SSL, technically

Since encryption of data is one of the responsibilities of the transport layer, it's important to mention that this is where encryption such as SSL/TLS (if used) operates. This has a very useful effect which benefits the protocol greatly: Because the transport layer is responsible for establishing connections as well, a connection will simply not be established if the security checks aren't passed.

The SSL / TLS procedure consists of two steps; the first one being authentication, and the second being encryption. Because SSL/TLS is based on the use of certificates, certificates is a very vital part of the process. One key difference (pun intended) between SSL and TLS, is that SSL requires the **server** to prove its identity to the client, whereas TLS requires both parties to prove their identity.

Handshake

The first thing that needs to be done to initiate encryption is what we call the 'handshake'. This is the first of the two processes that make up the SSL/TLS protocols, and in turn consists of several steps.

The first step in the handshake process is called the negotiation phase. During this phase, each packet contains a specific *message type code* [INSERT REFERENCE], which indicates what type of packet is being sent.

First, the client sends a 'hello' to the server, this is called the *ClientHello* (message type 1). Listed within this 'hello' message is the algorithms (*Cipher Suites*) which the client supports for encryption, allowing the server to pick the strongest supported algorithm and reply with the server's equivalent to ClientHello, namely *ServerHello* (message type 2), containing a random number, the protocol the server has selected from the list of suggestions from the client, a random number, the CipherSuite [INSERT REFERENCE], and compression method selected; followed by the server's x.509 certificate (message type 11) containing its public key. It's important to note that the server should always select the strongest available protocol.

When the negotiation is complete, the server will send a message indicating that it is done, called the *ServerHelloDone* (message type 14) packet, to which the client will respond with a packet called the *ClientKeyExchange* (message type 16). This packet contains either a *PreMasterSecret* [INSERT REFERENCE] (encrypted with the public key found in the server's certificate), a public key, or nothing at all depending on the selected protocol [INSERT REFERENCE].

The random numbers contained within the *ClientHello* and *ServerHello* will be used to calculate a *MasterSecret* together with the *PreMasterSecret* in the next step. The *MasterSecret* becomes a shared secret between the client and the server, which the data required for encryption will be based on.

When this is complete, both parties are ready to start encrypting data. This begins after the client sends a *ChangeCipherSpec* [INSERT REFERENCE] packet, essentially indicated that *"Now is the time to change to the cipher suite we have agreed to use, with the shared secret we calculated",* and finally a *Finished* packet containing the *message authentication code* [INSERT REFERENCE] and hash calculated from the previous handshake messages.

Upon receiving this from the client, the server in turn responds with a similar *ChangeCipherSpec* packet indicating that it, too, is ready to start encryption of data, followed by a *Finished* (message type 20) message.

x.509 Certificate

The certificate provided by the server is of the type x.509 as specified in RFC5280 (and earlier RFC2459) and updated in RFC6818. It is issued by a Certificate Authority, which [supposedly] proves the server's identity as verified by said authority, and contains a number of fields – in this case used to verify the legitimacy of the other party, as well as generating a private key.

x.509 certificates follow a very specific structure, expressed in ASN.1 (Abstract Syntax Notation One), the structure of which we'll look closer at further down in this document.

Generating a Private Key: Diffie-Hellman

The Diffie-Hellman key exchange was published in 1976 in a ground breaking paper by Whitfield Diffie and Martin Hellman. Essentially, this algorithm is together with the DES responsible for bringing cryptography to the public eye.

The algorithm devised is a way to generate an identical shared secret without any prior secrets. The algorithm's strength is based on the *discrete logarithm problem,* making it very easy to do one way, but very difficult to reverse.

The process does, however, have some prerequisites; these being that the random number g must be lower than the prime number p.

The Diffie-Hellman key exchange is based on two parties each agreeing on a randomly generated integer, and a prime integer. Once both parties have agreed on these, they generate a private integer (which must be less than the prime).

Each party proceeds to raise the generated number to the power of their private number, modulo the prime number they both agreed on, to create their *public key.* Thus, each party continues the procedure by sharing their newly created *public key* with one another, and use this to calculate their *shared secret.* This is done by raising the *other party's* public key to the power of their *private keys* modulo their agreed on prime.

For example, let's assume *Alice* and *Bob* agree on using a prime, *p,* with the value 83, and a generated value g of 58.

Prime number (p):	83
Generated number (g):	58

Alice			**Bob**	
Random number picked (a):	30		Random number picked (b):	65
Public key (g^a mod p):	58^{30} mod 83 = 9		Public key (g^b mod p):	58^{65} mod 83 = 22
Bob's public key (B):	22	←→	Alice's public key (A):	9
Shared key (B^a mod p):	22^{30} mod 83 = 29		Shared key (A^b mod p):	9^{65} mod 83 = 29
Shared secret: 29				

Generating a Private Key: DSA

Generating a Private Key: RSA

The RSA method is similar, albeit different, from the Diffie-Hellman method – it's similar in the sense that it's based on the exchange of public keys which are used for encryption, and using private keys for decryption.

Encryption

Having acquired a public key from the other party, and generated a private key of its own, encryption can now start. The public key will be used to encrypt, so as to allow the other party to decrypt using its own generated private key; whereas the other party will use our public key to encrypt, allowing us to decrypt with our generated private key.