

Network Storage Group Host and Switch Products Technology Brief

January 24, 2003

SAN SECURITY TERMINOLOGY

Security for Fibre Channel SANs is increasingly a topic of discussion. Standards from T11 are expected to occur in 1H03, with standards based products by the end of the year. It is therefore important we are familiar with SAN Security terminology. This Tech Brief provides definitions and explanations of the most common SAN Security terms and acronyms, in alphabetical order. It also provides sources of information so that you may do additional research if you wish.

Term / Acronym	Definition
3DES	3DES (read Triple DES) is an acronym that stands for Stands for Triple Data Encryption Standard, a more secure version of the DES encryption method. 3DES is a 168-bit encryption standard. In 3DES, the text is encoded three times, as opposed to just one. There are several variants of this mechanism. EEE3 method uses three different keys to encrypt a message three times. EDE3 also uses three keys, but the message is first encrypted with the first key, then decrypted with the second one, and finally encrypted with the third. EEE2 and EDE2 work similarly, except that they use the same key as the first and third one in the above algorithm.
AES	The Advanced Encryption Standard (AES) is an updated encryption standard replacing DES.
Authentication	Basic authentication is a standard which nearly all browsers and applications support. When you access a site or application and see a standard popup window which asks for your username and password, your are using basic authentication.
CAST	CAST is an encryption technique. The CAST algorithm supports variable key lengths, anywhere from 40 bits to 128 bits in length. This ensures that an appropriate security level is given to data for the intended purpose and enables seamless interoperation with exportable versions of products, where necessary. CAST uses a 64-bit block size which is the same as the Data Encryption Standard (DES), making it a suitable drop-in replacement. CAST has been shown to be two to three times faster than a typical implementation of DES and six to nine times faster than a typical implementation of 3DES. CAST is implemented in products from Pretty Good Privacy Inc., IBM, Tandem and Microsoft.
CHAP	CHAP is an acronym that stands for Challenge Handshake Authentication Protocol. It is used for authentication. A complete description of CHAP is included as Attachment A.
DES	DES is an acronym that stands for Data Encryption Standard. It is an encryption technique. DES, was the first official U.S. government cipher intended for commercial use. DES is the most widely used cryptosystem in the world. A complete description of DH is included as Attachment E.
DH	DH is an acronym that stands for Diffie-Hellman. It is used for authentication. A complete description of DH is included as Attachment B.

DH-CHAP	DH-CHAP is an acronym that stands for Diffie-Hellman - Challenge Handshake Authentication Protocol.
Encryption	A technique to secure data. Also see Attachment F.
FC-AP	FC-AP is an acronym that stands for Fibre Channel - Authentication Protocol. It is used for authentication.
IKE	Internet Key Exchange
Private key	Used in asymmetrical encryption schemas to protect communications streams.
Public Key	Used in asymmetrical encryption schemas to protect communications streams.
Replay Prevention	A previously legitimate message, such as the request to delete a file, can potentially cause significant harm if repeated even a few minutes later. Network authentication systems must prevent the replay of old communications to avoid this problem. A complete description of DH is included as Attachment D.
SRP	SRP is an acronym that stands for Secure Remote Password. It is used for authentication. A complete description of DH is included as Attachment C.
VPN	Virtual Private Network

Attachment A Challenge Handshake Authentication Protocol

Source: <http://www.freesoft.org/CIE/RFC/1334/9.htm>

The Challenge-Handshake Authentication Protocol (CHAP) is used to periodically verify the identity of the peer using a 3-way handshake. This is done upon initial link establishment, and MAY be repeated anytime after the link has been established.

After the Link Establishment phase is complete, the authenticator sends a "challenge" message to the peer. The peer responds with a value calculated using a "one-way hash" function. The authenticator checks the response against its own calculation of the expected hash value. If the values match, the authentication is acknowledged; otherwise the connection SHOULD be terminated.

CHAP provides protection against playback attack through the use of an incrementally changing identifier and a variable challenge value. The use of repeated challenges is intended to limit the time of exposure to any single attack. The authenticator is in control of the frequency and timing of the challenges.

This authentication method depends upon a "secret" known only to the authenticator and that peer. The secret is not sent over the link. This method is most likely used where the same secret is easily accessed from both ends of the link.

Implementation Note: CHAP requires that the secret be available in plaintext form. To avoid sending the secret over other links in the network, it is recommended that the challenge and response values be examined at a central server, rather than each network access server. Otherwise, the secret SHOULD be sent to such servers in a reversably encrypted form.

The CHAP algorithm requires that the length of the secret MUST be at least 1 octet. The secret SHOULD be at least as large and unguessable as a well-chosen password. It is preferred that the secret be at least the length of the hash value for the hashing algorithm chosen (16 octets for MD5). This is to ensure a sufficiently large range for the secret to provide protection against exhaustive search attacks.

The one-way hash algorithm is chosen such that it is computationally infeasible to determine the secret from the known challenge and response values.

The challenge value SHOULD satisfy two criteria: uniqueness and unpredictability. Each challenge value SHOULD be unique, since repetition of a challenge value in conjunction with the same secret would permit an attacker to reply with a previously intercepted response. Since it is expected that the same secret MAY be used to authenticate with servers in disparate geographic regions, the challenge SHOULD exhibit global and temporal uniqueness. Each challenge value SHOULD also be unpredictable, least an attacker trick a peer into responding to a predicted future challenge, and then use the response to masquerade as that peer to an authenticator. Although protocols such as CHAP are incapable of protecting against realtime active wiretapping attacks, generation of unique unpredictable challenges can protect against a wide range of active attacks.

Attachment B

Diffie-Hellman

Source: <http://rr.sans.org/encryption/diffie.php>

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 $n = g^k \text{ mod } p$.

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 $\{1, \dots, p-2\}$. 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

o $y_A = [\alpha]^{x_A} \bmod p$ $y_B = [\alpha]^{x_B} \bmod p$

?? the key is then calculated as

o $K_{AB} = [\alpha]^{x_A \cdot x_B} \bmod p$

o $= y_A^{x_B} \bmod p$ (which **B** can compute)

o $= y_B^{x_A} \bmod p$ (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 D^3 , 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

$$(x_1^{y_2}) (x_2^{y_1})$$

$$A + A$$

all calculated modulo P , where x_1 and x_2 came from the first party to the communication, and y_1 and y_2 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 x_1 as a persistent private key, and when A^{x_1} is presented as his public key, it is accompanied by a digital certificate. Similarly, the second user has a digital certificate for A^{y_1} as a public key.

On the other hand, x_2 and y_2 were random numbers generated at the time of the communication, and thus A^{x_2} and A^{y_2} 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 x_2 and y_2 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, $A^{(x_2*y_2)}$ 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 $g^a \text{ mod } p$. 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](http://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)

Attachment C Secure Remote Password

Source: <http://srp.stanford.edu/whatisit.html>

The **Secure Remote Password** protocol is the core technology behind the Stanford SRP Authentication Project. The Project is an Open Source initiative that integrates secure password authentication into existing networked applications.

The Project's primary purpose is to improve network security from the ground up - by integrating strong password authentication into widely-used protocols instead of adding security as an afterthought. SRP makes these objectives possible because it offers a unique combination of password security, user convenience, and freedom from restrictive licenses.

This site serves as the semi-official home of the SRP distribution, which contains secure versions of **Telnet** and **FTP**. In addition, it contains links to a number of SRP-related projects, products (both commercial and non-commercial), and research on the Web.

SRP is a secure password-based authentication and key-exchange protocol. It solves the problem of authenticating clients to servers securely, in cases where the user of the client software must memorize a small secret (like a password) and carries no other secret information, and where the server carries a *verifier* for each user, which allows it to authenticate the client but which, if compromised, would not allow the attacker to impersonate the client. In addition, SRP exchanges a cryptographically-strong secret as a byproduct of successful authentication, which enables the two parties to communicate securely.

Many password authentication solutions claim to solve this exact problem, and new ones are constantly being proposed. Although one can claim security by devising a protocol that avoids sending the plaintext password unencrypted, it is much more difficult to devise a protocol that remains secure when:

- ?? Attackers have complete knowledge of the protocol.
- ?? Attackers have access to a large dictionary of commonly used passwords.
- ?? Attackers can eavesdrop on all communications between client and server.
- ?? Attackers can intercept, modify, and forge arbitrary messages between client and server.
- ?? A mutually trusted third party is not available.

The idea behind SRP first appeared on USENET in late 1996, and subsequent discussion led to refined proposals in 1997 to address these security properties. The variant in use today, known as SRP-3, was published in 1998 after several rounds of discussion and refinement on cryptography-related newsgroups and mailing lists, and has withstood considerable public analysis and scrutiny since then. Technical details of the actual [protocol design](#) are available from this site.

SRP is available to commercial and non-commercial users under a royalty-free [license](#). The Internet played a significant role in SRP's early development; without it, SRP would not have received anywhere near the amount of analysis and feedback that it has gotten since it was first proposed and refined. It is thus fitting that the Internet at large can benefit from the fruits of this endeavor. Since SRP also works around existing patents in the area, it gives everybody access to strong, unencumbered password authentication technology that can be put to a variety of uses.

The SRP distribution is available under Open Source-friendly licensing terms (for the net.savvy reader, it's a "BSD-style" license). More information about the [SRP project](#) is available at this site, and a [reference implementation](#), which includes versions of Telnet and FTP that incorporate SRP support, can be downloaded as well. The [links page](#) has pointers to a wide range of projects and products, both commercial and non-commercial, that use SRP, as well as related work and papers that cover strong password authentication.

Attachment D Replay Prevention in Authentication Systems

Source: <http://web.mit.edu/6.033/1998/www/reports/r05-dm.html>

Network authentication systems must prevent the replay of old communications. A previously legitimate message, such as the request to delete a file, can potentially cause significant harm if repeated even a few minutes later.

There are two ways to prevent replay: First, secure network protocols can require the exchange of freshly generated random strings, or *nonces*. Old messages cannot contain subsequently generated random data, and will therefore contain the wrong nonces if replayed. Alternatively, servers can simply record every message they receive in a *replay cache*, and discard any duplicates. To prevent replay caches from having to grow without bounds, messages should include timestamps. If servers automatically reject messages with old timestamps, they can also evict such messages from the replay cache. However, relying on timestamps in this way has the added complication of requiring all machines to have loosely synchronized clocks.

The main advantage of replay caches is that they allow stateless protocols. If, for instance, a client wishes to make an RPC to a server, it can send a single message containing a valid timestamp and immediately receive a reply. Thus, an RPC need only incur the overhead of a single network round trip, and servers don't have to remember clients between RPCs. With nonces, in contrast, a client must first request a nonce from the server, then send the server an RPC containing that nonce. The cost of fetching a nonce can be amortized over multiple RPCs, but this requires the server to remember the latest nonce of each client. Such per-client state can waste quite a bit of space when large numbers of clients each make RPCs infrequently.

Replay caches and timestamps have some disadvantages, however. First, security depends on clocks remaining roughly synchronized. Whatever mechanism sets the system clock must therefore be a trusted part of the authentication system. Second, the task of maintaining a coherent replay cache across multiple processes and programs can be rather tricky, particularly if a server can crash and reboot in less time than the clock tolerance. Indeed, the Kerberos authentication system uses protocols with timestamps rather than nonces. Yet, 10~years after deployment, most Kerberos implementations still don't have a working replay cache. Could such a crucial part of a widely used system really go unimplemented for over a decade if there weren't serious complications involved?

In summary, replay caches can save secure network servers from the need to keep per-client state. Unfortunately, this statelessness comes at the cost of trusting the system clock. Moreover, experience has shown replay caches difficult to implement. A better alternative for most purposes is to exchange nonces in authentication protocols.

Attachment E Data Encryption Standard

Source: <http://www.rsasecurity.com/rsalabs/faq/3-2-1.html>

DES, an acronym for the Data Encryption Standard, is the name of the Federal Information Processing Standard (FIPS) 46-3, which describes the data encryption algorithm (DEA). The DEA is also defined in the ANSI standard X9.32.

DEA is an improvement of the algorithm Lucifer developed by IBM in the early 1970s. While the algorithm was essentially designed by IBM, the NSA (see Question [6.2.2](#)) and NBS (now NIST; see Question [6.2.1](#)) played a substantial role in the final stages of the development. The DEA, often called DES, has been extensively studied since its publication and is the best known and widely used symmetric algorithm in the world.

The DEA has a 64-bit block size (see Question [2.1.4](#)) and uses a 56-bit key during execution (8 parity bits are stripped off from the full 64-bit key). The DEA is a symmetric cryptosystem, specifically a 16-round Feistel cipher (see Question [2.1.4](#)) and was originally designed for implementation in hardware. When used for communication, both sender and receiver must know the same secret key, which can be used to encrypt and decrypt the message, or to generate and verify a message authentication code (MAC). The DEA can also be used for single-user encryption, such as to store files on a hard disk in encrypted form. In a multi-user environment, secure key distribution may be difficult; public-key cryptography provides an ideal solution to this problem (see Question [2.1.3](#)).

NIST (see Question [6.2.1](#)) has re-certified DES (FIPS 46-1, 46-2, 46-3) every five years. FIPS 46-3 reaffirms DES usage as of October 1999, but single DES is permitted only for legacy systems. FIPS 46-3 includes a definition of triple-DES (TDEA, corresponding to X9.52); TDEA is "the FIPS approved symmetric algorithm of choice." Within a few years, DES and triple-DES will be replaced with the Advanced Encryption Standard (AES, see Section [3.3](#)).

Attachment F Encryption

Source: <http://hotwired.lycos.com/webmonkey/programming/php/tutorials/tutorial1.html>

Encryption Tutorial Overview (by [Julie Meloni](#))

Crypto," to use the all-purpose abbreviation for cryptography, cryptanalysis, and cryptology, is cool. Just plain cool. My biggest regret in life is that I never took a math class past Algebra II, so I really don't know jack about the mathematical foundations of intense crypto systems. But boy, do I respect those who do.

If you're a person who finds crypto textbooks really boring yet wants to understand this whole crypto bit in the broad sense, go read Neal Stephenson's [Cryptonomicon](#). Sure, it's more than 900 pages of quasi-fiction, but it manages to tell a fascinating story while giving an incredible amount of insight into modern cryptography.

In this tutorial, you'll learn something or another about the common, Web-based uses for the following basic encryption techniques

Asymmetric key-based algorithms. This method uses one key to encrypt data and a different key to decrypt the same data. You have likely heard of this technique; it is sometimes called public key/private key encryption, or something to that effect.

Symmetric key-based algorithms, or block-and-stream ciphers. Using these cipher types, your data is separated into chunks, and those chunks are encrypted and decrypted based on a specific key. Stream ciphers are used more predominantly than block ciphers, as the chunks are encrypted on a bit-by-bit basis. This process is much smaller and faster than encrypting larger (block) chunks of data.

Hashing, or creating a digital summary of a string or file. This is the most common way to store passwords on a system, as the passwords aren't really what's stored, just a hash that can't be decrypted.

If your head's already spinning, stick with me — it does get better. The following sections will show you the why and how of real-life data encryption in a Web environment, using PHP and various other tools such as the mcrypt and mhash libraries.

Get started: [Lesson 1](#) »