Integrating security into the Network stack

The Internet Protocol stack consists of many layers working together to provide a service. Over the years, many security protocols have been developed and refined. Each security protocol operates at a particular layer of the network stack, providing its own security service.

For instance, in the upper three layers of the network stack, we can find the corresponding security protocols. The most familiar one would be HTTPS, an extension of the insecure HTTP protocol. There are many security protocols that exist, but only the more common ones are shown here.

Layer	Communication Protocol	Security Protocol
Application	HTTP	HTTPS, PGP
Transport	TCP, UDP	TLS, SSH
Network	IP	IPSec

HTTPS

HTTPS stands for HTTP over TLS (a transport layer-security protocol). HTTP is insecure because the browser can read everything that is in the HTTP message. Most modern browsers and sites run on HTTPS, especially for sensitive functions like banking.

A site is running on HTTPS when it starts with https:// instead of http://. Some sites like google.com are enforcing that users use HTTPS. Then, using an old operating system and browser like Internet Explorer 3 on Windows XP, users cannot visit google.com

HTTPS provides two keys features.

1. HTTPS will encrypt the Application Payload using TLS.

> Recall that:
> Transport Packet = [Transport Header | Application Payload]
> With HTTPS,
> Transport Packet = [Transport Header | **TLSEncrypt (Application Payload)**]

Since the application payload is encrypted, the HTTP fields are hidden. Browsers cannot know which URL was visited (e.g. www.google.com/translate). However, this does not provide 100% privacy as the IP address of the website is still found in the Network Layer. Hence, attackers know which site you visit but not specific pages (/home, /about, /contact) within the site. This protection reduces a lot of data that browsers collect to tailor advertisements to users.

2. HTTPS authenticates the server with a server certificate

During the TLS protocol, the server is require to show its certificate. This certificate verifies the identity of the server. Example:

This certificate verifies that www.google.com is owned by Google LLC.

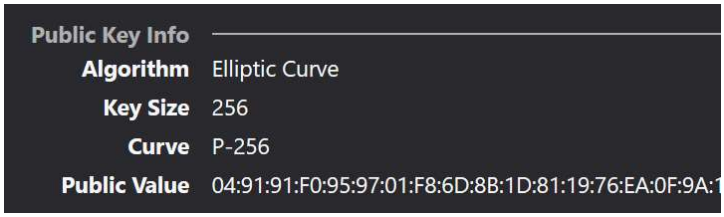
HTTPS is particularly useful against spoofing attacks, where an attacker uses a website like www.googlesite.com and claims to be Google. For users, it is difficult to verify if the claim is true because they would have to call Google up. However, with a certificate, the site can prove its identity. The certificate for www.googlesite.com (if valid) would tie the www.googlesite.com to Google LLC.

The browser will not display the lock icon if the URL of the site does not match the URL of the certificate. Certificates are a quick way to check the legitimacy of different sites. However, having a valid certificate does not make the site valid. For instance, www.googlesite.com can have a certificate for the organisation GoogleSite LLC (another non-Google company). The site can still pretend to be Google LLC if users do not check the organisation in the certificate.

Within the certificate, the client can retrieve the public key of the server. In essence, the certificate verifies that the public key belongs to the server. The public key is subsequently used to decide on a shared, secret private key in a key exchange protocol. An example is the Diffie-Hellman Key Exchange protocol, which is commonly used in TLS.

TLS

HTTPS is HTTP over TLS. TLS allows a client and server to exchange server certificates and decide on an encryption protocol. For instance, during the TLS handshake with Google, my browser used ECC. In TLS, it is possible to specify a preferred encryption scheme. Most browsers do not specify this, and instead let the server decide on the encryption algorithm to use.



During TLS, the client knows the server's public key, and can initiate a key exchange protocol. The result of the exchange is a shared private key that both client and server will use to encrypt all messages. In the case of websites, this shared key will be used to encrypt the HTTP message. During the TLS handshake, both sides will also agree on which symmetric key algorithm to use. The most common algorithm is AES.

Together, TLS and HTTPS allow a client visiting a site to verify the server's identity, and also encrypt the HTTP messages that are exchanged.