Diffie Hellman Key Exchange

Rupa Ganjewar

Abstract— **Diffie–Hellman key exchange is a cryptographic protocol that allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure communications channel. This key can then be used to encrypt subsequent communications using a symmetric key cipher. The initial implementation of the protocol used the multiplicative group of integers modulo p, where p is prime and g is primitive root** mod$p$**. Both parties arrive at the same value g^{ab} and g^{ba} both equal to** $\bmod p$**, and use this value to encrypt and decrypt data. Only a, b and** $\bmod p$ **are kept secret (private key). All the other values viz., p, g, $g^a \bmod p$, and $g^b \bmod p$ are sent in the clear (public key). An efficient algorithm to solve the discrete logarithm problem would make it easy to compute a or b and solve the Diffie–Hellman problem, making this cryptosystem insecure. The Diffie–Hellman protocol can be strengthened by using elliptic curve cryptography, which makes the discrete logarithm problem almost impossible to solve. Advantages of ECC include its small key size and strong security, for e.g., a 160-bit key in ECC is considered to be as secure as 1024-bit key in RSA. When using Elliptic Curve Diffie–Hellman, it is practically impossible to find the private key from the public key. The Diffie–Hellman key exchange is vulnerable to a man-in-the-middle attack, where an opponent can intercept and decrypt secured communication by subverting the keys being exchanged. We describe a few solutions to preventing man-in-the-middle attacks by modifying the Elliptic Curve Diffie–Hellman protocol.**

I. Introduction

Cryptography is the science of information security. Cryptography is conventionally referred to the process of converting ordinary information (plaintext) into unintelligible gibberish (i.e., ciphertext). Decryption is the reverse, in other words, moving from the unintelligible ciphertext back to plaintext. Consider Alice is interested in sending a private message (plaintext) to Bob after encryption (ciphertext). A cryptosystem consists of a finite set of possible plaintexts, a finite set of possible ciphertexts, a finite set of possible keys, an encryption rule for encrypting plaintext into ciphertext and a decryption rule for decrypting ciphertext back to plaintext. The general idea behind any cryptosystem is that Alice and Bob must share a sector key which is used to encrypt a message, and without which the plaintext cannot be recovered.

Private-key Cryptosystems are such that there is a way for Alice and Bob to secretly share a key k prior to the transmission of plaintext and they can use encryption and decryption rules defined by their secret value of k. One approach to sharing keys is the *key agreement protocol* whereby Alice and Bob jointly establish the secret key by using values they have sent to each other over a public channel. On the other hand, Public-key Cryptosystems are such that Bob keeps his key (and his decryption rule) to himself, whereas the corresponding encryption rule is publicly known. Therefore, Alice can send encrypted messages without any prior sharing of keys, and Bob will be the only person able to decrypt the message sent to him.

The Diffie–Hellman protocol is a public-key cryptosystem developed in 1976 and published in a ground-breaking paper [1]. Diffie-Hellman is not an encryption mechanism to encrypt data. Instead, it is a method to securely exchange the keys that encrypt data. Diffie-Hellman accomplishes this secure exchange by creating a "shared secret" (sometimes called a "key encryption key") between two devices. The shared secret then encrypts the symmetric key (or "data encryption key" i.e. DES, Triple DES, CAST, IDEA, Blowfish, etc.) for secure transmittal [2]. The Diffie–Hellman Key Exchange's security depends on the difficulty of solving the discrete logarithm problem. Computing the discrete logarithm of a number modulo is difficult since it takes roughly the same amount of time as factoring the product of two primes, which is the basis for RSA algorithm. Therefore, the Diffie–Hellman protocol is roughly as secure as RSA.

Section II of this paper introduces the Diffie–Hellman concept, Section III describes the security of Diffie-Hellman, and applications of Diffie-Hellman, Section III man-in-middle attack on Diffie–Hellman. Section IV describes a solution to the man-in-middle attack. Application of Elliptic Curves is described in Section V with some concepts to prevent man-in-the-middle attacks.

II. Diffie–Hellman Concept

The Diffie–Hellman protocol is based on the discrete logarithm problem (DLP). The protocol re-

Author is with the Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106, Email: rupa@gadewar.com

Manuscript submitted on June 9, 2010.

quires two large numbers p and g, where, p and g are both publicly available numbers. Parameter p is a prime number with at least 512 bits and parameter g (called a generator) is an integer less than p, with the property: for every number n between 1 and p-1 inclusive, there is a power k of g such that $n = g^k \bmod p$.

Suppose Alice and Bob want to agree on a shared secret key using the Diffie–Hellman key agreement protocol, they will generate shared key as follows: first, Alice generates a random private value a and Bob generates a random private value b. Both a and b are drawn from the same set of integers. Then they derive their public values using parameters p and g and their private values. Alice's public value is $x = g^a \bmod p$ and Bob's public value is $y = g^b \bmod p$. They then exchange public values x and y. Finally, Alice computes $ka = y^a \bmod p$, and Bob computes $kb = x^b \bmod p$. Since $ka = kb = k$, Alice and Bob now have a shared secret key k [3]. Figure 1 depicts a simplified schematic of the Diffie–Hellman key exchange.

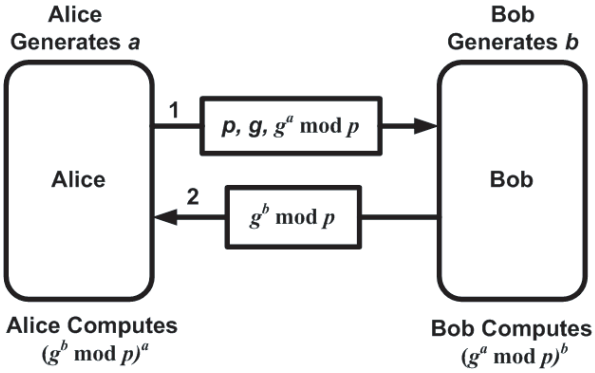

Figure 1: Diffie Hellman Protocol

A. Diffie–Hellman Key Exchange Algorithm

Let p be a large prime and assume that α is a primitive element of Z_p, p and α are publicly known [4].

1. Alice choose a $(0 \leq a \leq -2)$at random.
2. Alice computes $x = g^a \bmod p$ and sends it to Bob.
3. Bob chooses b $(0 \leq b \leq -2)$at random.
4. Bob computes $y = g^b \bmod p$ and sends it to Alice.
5. Alice computes $(g^b)^a \bmod p$

whereas Bob computes $(g^a)^b \bmod p$.

In other words, both Alice and Bob compute the same key

$g^{ab} \bmod p$.

Figure 2 depicts an example of the Diffie–Hellman key exchange. If $p = 170141183460469231731687303715884105727$, then it would take roughly 1.14824×10^{21} steps to solve, and each step requires many calculations. Even using Google′s computers which are estimated to perform 300 trillion calculations per second, it would take roughly 5 years to solve.

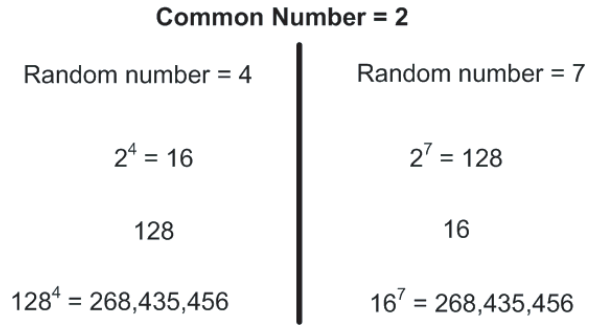

Figure 2: Example of Diffie Hellman Key Exchange

B. Security of Diffie–Hellman

If eavesdropper, Mallory, learns integers p, g, x, y but not the discrete logarithm a of x and b of y to the base g. She wants to determine the secret key $k = g^{ab} \bmod p$ from p, g, x, y. This is called Diffie-Hellman problem. She can compute discrete logarithms $\bmod p$, and try to solve the Diffie-Hellman problem. She determines the discrete logarithm b of y to the base g and computes the key $k = x^b$ This is the only known method for breaking the Diffie-Hellman protocol. Until now, no one has succeeded in breaking Diffie-Hellman problem. It is an important open problem of public-key cryptography to find such a proof. As long as the Diffie-Hellman problem is difficult to solve, no eavesdropper can determine the secret key from publicly known information [5].

C. Applications of Diffie–Hellman in Network Protocols

Diffie-Hellman is currently used in many network protocols, such as:

- Secure Sockets Layer (SSL)/Transport Layer Security (TLS).
- Secure Shell (SSH).
- Internet Protocol Security (IPSec).
- Public Key Infrastructure (PKI).
- In all major VPN gateway's today (PKI).

III. Man-in-the-middle Attack on Diffie–Hellman

The Diffie–Hellman key exchange is vulnerable to a man-in-the-middle attack. Though this system cannot be broken but it can be bypassed. In this attack, an adversary Mallory can not only intercept messages form Alice and Bob but also can send different message and stop sending the the original message. The

attack is as follows: an adversary Mallory intercepts Alice's public value and sends her own public value to Bob. When Bob transmits his public value, Mallory substitutes it with her own and sends it to Alice. Mallory and Alice thus agree on one shared key and Mallory and Bob agree on another shared key. After this exchange, Mallory simply decrypts any messages sent out by Alice or Bob, and then reads and possibly modifies them before re-encrypting with the appropriate key and transmitting them to the other party. This vulnerability is present because Diffie–Hellman key exchange does not authenticate the participants. The man-in-middle attack is shown in Figure 3.

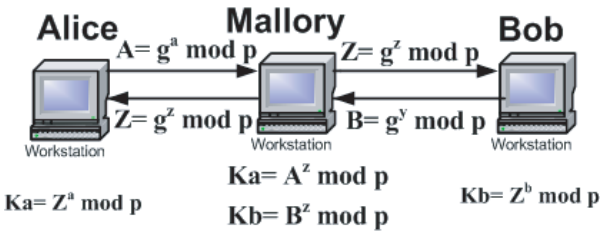

Figure 3: Man-in-the-middle attack against Diffie–Hellman

Imagine now that an adversary Mallory is capable of not only intercepting messages between Alice and Bob, but also stopping them and substituting his own messages instead. Then Mallory can do the following: pick his own random $e \in Z_p^*$, and compute $g^z \bmod p$. Then intercept g^a that Alice sends to Bob, and substitute g^z instead. Note that Bob does not notice any difference (because, after all, both g^a and g^z are random elements of Z_p^*) and dutifully replies with g^b. Mallory intercepts g^b, and sends g^z to Alice instead. This way, Alice ends up thinking that she is sharing $ka = (g^z)^a \bmod p$ with Bob, while Bob ends up thinking that he is sharing $kb = (g^z)^b \bmod p$ with Alice. Note that, in fact, they are both sharing a key with Mallory, who can compute ka and kb. Now whenever Bob tries to send something to Alice, he will presumably encrypt (and/or authenticate) it using kb . Mallory can intercept it, decrypt with kb, re-encrypt with ka, and send it on to Alice. So Bob and Alice will never realize they are not sharing a key with each other. This is known as "man-in-the-middle" attack, and is just one of the reasons why key agreement is a difficult problem. In fact, satisfactory formal definitions for key agreement took about a decade and a half longer to appear than definitions for encryption and signature.

IV. Solution to Man-in-Middle attack

A. Authentication

Possible solutions include the use of digital signatures and other protocol variants. The authenticated Diffie–Hellman key agreement protocol, or Station-to-Station (STS) protocol, was developed by Diffie, van Oorschot, and Wiener in 1992 [6] to defeat the man-in-the-middle attack on the Diffie–Hellman key agreement protocol. The immunity is achieved by allowing the two parties to authenticate themselves to each other by the use of digital signatures and public-key certificates. The basic idea is as follows: prior to execution of the protocol, the two parties Alice and Bob each obtain a public/private key pair and a certificate for the public key. During the protocol, Alice computes a signature on certain messages, covering the public value $g^a \bmod p$. Bob proceeds in a similar way. Even though Carol is still able to intercept messages between Alice and Bob, she cannot forge signatures without Alice's private key and Bob's private key. Hence, the enhanced protocol defeats the man-in-the-middle attack.

B. Digital Signature Algorithm

To provide authentication to the Diffie–Hellman key exchange Arazi used DSA (Digital Signature Algorithm) [7]. Digital signatures are used to sign electronic documents. They are similar to handwritten signatures. If Alice signs a document with her handwritten signature, then everybody who sees the document and also knows Alice's signature can verify that Alice has in fact signed document. For example, the signature can be used in a trial as proof that Alice has knowledge of the document and has agreed to its contents. In principle, digital signatures work as follows: suppose that Alice wants to sign the document m. She uses a secret key d and computes the signature s. Using corresponding public key e, Bob can verify that is in fact the signature of m [5].

The DSA signature scheme consists of DSA = (DSA.key, DSA.gen, DSA.ver). DSA.key generates a private-public key pair for the party. That is, a public key consists of a prime p, an order q which is also a prime, a generator g, and y. A private key is x such that $y = g^x \bmod p$. DSA.gen makes a signature $(r,\ s)$ for a message m with the private key x such that $r = ((g^k \bmod p) \bmod q)$ and $s = (k^-1(H(m) + xr)) \bmod p$, where k is a random value and H is a hash function. DSA.ver verifies a message-signature pair (m, r, s) with the public key y and returns 1 if valid or 0 otherwise. That is, the algorithm checks for $0 < r$,$s < q$ and

$((g^{H(m)S^{-1}}y^{rs^{-1}}) \bmod p) \bmod q = r$[7].

C. Message authentication

In this approach, the receiver should be sure about the senders identity. One approach to provide authentication is with the help of digital signature. The idea is similar to signing a document. Digital Signature provides the remaining three security services; Authentication, Integrity and Non-repudiation. Digital Signature There are two alternatives for Digital Signature: signing the entire document, signing the digest. In the first case, the entire document is encrypted using private key of the sender and at the receiving end it is decrypted using the public key of the sender. For a large message this approach is very inefficient. In the second case a miniature version of the message, known as digest, is encrypted using the private key of the digest created using the received message is compared the decrypted digest. If two are identical , it is assumed that the sender is authenticated .This is somewhat similar to error detection using parity bit.

D. User authentication

User authentication is different from message authentication since the identity of the sender is verified for each and every message for message authentication. On the other hand, user authentication is performed only once for the duration of system access.

V. Application of Elliptic Curve Cryptography

Elliptic Curve Cryptography (ECC) is a public key cryptography, where each user or device participating in the communication has a private and a public key. Only the particular user knows the private key whereas the public key is distributed to all users taking part in the communication. The mathematical operations of ECC is defined over the elliptic curve $y^2 = x^3 + ax + b$, where $4a^3 + 27b^2 \neq 0$. Each value of a and b gives a different elliptic curve. The public key is a point in the curve and the private key is a random number. The public key is obtained by multiplying the private key with the generator point G in the curve. The security of ECC depends on the difficulty of the elliptic curve discrete logarithm problem. If P and Q are two points on an elliptic curve such that $kP = Q$, where k is a scalar and sufficiently large, it is computationally infeasible to obtain k if k is the discrete logarithm of Q to the base P. Note that this is computationally infeasible even if P and Q are known. The main operation involved in ECC is point multiplication of scalar k with any point P on the curve to obtain another point Q on the curve. A schematic for the ECDH protocol is shown in Figure 4.

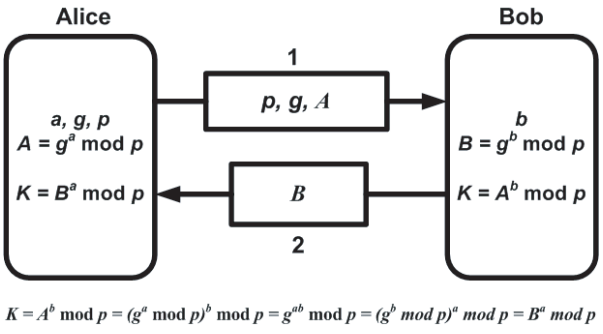

Figure 4: Elliptic Curve Diffie–Hellman Protocol

Elliptic curve Diffie–Hellman protocol (ECDH) is one of the key exchange protocols used to establish a shared key between two parties. ECDH protocol is based on the additive elliptic curve group. ECDH protocol begins by selecting the underlying field $\mathbf{F}_q$ or $\mathbf{F}_{2^k}$, the curve $\mathbf{E}$ with parameters a, b and the base point P. The order of the base point P is equal to k. The standards often suggest that we select an elliptic curve with prime order and therefore any element of the group would be selected and their order will be the prime number k. At the end of the protocol, the communicating parties end up with the same value K which is a point on the curve. The conventional ECDH protocol is susceptible to man-in-the-middle attack. Three approaches to modify the ECDH protocol for preventing man-in-the-middle attacks are summarized below.

A. ECC with no public point

Conventional protocols for elliptic curve cryptosystems such as ECDH assume that the curve $\mathbf{E}$, the field $\mathbf{F}_q$ and a point P on the curve are all public. A protocol proposed by Kaabneh and Al-Bdour [8] assumes that only the curve $\mathbf{E}$ and $\mathbf{F}_q$ are public, keeping the base point P secret, which makes it difficult to launch a man-in-the middle attack. Key steps in the protocol include:

- Two parties select prime number n and two parameters a and b satisfying the elliptic curve equations and also such that $n = pq$.
- p and q are chosen to be prime numbers. Integer e is selected such that $d \equiv e^{-1} \bmod (p-1)(q-1)$.
- Party A selects a first random number X_a in $\mathbf{F}_q$, a second random number R_a in $\mathbf{F}_q$, and a point P_a on the elliptic curve.
- Party B selects its own first random number X_b in $\mathbf{F}_q$, second random number R_b in $\mathbf{F}_q$, and a point P_b

on the elliptic curve.

The agreement between A and B has the following scheme:

- A computes point $G_a = X_a P_a$ and sends it to B, while B computes $G_b = X_b P_b$ and sends it to A.
- A computes point $S_a = R_a G_b$ and B computes point $S_b = R_b G_a$.
- A receives S_b from B and B receives S_a from A and each arrive at $K = e(S_a + S_b)$.
- Result K is cross-checked against result O from a previous interaction and the scheme terminates with a failure if $K = O$.

Multiplication by e gives the protocol public key characteristics so that the public key will be K for both parties and the private key will be $d(S_a + S_b)$. The protocol provides known-key security since each run of the protocol produces a unique session key. Although an adversary may have learned some other session key, they can't compute K, because they don't know the private keys d. The protocol also possesses forward secrecy since the two parties generate and share a unique key for each session. A two pass key agreement used in this protocol requires that the imposer has to intercept two sets of values, viz., G_a, S_a and G_b, S_b. Also, since the value e is secret, it is not possible for the imposer to calculate key K.

B. Authenticated key agreement with pre-shared password

One of the disadvantages of the station-to-station (STS) protocol as described in Section IV is that the extension to larger systems requires larger storage for certificates and more bandwidth for verification of the signature as the number of users increases. ECC based authentication and key agreement protocol using Diffie–Hellman mechanism can provide identity authentication, key validation and user anonymity. A named certificate authority is needed in these systems and the user is required to process certificates. The SAKA (Simple Authenticated Key Agreement) algorithm uses a pre-shared secret password to ensure identity authentication, however, it is still susceptible to man-in-the-middle attack if the password is compromised. Aifen et al., combined the advantages of the two protocols to develop the Elliptic Curve Authenticated Key Agreement (ECAKA) to foil man-in-the-middle attacks [9]. The protocol uses the following steps:

B.1 Key establishment

- A chooses a random integer $d_A \in [1, n-1]$, computes $Q_A = (d_A + t)P$, and transmits Q_A to B.
- B chooses a random integer $d_B \in [1, n-1]$, computes $Q_B = (d_B + t)P$, and transmits Q_B to A.
- A computes $X = Q_B + (-t)P = d_B P$ and $K_A = d_A X = d_A d_B P$.
- B computes $X = Q_A + (-t)P = d_A P$ and $K_B = d_B X = d_A d_B P$.

B.2 Key validation

- A computes $tK_A = td_A d_B P$, and transmits to B.
- B checks whether $tK_A = tK_B$ holds or not. If it does, B believes that she and A have obtained the same session key $K_A = K_B$. Therefore B can confirm the validity of Q_A. B thereby validates K_B and transmits $td_A P$ to A.
- A checks $td_A P$ and if it is correct, A believes that B has obtained the correct Q_A. A also believes that she has obtained the correct Q_B since only B knows t besides A. A therefore believes that K_A is validated.

B.3 Security Analysis

Only by knowing t, it may be possible to generate positive validation messages, however, an imposer does not know the value of t. In the Diffie–Hellman protocol, an imposer can alter the public values such as $g^a \bmod n$ or $g^b \bmod n$ with her own values. In the ECAKA protocol, when the imposer receives $Q_A = (d_A + t)P$, she cannot guess d_A and t. She also must generate $d_I P = (d_A^* + t)P$ and send it to B; B will obtain a wrong value $d_A^* d_B P$, which is impossible for the imposer to know. The imposer cannot therefore share a session key with B or A. The ECAKA protocol has excellent forward secrecy since it is difficult to compute discrete logarithm to decipher old session key. The ECAKA protocol can successfully repel replay attacks and man-in-the-middle attacks.

C. Deniable authentication protocol

Han et al., have developed a deniable authentication protocol that enables a receiver to identify the source of a received message and prevent a third party from identifying the source of the message [10]. Their proposed protocol uses bilinear pairings over elliptic curves together with the Diffie–Hellman key exchange protocol. The proposed scheme can also be used for wireless communications. For a three party setup that includes the sender, S, the receiver, R, and an imposer, I, the deniable authentication protocol is comprised of the following:

C.1 Specifications

- Find a sufficiently large prime p such that $p \equiv 2 \bmod 3$ and $p = 6q - 1$, where q is also a large prime.

• Consider two elliptic curves E/F_p and E/F_{p^2} defined by $y^2 \equiv x_3 + 1 \bmod p$.
• Choose a secure cryptographic hash function and construct a bilinear function $e : G_1 \times G_1 \to G_2$.
• Select a generator element $P \in G_1$, therefore, $e(P, P)$ is the generator element of G_2.
• The certificate authority CEA chooses $Q \in G_1$ as one public parameter satisfying $Q = fP \in G_1$.
• Conventionally available public key digital signature scheme is used. The receiver obtains the public key of the sender from the CEA and verifies its validity.

C.2 Protocol Implementation

• S randomly chooses a number x and computes $X = xP \in G_1$ and $X' = E_{K_{Prv}}(X)$ and sends X' to R. Note that K_{Prv} is the private key of S.
• R chooses a number $y \in Z_q^*$ randomly and sends $Y = yP$ to S.
• R decrypts X' and gets $X = E_{k_{Pub}}(X')$, and then computes $k = e(Q, X)^y \in G_2$.
• S computes $k' = e(Y, Q)^x \in G_2$.
• S sends a message m with a hash message authentication code $hmac' = H(k', m) \in G$ to R.
• R computes $hmac' = H(k, m) \in G_1$. If $hmac' = hmac$, then R accepts m, otherwise R can reject it.

This protocol can also be used in the absence of a trusted center by the sender and receiver deciding Q and k_{Pub} values through another identification protocol. The proposed protocol has the *deniable property* since it is possible to design a simulator such that the hash code is indistinguishable to a third party thereby protecting the identity of the sender. Due to the difficulty of the Bilinear Diffie–Hellman problem, an imposer cannot compute $k = e(P, P)^{fxy} \in G_2$ even if she obtains X and Y by interception. This allows sender authentication by using the hash function. For perpetrating a man-in-the-middle attack, the imposer must produce key X', which is a hard problem even if the imposer knows X and P. The deniable authentication protocol therefore can successfully withstand man-in-the-middle attacks.

VI. Areas for Further Work

A. Group Key Generation

Tree based group key agreement protocols such as Diffie–Hellman involve unnecessary delays because members with low-performance computer systems can join group key computation. These delays are caused by the computations needed to balance a key tree after membership changes. An alternate approach to group key generation that can reduce delays is needed. A dynamic prioritizing mechanism of filtering low performance members in group key generation can be used reduce the computational overhead.

B. Hardness of Discrete Logarithm Problems

The Diffie–Hellman key exchange scheme is secure provided the Diffie-Hellman problem is hard. If one can solve the discrete logarithm problem, then it is clear that one can solve the Diffie-Hellman problem, hence the latter problem is no harder than the former. It is believed that the two problems are equivalent, and in fact this equivalence has been established for some special cases. One method to determine the hardness of solving the Diffie–Hellman instances is to pose a special case of the Diffie-Hellman problem called the static Diffie–Hellman problem for an arbitrary group element. Algorithms must be developed to solve the static problem to determine the weakness of the discrete logarithm problem to attacks devised to find the private key.

VII. Conclusions

The Diffie-Hellman key agreement was invented in 1976 by Whitfield Diffie and Martin Hellman and was the first practical method for establishing a shared secret over an unprotected communications channel. Their work was influenced by Ralph Merkle's work on public key distribution. The Diffie-Hellman key exchange's security depends on the difficulty of solving the discrete logarithm problem. Computing the discrete logarithm of a number modulo is difficult since it takes roughly the same amount of time as factoring the product of two primes. The security of the Diffie-Hellman key exchange can be further bolstered by utilizing elliptic curve cryptography, which makes the discrete logarithm problem almost impossible to solve. The Diffie-Hellman key exchange is, however, vulnerable to the man-in-the-middle attack. Use of digital signatures can prevent such attacks. We have discussed a few solutions to prevent man-in-the-middle attacks on elliptic curve Diffie-Hellman protocol, including ECC with no public point, authenticated key agreement with pre-shared password and deniable authentication protocol.

References

[1] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Trans. Inf. Theory*, vol. IT-22, pp. 644–654, 1976.
[2] "Diffie-Hellman Key Exchange - A Non-Mathematicians Explanation," http://www.netip.com/articles/keith/diffie-helman.htm, May 2010.
[3] "What is diffie-hellman?" http://www.rsa.com/rsalabs/node.asp?id=2248, May 2010.

[4] M. Saeki, "Elliptic curve cryptosystems," Master's thesis, McGill University, Montreal, QC, Canada, 1997.
[5] J. A. Buchmann, *Introduction to Cryptography*, Springer, 2004.
[6] M. J. W. W. Diffie, P. C. van Oorschot, "Authentication and authenticated key exchanges," *Designs, Codes and Cryptography*, vol. 2, pp. 107–125, 1992.
[7] A. Arazi, "Integrating a key cryptosystem into the digital signature standard," *Electron. Lett.*, vol. 29, no. 11, pp. 966–967, 1993.
[8] K. Kaabneh and H. Al-Bdour, "Key exchange protocol in elliptic curve cryptography with no public point," *Am. J. App. Sci.*, vol. 2, no. 8, pp. 1232–1235, 2005.
[9] Y. Y. S. Aifen, L. C. K. Hui and K. P. Chow, "Elliptic curve cryptography based authenticated key agreement with pre-shared password," *J. Electron.*, vol. 22, no. 3, pp. 268–272, 2005.
[10] W. L. S. Han and E. Chang, "Deniable authentication protocol resisting man-in-the-middle attack," *World Academy of Science, Eng. Technol.*, vol. 1, no. 3, pp. 161–164, 2005.