Web Security

HTTPS

Normal web transactions - request to a web server for a page and its response - are executed by *HTTP* (Hyper Text Transfer Protocol). For more secure transactions that involve sensitive information like credit card numbers *HTTPS* is used. HTTPS is not a new protocol. It is the same Hyper Text Transfer Protocol but uses a Secure Socket Layer (SSL).

Standard HTTP uses the default port of 80, whereas HTTPS uses the default port of 443. HTTPS encrypts and decrypts the page requests and page response between the client browser and web server using the Secure Socket Layer. The encryption or decryption is based on a key that is either 40 bits or 128 bits. The higher number is indicative of greater security.

Most web browsers can communicate via either HTTP or HTTPS. When the browser connects to a secure web site it will display a padlock icon in the address bar when it receives a SSL certificate from the web server. The browser will also display the word *https* in the url.

To become a secure web site, the system administrator of the web server must first obtain a X.509 Digital Certificate from a trusted Certificate Authority (like VeriSign, Comodo, GoDaddy). The digital certificate authenticates the web site. When a browser makes a request the web server sends a copy of the certificate. It will have the digital signature of VeriSign and can be verified.

Along with the certificate, the web server will send its public key. The public key is used by the browser to encrypt messages that are sent to the server. The web server will decrypt messages that it receives using its private key.

Public / Private Key Cryptography

There are several ways in which sensitive information can be sent across an insecure network using - symmetric key or asymmetric keys. An analogy works best to explain the difference between the two methods. In the *symmetric key* method the same key is used to both encrypt and decrypt. Supposing that A wants to send a secret message to B. Then A can put the secret message in a box and put a padlock on that box with his key (encryption). He then sends the box through regular mail to B. B has a copy of the key that A used and when B receives the box he can open it with his key and read the message (decryption). The problem with this method is how to give B a copy of the key that A used.

In the *asymmetric key* method there are two keys - a public key and a private key. In the above scenario A wants to send a secret message to B and he lets B know that. So B sends A an open padlock. A then puts that padlock on the box containing the secret message and sends the box by regular mail. When B receives it, he can open the padlock since only he has the key. In other words, B sends A his public key. A encrypts the message with this public key and sends it to B. B decrypts the message with his private key. The method works because possessing the public key does not let anyone decrypt the encrypted message with that key. It can only be decrypted with the private key. The security of this method lies in how safe B can keep his private key.

There is another protocol that one can use - *three-pass protocol* - that does not involve sending keys across the network. An analogy can also help explain this method. If A wants to send a secret message to B he can send it in a box with his padlock. When B receives the box he sends it back to A with a padlock of his own. After receiving the box, A removes his padlock and returns the box to B. B can now open the box because it has only his padlock on it.

RSA

The algorithm used by the *Public / Private Key* method goes by name of RSA standing for the surnames of Ron Rivest, Adi Shamir, and Leonard Adleman that they described in 1977. Variations of this algorithm existed prior to the publication of their paper. This algorithm relies on *one way function*. A *one way function* is easy to compute but hard to invert. For example it is easy to take the product of two prime numbers but given the product it is difficult to split it into the original prime factors. The outline of the algorithm is given below:

1. Choose two large and distinct prime numbers p and q.
2. Take the product $n = p * q$
3. Compute the number of integers less than n that are coprime with n (otherwise known as the totient):$\varphi(n) = (p - 1) * (q - 1)$
4. Choose an integer e such that $1 < e < \varphi(n)$ and e and $\varphi(n)$ are coprime (i.e. share no common factors other than 1)
5. Compute a value for d such that it satisfies the relation: $(d * e) \% \varphi(n) = 1$
6. The public key is (e, n)
7. The private key is (d, n)
8. To encrypt m using the public key use the relation: $c = m^e \% n$
9. To decrypt c using the private key use the relation: $m = c^d \% n$

Here is a worked example using simple numbers.

Encryption using public keys is normally computationally intensive. So, in practice the sender encrypts the message with a secret key that is randomly generated. The secret key is encrypted using the public key of the recipient and sent with the encrypted message. The recipient decrypts the secret key using his private key and using that decrypts the rest of the message. The following lists all the steps in the process:

- The client and server go through a handshaking procedure.
- The handshake begins when the client connects to a SSL enabled server requesting a secure connection and presents a list of encryption algorithms and hash functions that it supports.
- From this list the server chooses the most secure encryption algorithm and hash function that it also supports and lets the client know about its choice.
- In the above transaction, the server also sends it identification in the form of a digital certificate. The digital certificate contains the server's name, the trusted Certificate Authority, and the server's public encryption key.
- The client may contact the trusted Certificate Authority for verification.
- The client generates a random number and encrypts it with the server's public key and sends it to the server. Only the server can decrypt this with its private key.
- The random number generated by the client is then used in the encryption and decryption process on both the client and server sides.

Digital Certificate

A digital certificate authenticates a person, organization, or computer on the internet. To be exact it verifies that the public key indeed belongs to the individual. The digital certificate is issued by a trusted Certificate Authority (CA). A certificate maybe revoked if the private key has been compromised or if the relation between the individual and organization has changed. A *certification revocation list* (CRL) is maintained that lists revoked or cancelled certificates. A certificate includes:

- The public key that is being signed
- The name of the person, organization, or computer
- Validity period
- Digital signature of the certificate produced by the CA's private key

The digital signature consists of three algorithms:

- A key generation algorithm G that randomly produces a key pair (PK, SK) for the CA. PK is for verifying and which is public and SK the signing key that is kept private by the CA.
- A signing algorithm S, that on input of a message *m* and the signing key SK, produces a signature σ.
- A signature verifying algorithm V, that on input of a message *m*, a verifying key PK, and a signature σ either accepts or rejects it.

To create the digital signature, the CA creates a cryptographic hash of the information - client's public key, identity, validity period. The hash is a shortened version of that information. The usual algorithms used are MD5 or SHA-1. The CA then encrypts the hash with his private or signing key SK. The encrypted hash is the digital signature and is appended to the message.

To verify the integrity of the message the recipient does the following things - decrypts the digital signature with CA's public key and retrieves the hash of the message. The recipient also makes a hash of the original message. If the two hashes match then the message has not been tampered with and the public key is identified.

Cryptographic Hash Function

A cryptographic hash function takes an input of any length and returns as output a fixed size string. This string is considered to be a *digital fingerprint* of the input. The two commonly used hash functions are MD5 and SHA-1. MD5 returns a string of 128 bit and SHA-1 returns a string of 160 bits. Both have been shown to have security flaws.

A desirable cryptographic hash function should have these features:

- Given the hash *h* it should be difficult to find the input *m* where *h = hash (m)*. The only way to determine *m* would be by brute force and would be so computationally and time intensive that it would not be worthwhile to do so.
- Given an input m_1 it should be hard to find another input m_2 such that $hash (m_1) = hash (m_2)$.
- A small change in the input *m* would cause a large change in the output of the hash function.

Both MD5 and SHA-1 hash functions are built with the Merkle-Damgard construction. This method takes an arbitrary sized input and breaks the input into fixed size blocks of the same size as the output. It applies a one way compression function to each of the blocks in turn, combining a block of input with the output of the previous block. The last block has bits representing the length of the entire message.

A one way compression function takes two fixed size inputs - the key and the plain text - and returns one single output - the cipher text which is the same size as the plain text. An example of such a function is the Davis-Meyer compression function. It feeds the previous hash value (H_{i-1}) as the plaintext to be encrypted. It uses the each block of the message (m_i) as the key. The output ciphertext is then XORed with the previous hash value (H_{i-1}) to produce the next hash value (H_i). In the first round when there is no previous hash value it uses a predefined inital value (H_0).