reason behind stopping until 128 bits and chose 256 bits. The 256 bits sequence we got from the password is:
0100010011100100011001000000100100000010101000001111111011101010101010100
0011000100110001000111100111000000011010101010010000100011111100110110 01
1010100010001101010111001010010110110100001001011100001010110111110010 01
0010010110001111011100111011111000111010

This sequence derived from Alice password and it has passed all the tests. Next, convert every 8 bits into decimal number using ASCII code. As a result we got those numbers: 68 228 100 18 5 65 253 213 84 49 49 30 112 26 169 8 252 217 168 141 92 165 180 37 194 183 201 37 143 115 190 58, then modular each of them by 10 to ensure that all the numbers will be represented by one digit (1, 2, 3, 4, 5, 6, 7, 8 or 9). Merge them all together to get one number with 32 digits 88085533499026982781250743173508. Then assumed that the user's private key will be eight digits (the private key should be between 2 and P-2 [7], P will be provided by the server), therefore will divide the 32 digits number into eight blocks and take one number from each block, and those eight digits will be the private key (take the first number from each block (randomly), the private number will be PrK=85422243). In this stage, every user has his/her own private key and the server has all of them together in one table.

Alice and Bob get two numbers g and p, where p is a prime and g (generator) is a primitive root Modulo p, and those two numbers are coming from the server with the public key, Alice will receive p, g, Bob public key and username, Bob will receive p, g, Alice public key and username. They will be encrypted using the receiver private key. As mentioned above, the private key should be less than the prime p [7]. Hence, our private key less than nine digits will satisfy the requirements. For our example let p=100000007 and g=5. The public key will be PuK=15071649. If Alice pw=abcdefgh and Bob pw=12345678 (a character string), then the private key of Alice PrK=85422243 and Bob PrK = 68203955. After mix them with p and g will get the PuK of Alice PuK=15071649 and Bob PuK = 6629794. As a result the shared key SK = 68202249 (for both Alice and Bob).

Noticeably, the central server would not use the existing identification technique such as X.509 for some reasons: X.509 certificate require digital signature to be used for integrity, thus it has the same problems of using DS [11]. In addition X.509 certificate has problem with certificates expire, sometimes no one knows that the certificate has been expired until the website or sever goes down. It is really hard to figure out what is going on.

C. Algorithm

- Registration time

1. The system converts the eight character password into a 64 bit binary sequence using ASCII code.
2. The binary sequence is divided into three registers with lengths of 20, 21, and 23 respectively as indicated in section III. The Geffe pseudo random sequence generator is used to generate one sequence with 256 bit.
3. The resulting sequence, S is tested using the three basic statistical tests frequency, serial and poker:
   a. If the sequence passes all the tests, then continue,
   b. Else, go back to step 1 and ask the user to try another password.
4. Convert the successful sequence into a decimal number. This is then divided into eight blocks to obtain the eight digit private key using a digit from each of the block.
5. The server uses the same method to calculate PrK1 and PrK2.
6. The administrator sends an activation message to Alice and Bob asking them to activate their passwords:
   a. Alice and Bob receive the activation message, and then continue,
   b. Else, Pw1 and Pw2 have changed by Eve. Go to 15
7. The server hashs Pw1and Pw2, encrypts PrK1 and PrK2 using the admin key, and add salt (random number) then saves these in one table: the user info table.

- Log in time

8. The server redirects Alice and Bob to the second page and provides them with p, g, PuKs and usernames. These will be encrypted using the received private key where p and g will change for every session. Alice and Bob will need their password to access the information. The public keys are calculated using the following equations [3]:

$$PuK_1 = g^{PrK1} \bmod p \qquad (5)$$

$$PuK_2 = g^{PrK2} \bmod p \tag{6}$$

9. The server redirects Alice and Bob to the second page and provides them with p, g, PuKs and usernames. These will be encrypted using the receiver private key. Alice and Bob have to use their password to access the information.
10. Alice and Bob compute their shared key (SK), and the server also calculates shared key (SK). The following equations are used to calculate the shared keys [3].

$$SK_{Alice} = PuK_2^{PrK1} \bmod p \tag{7}$$

$$SK_{Bob} = PuK_1^{PrK2} \bmod p \tag{8}$$

11. The server hashes the SK and saves it in the user info table.
12. Alice and Bob hash their shared keys (Sk) and send H(Sk) to the server.
13. The server checks the resulted hashed shared key H(Sk) with the hash table:
    a. If H(Sk) matches with one of the hashes in the hash table, then continue.
    b. Else, Sk has come from Eve. Go to 15
14. Alice and Bob are ready to share their messages encrypted with the shared key. Go to 16
15. Warning message: “Your data were compromised”.
16. End.

TABLE III. OBTAINED DATA

	Alice	Server	Eve	Bob
1	Alice-username	Alice-username & Bob-username	H(Alice, Bob) username	Bob-username
2	Pw1, PrK1	Pw1, Pw2, PrK1 & PrK2	H(Pw)	Pw2, PrK2
3	P, g	P, g	E(P, g, PuKs, username)	P, g
4	PuK1	PuK1, PuK2	--	PuK2
5	SK	SK	--	SK

As shown in Table III, Eve can only obtain hashed and encrypted data and she cannot use them to intercept the communication channels.

To sum up, the MITM attacks occur when we share our keys in plain text. Eve can sit in the middle and pretend she is the intended destination for both Alice and Bob, Alice and Bob have no way of knowing Eve is there and believe they are communicating directly with each other. The proposed method proved that Alice and Bob could generate the keys without sending them in plain text using the data obtained from the server, so that will be controlled by the server. Even though some may argue that Eve can compromise all the data on the channels, but that will be not enough for her to get the shared key, all the data will be hashed and encrypted. In addition, the private key will never be sent to any party. If an attacker steal the private key from the server it would be not useful for him/her due to the entire private keys in the server are encrypted using the admin key. This offers high levels of security and helps prevent MITM attacks in DH. There are many applications used DH as a key exchange like SSL (Secure Sockets Layer), Secure Shell (SSH) and IP Security (IPSec) [12]. The proposed infrastructure could distribute the keys in secure manner and these applications would benefit from the proposed infrastructure.

IV. CONCLUSION

An efficient method to harden the Diffie-Hellman protocol against man-in-the-middle attack was described within this paper. Geffe generator was used to generate a binary sequence with a high level of randomness. Furthermore, statistical tests are used to check these sequences, before calculating the private key and the shared key. The proposed method ensures that the private keys will not be sent through the channels, and will be saved as hashes in the server. Additionally, it provides a non-repudiation property, as it can identify the sender and the receiver from their user information. As a result, this method provides more security properties than existing methods and prevents MITM attack. In the future, we will employ this method over other encryption methods to provide a secure cryptosystem for sharing our messages securely. The researchers are planning to employ the proposed algorithm in real cloud cryptosystem.

REFERENCES

[1] N. Kumar, P. Gupta, M. Sahu, and M. Rizvi, "Boolean Algebra based effective and efficient asymmetric key cryptography algorithm: BAC algorithm," in Automation, Computing, Communication, Control and Compressed Sensing (iMac4s), 2013 International Multi-Conference on, 2013, pp. 250-254.

[2] W. Diffie and M. E. Hellman, "New directions in cryptography," Information Theory, IEEE Transactions on, vol. 22, pp. 644-654, 1976.

[3] C. K. Kumar, G. J. A. Jose, C. Sajeev, and C. Suyambulingom, "Safety measures against man-in-the-middle attack in key exchange," Asia Research Publishing Network (ARPN) Journal of Engineering and Applied Sciences, vol. 7, pp. 243-6, 2006.

[4] M. K. Ibrahem, "Modification of Diffie-Hellman key exchange algorithm for Zero knowledge proof," in Future Communication Networks (ICFCN), 2012 International Conference on, 2012, pp. 147-152.

[5] J.-h. Chen, Y. Long, K.-f. Chen, Y.-t. Wang, and X.-x. Li, "An efficient threshold key-insulated signature scheme," Journal of Shanghai Jiaotong University (Science), vol. 16, pp. 658-662, 2011.

[6] F. Boudrez, "Digital signatures and electronic records," Archival Science, vol. 7, pp. 179-193, 2007.

[7] B. Preneel, C. Paar, and J. Pelzl, Understanding cryptography: a textbook for students and practitioners. London: Springer, 2009.

[8] N. Ferguson and B. Schneier, Practical cryptography vol. 141: Wiley New York, Indianapolis, 2003.

[9] S. Wei, "On Generalization of Geffe’s Generator," IJCSNS August, vol. 6, pp. 161-5, 2006.

[10] S. M. Hosseini, H. Karimi, and M. V. Jahan, "Generating pseudo-random numbers by combining two systems with complex behaviors," Journal of Information Security and Applications, pp. 149–62, 2014.

[11] T. Cristian, "Security Issues of the Digital Certificates within Public Key Infrastructures," Informatica Economica, vol. 13, pp. 16-28, 2009.

[12] M. Ahmed, B. Sanjabi, D. Aldiaz, A. Rezaei, and H. Omotunde, "Diffie-Hellman and Its Application in Security Protocols," International Journal of Engineering Science and Innovative Technology (IJESIT), vol. 1, pp. 69-73, 2012.