Global Information Assurance Certification Paper

Diffie-Hellman Public Key Distribution Scheme: A Complete Overview
Hamid Pirooz
December 4, 2000

Introduction

The transition from paper to electronic media brings with it the need for electronic privacy and authenticity. Public-key cryptography and digital signatures offer fundamental technology addressing this need without the need to share a secret decryption key. Many alternative public-key techniques have been proposed, each with its own benefits. However, there has been no single, comprehensive reference defining a full range of common public-key techniques covering key agreement, public-key encryption, digital signatures, and identification from several families, such as discrete logarithms, integer factorization, and elliptic curves.

Diffie-Hellman was the first public key cryptographic technique published. It is primarily used for public key exchange for use of some other private key type crypto system. If you are involved in any sort of Virtual Private Network (VPN), you are probably using Diffie-Hellman. In fact, if that VPN is operating on the IPSec standard, then Diffie-Hellman is certainly in use. The standards trail for key management in IPSec begins with the overall framework called Internet Security Association and Key Management Protocol . Within that framework is the Internet Key Exchange (IKE) protocol . IKE relies on yet another protocol known as OAKLEY, which uses Diffie-Hellman. It is a long trail to follow, but the result is that Diffie-Hellman is, indeed, a part of the IPSec standard.

History

The Diffie-Hellman key agreement protocol (also called exponential key agreement) was developed by Whitfield Diffie and Martin Hellman in 1976 and published in the ground-breaking paper "New Directions in Cryptography." The protocol allows two users to exchange a secret key over an insecure medium without any prior secrets.

The Diffie-Hellman algorithm was the first system to utilize "public-key" or "asymmetric" cryptographic keys. (Note: Evidence shows that it was previously invented within GCHQ in Britain, this time by Malcolm Williamson in 1974. Also, Comm-Electronics Security Group (an arm of the UK government) may have invented the concept of asymmetric key a few years before D-H. The CESG papers were classified for 20 years, but Diffie-Hellman figured it out on their own without the information in those papers.) These systems overcome the difficulties of "private-key" or "symmetric" key systems because key management is much easier.

Methodology and Process

Diffie-Hellman is not an encryption mechanism as we normally think of them, in that we do not typically use it 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" - DES, Triple DES, CAST, IDEA, Blowfish, etc.) for secure transmission.

In a symmetric key system, both sides of the communication must have identical keys. Securely exchanging those keys has always been an enormous issue. Businesses simply do not want to mess with that.

The basis for the technique is the difficulty of calculating logs in modular arithmetic.

Asymmetric key systems alleviate that issue because they use two keys - one called the "private key" that the user keeps secret and one called the "public key" that can be shared with the world. Unfortunately, the advantages of asymmetric key systems are overshadowed by speed - they are extremely slow for any sort of bulk encryption. Today, it is typical practice to use a symmetric system to encrypt the data and an asymmetric system to encrypt the symmetric keys. That is precisely what Diffie-Hellman is capable of doing - and does do when used for key exchange as described here.

The process begins when each side of the communication generates a private key. Each side then generates a public key, which is a derivative of the private key. The two systems then exchange their public keys. Each side of the communication now has its own private key and the other system's public key.

Once the key exchange is complete, the process continues. An important feature of the Diffie-Hellman protocol

is its ability to generate "shared secrets" - an identical cryptographic key shared by each side of the communication. By running the mathematical operation against your own private key and the other side's public key, you generate a value. When the distant end runs the same operation against your public key and their own private key, they also generate a value. The important point is that the two values generated are identical.

At this point, the Diffie-Hellman operation could be considered complete. The shared secret is, after all, a cryptographic key that could encrypt traffic. That is very rare, however, the reason being that the shared secret is, by its mathematical nature, an asymmetric key. As with all asymmetric key systems, it is inherently slow. If the two sides are passing very little traffic, the shared secret may encrypt actual data. Any attempt at bulk traffic encryption requires a symmetric key system such as DES, Triple DES, IDEA, CAST, Blowfish, etc.

In most real applications of the Diffie-Hellman protocol (IPSec in particular), the shared secret encrypts a symmetric key for one of the symmetric algorithms, transmits it securely, and the distant end decrypts it with the shared secret. Because the symmetric key is a relatively short value as compared to bulk data, the shared secret can encrypt and decrypt it very quickly. Speed is not so much of an issue with short values.

Which side of the communication actually transmits the symmetric key varies. However, it is most common for the initiator of the communication to be the one that transmits the key. It should also be pointed out that some sort of negotiation typically occurs to decide on the symmetric algorithm, mode of the algorithms, hash functions, key lengths, refresh rates, and so on. That negotiation is not a part of Diffie-Hellman, but it is an obviously important task since both sides must support the same schemes for encryption to function. This also points out why key management planning is so important - and why poor key management so often leads to failure of systems.

Noting that the public key is a derivative of the private key is important - the two keys are mathematically linked. However, in order to trust this system, you must accept that you cannot discern the private key from the public key. Because the public key is indeed public and ends up on other systems, the ability to figure out the private key from it would render the system useless. This is one area requiring trust in the mathematical experts. The fact that the very best in the world have tried for years to defeat this and failed bolsters our confidence a great deal.

Diffie-Hellman Algorithm Overview

The protocol has two system parameters p and g. They are both public and may be used by all the users in a system. Parameter p is a prime number and parameter g (usually called a generator) is an integer less than p, with the following 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 proceed 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 set of integers {1, ?, p-2}. Then they derive their public values using parameters p and g and their private values.

Alice's public value is $g^a \bmod p$ and Bob's public value is $g^b \bmod p$. They then exchange their public values. Finally, Alice computes $g^{ab} = (g^b)^a \bmod p$, and Bob computes $g^{ba} = (g^a)^b \bmod p$. Since $g^{ab} = g^{ba} = k$, Alice and Bob now have a shared secret key k.

The protocol depends on the discrete logarithm problem (all of the fast algorithms known for computing discrete logarithms modulo p, where p is a large prime, are forms of the index-calculus algorithm) for its security. It assumes that it is computationally infeasible to calculate the shared secret key $k = g^{ab} \bmod p$ given the two public values $g^a \bmod p$ and $g^b \bmod p$ when the prime p is sufficiently large. Maurer has shown that breaking the Diffie-Hellman protocol is equivalent to computing discrete logarithms under certain assumptions.

The algorithm is based on exponentiation in a finite (Galois) field, either over integers modulo a prime, or a polynomial field

- nb exponentiation takes $O((\log n)^3)$ operations

its security relies on the difficulty of computing logarithms in these fields

- nb discrete logarithms takes $O(e^{\log n \log \log n})$ operations

Diffie-Hellman PKDS works as follows:

- two people A & B, who wish to exchange some key over an insecure communications channel can:
- select a large prime p (~200 digit), and
- [[alpha]] a primitive element mod p
- A has a secret number x_A
- B has a secret number x_B
- A and B compute y_A and y_B respectively, which are then made public
  - $y_A = [[alpha]]^{xA} \bmod p$ $y_B = [[alpha]]^{xB} \bmod p$
- the key is then calculated as
  - $K_{AB} = [[alpha]]^{xA \cdot xB} \bmod p$
  - $= y_A^{xB} \bmod p$ (which **B** can compute)
  - $= y_B^{xA} \bmod p$ (which **A** can compute)
- and may then be used in a private-key cipher to secure communications between A and B

nb: if the same two people subsequently wish to communicate, they will have the **same** key as before, unless they change their public-key (usually not often)

The time to execute DHC scheme in software is proportional to D3, where D is the number of bits of p. Thus, going up from 200 to 800 bits raises the complexity by a factor of 64 .

Mitigating Potential Weaknesses

The Diffie-Hellman key exchange is vulnerable to a man-in-the-middle attack. In this attack, an opponent Carol intercepts Alice's public value and sends her own public value to Bob. When Bob transmits his public value, Carol substitutes it with her own and sends it to Alice. Carol and Alice thus agree on one shared key and Carol and Bob agree on another shared key. After this exchange, Carol 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. Possible solutions include the use of digital signatures and other protocol variants.

A potential danger when Diffie-Hellman is used is that one could, for some values of the modulus P, choose values of A such that A^x has only a small number of possible values, no matter what x is, which would make it easy to find for a value of x that was equivalent to the original value of x. This can be defended against by using a modulus P such that P-1 is equal to 2 times another prime number. Primes of this form are known as Sophie Germain primes, after a noted mathematician who discovered some of their remarkable properties.

The Key Exchange Algorithm (KEA) which was used with SKIPJACK in the Clipper chip, and which was declassified at the same time, uses Diffie-Hellman in an interesting manner which highlights some of its properties. As in the Digital Signature Algorithm, the prime modulus P is required to be one such that P-1 has a factor that is 160 bits long. Since P is 1024 bits long, (or 512 to 1024 bits long, in the case of the Digital Signature Standard) this appears to be less secure than the use of a Sophie Germain prime. Also, it is somewhat more cumbersome to use: where f is the 160-bit long prime factor of P-1, A cannot simply be chosen at random; instead, another number less than P-1, B, is chosen at random, and B^((P-1)/f) is used as A as long as it is not equal to 0 or 1 (modulo P, of course) and it also requires that each A^x used be tested to determine that A^(xf) is equal to 1 modulo P.

Although this may be less secure than the use of a Sophie Germain prime, it is certainly more secure than simply choosing P at random, and making no effort to avoid the problems that small factors of P-1 could cause.

The protocol involved in KEA is of more interest. The session key is derived by hashing

$$A^{(x1*y2)} + A^{(x2*y1)}$$

all calculated modulo P, where x1 and x2 came from the first party to the communication, and y1 and y2 came from the second.

The important thing about KEA is the difference between where the x values and the y values came from.

The first user retains x1 as a persistent private key, and when A^x1 is presented as his public key, it is accompanied by a digital certificate. Similarly, the second user has a digital certificate for A^y1 as a public key.

On the other hand, x2 and y2 were random numbers generated at the time of the communication, and thus A^x2 and A^y2 do not have certificates corresponding to them. Thus, by involving the four parameters, a persistent and a temporary key from each user in the procedure, both users prove their identity with a digital certificate, but the resulting session key is something that is different for every message.

Since the x2 and y2 values are not persistent (they are sometimes called nonces), an attacker could only obtain them through access to the computers or other equipment of the parties to the communication at about the same time as the message with which they are associated is itself present in plaintext form on that equipment. Thus, unlike persistent private keys, nonces do not contribute to the security burden of key storage. This means that a passive attacker would need to compromise the persistent private keys of both parties (if that attacker could not also obtain one of the nonces) to read a message whose key was generated through KEA. This, however, was not likely to have been a major design consideration, because additional security against passive attacks could be gained by making the protocol more elaborate (for example, A^(x2*y2) could be used in addition in the generation of the session key); but this by itself would not increase security against an active attack. Although this technique could be combined with other measures that do combat active attacks, an attacker who could also partially compromise keys is not only likely to be able to mount an active attack of the "man-in-the-middle" type, but may even be able to tamper with the encryption hardware or software being used.

Authenticated Diffie-Hellman Key Agreement

It is not common, but the ability does exist with the Diffie-Hellman protocol to have a Certificate Authority certify that the public key is indeed coming from the source you think it is. The purpose of this certification is to prevent Man In the Middle (MIM) attacks. The attack consists of someone intercepting both public keys and forwarding bogus public keys of their own. The "man in the middle" potentially intercepts encrypted traffic, decrypts it, copies or modifies it, re-encrypts it with the bogus key, and forwards it on to its destination. If successful, the parties on each end would have no idea that there is an unauthorized intermediary. It is an extremely difficult attack to pull off outside the laboratory, but it is indeed possible. Properly implemented Certificate Authority systems have the potential to disable the attack.

The authenticated Diffie-Hellman key agreement protocol, or Station-to-Station (STS) protocol, was developed by Diffie, van Oorschot, and Wiener in 1992 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.

Roughly speaking, 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.

Conclusion

Once secure exchange of the symmetric key is complete (and note that passing that key is the whole point of the Diffie-Hellman operation), data encryption and secure communication can occur. Note that changing the symmetric key for increased security is simple at this point. The longer a symmetric key is in use, the easier it is to perform a successful cryptanalytic attack against it. Therefore, changing keys frequently is important. Both

sides of the communication still have the shared secret and it can be used to encrypt future keys at any time and any frequency desired.

The use of Diffie-Hellman greatly reduces the headache of using symmetric key systems. These systems have astounding speed benefits, but managing their keys has always been difficult, to say the very least. Because the steps we have just gone through happen in a matter of a second or two and are completely transparent to the user, ease of use could not be better…, provided that it works. The good news is that it almost always does work. Understanding the underlying protocol only becomes necessary in the rare case that it doesn't.

In recent years, the original Diffie-Hellman protocol has been understood to be an example of a much more general cryptographic technique, the common element being the derivation of a shared secret value (that is, key) from one party's public key and another party's private key. The parties' key pairs may be generated anew at each run of the protocol, as in the original Diffie-Hellman protocol. The public keys may be certified, so that the parties can be authenticated and there may be a combination of these attributes. The draft ANSI X9.42 illustrates some of these combinations, and a recent paper by Blake-Wilson, Johnson, and Menezes provides some relevant security proofs.

References

The IEEE P1363 Home page
IEEE P1363: Standard Specifications For Public Key Cryptography (12/1/00)

Number Theory and Public Key Cryptography
URL: http://www.cs.adfa.edu.au/teaching/studinfo/csc/lectures/publickey.html (12/1/00)

Network Security, C. Kaufman, R. Perlman, M Speciner, Prentice-hall 1995. (12/2/00)

What is Diffie-Hellman?
www.rsasecurity.com/rsalabs/faq/3-6-1.html (12/3/00)

Diffie-Hellman Key Exchange, by Keith Palmgren

Security portal.com/topnews/dhkeyexchange2000706.html (12/1/00)

"RSA Data Security, Inc. Public-Key Cryptography Standards (PKCS)" #3: Diffie-Hellman Key Agreement Standard. Which is based upon W. Diffie and M.E. Hellman's *New directions in cryptography* from IEEE transactions on Information Theory, IT 22:644-654, 1976.

PKCS-3 is available for anonymous FTP from ftp://ftp.rsa.com/ in /pub/pkcs/ps/pkcs-3.ps (or /pub/pkcs/ascii/pkcs-3.asc). (12/1/00)

Encription I, Harish Bhatt, SANS' GIAC, March 2000. (12/1/00)

The Diffie-Hellman Key Agreement patent (U.S. Patent 4,200,770) was owned by Public Key Partners. It expired (9/6/1997). (12/1/00)

Security Needs On Networks, 6.805 Lecture, Sept. 28, 1999
URL: http://www-swiss.ai.mit.edu/6095/admin/admin-1999/weeks/crypto-lecture/sld008.htm (12/3/00)

W Diffie, M E Hellman, "Privacy and Authentication: An Introduction to Cryptography", Proc. of the IEEE, Vol 67 No 3, pp 397-427, Mar 1979 (12/2/00)

Last Updated: May 27th, 2017

Upcoming Training

SANS Stockholm 2017	Stockholm, Sweden	May 29, 2017 - Jun 03, 2017	Live Event
SANS San Francisco Summer 2017	San Francisco, CA	Jun 05, 2017 - Jun 10, 2017	Live Event
Security Operations Center Summit & Training	Washington, DC	Jun 05, 2017 - Jun 12, 2017	Live Event
SANS Houston 2017	Houston, TX	Jun 05, 2017 - Jun 10, 2017	Live Event
Community SANS Ottawa SEC401	Ottawa, ON	Jun 05, 2017 - Jun 10, 2017	Community SANS
SANS Charlotte 2017	Charlotte, NC	Jun 12, 2017 - Jun 17, 2017	Live Event
SANS Rocky Mountain 2017 - SEC401: Security Essentials Bootcamp Style	Denver, CO	Jun 12, 2017 - Jun 17, 2017	vLive
SANS Secure Europe 2017	Amsterdam, Netherlands	Jun 12, 2017 - Jun 20, 2017	Live Event
Community SANS Portland SEC401	Portland, OR	Jun 12, 2017 - Jun 17, 2017	Community SANS
SANS Rocky Mountain 2017	Denver, CO	Jun 12, 2017 - Jun 17, 2017	Live Event
SANS Minneapolis 2017	Minneapolis, MN	Jun 19, 2017 - Jun 24, 2017	Live Event
SANS Columbia, MD 2017	Columbia, MD	Jun 26, 2017 - Jul 01, 2017	Live Event
SANS Cyber Defence Canberra 2017	Canberra, Australia	Jun 26, 2017 - Jul 08, 2017	Live Event
SANS Paris 2017	Paris, France	Jun 26, 2017 - Jul 01, 2017	Live Event
SANS London July 2017	London, United Kingdom	Jul 03, 2017 - Jul 08, 2017	Live Event
Cyber Defence Japan 2017	Tokyo, Japan	Jul 05, 2017 - Jul 15, 2017	Live Event
Community SANS Minneapolis SEC401	Minneapolis, MN	Jul 10, 2017 - Jul 15, 2017	Community SANS
SANS Los Angeles - Long Beach 2017	Long Beach, CA	Jul 10, 2017 - Jul 15, 2017	Live Event
Community SANS Phoenix SEC401	Phoenix, AZ	Jul 10, 2017 - Jul 15, 2017	Community SANS
SANS Munich Summer 2017	Munich, Germany	Jul 10, 2017 - Jul 15, 2017	Live Event
SANS Cyber Defence Singapore 2017	Singapore, Singapore	Jul 10, 2017 - Jul 15, 2017	Live Event
Mentor Session - SEC401	Macon, GA	Jul 12, 2017 - Aug 23, 2017	Mentor
Mentor Session - SEC401	Ventura, CA	Jul 12, 2017 - Sep 13, 2017	Mentor
Community SANS Atlanta SEC401	Atlanta, GA	Jul 17, 2017 - Jul 22, 2017	Community SANS
Community SANS Colorado Springs SEC401	Colorado Springs, CO	Jul 17, 2017 - Jul 22, 2017	Community SANS
SANSFIRE 2017	Washington, DC	Jul 22, 2017 - Jul 29, 2017	Live Event
SANSFIRE 2017 - SEC401: Security Essentials Bootcamp Style	Washington, DC	Jul 24, 2017 - Jul 29, 2017	vLive
Community SANS Charleston SEC401	Charleston, SC	Jul 24, 2017 - Jul 29, 2017	Community SANS
Community SANS Fort Lauderdale SEC401	Fort Lauderdale, FL	Jul 31, 2017 - Aug 05, 2017	Community SANS
SANS San Antonio 2017	San Antonio, TX	Aug 06, 2017 - Aug 11, 2017	Live Event
SANS Prague 2017	Prague, Czech Republic	Aug 07, 2017 - Aug 12, 2017	Live Event