

# TEMA 4

---

Seguridad en Aplicaciones Web

# Índice

- Seguridad a nivel de Transporte (SSL)
- Control de Acceso en la web (OAuth 2.0)
- Vulnerabilidades web y pruebas de intrusión (WebGoat)
- Privacidad y Anonimato (Tor)

# SEGURIDAD A NIVEL DE TRANSPORTE

---

# Seguridad en la web

- La expansión de la WWW a mediados de los 90 llevó a los usuarios a hacer compras por la red
  - usando sistemas de pagos electrónicos, principalmente los basados en tarjetas de crédito
- Pero el potencial uso fraudulento de la información de las tarjetas de crédito llevó a buscar cómo proporcionar seguridad a las transacciones Web y a los servicios asociados
  - especialmente una vez conocidos los muchos tipos de amenazas que pueden aparecer al usar la Web (ver tabla siguiente)

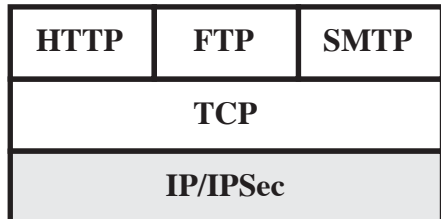


# Servicios de seguridad necesarios

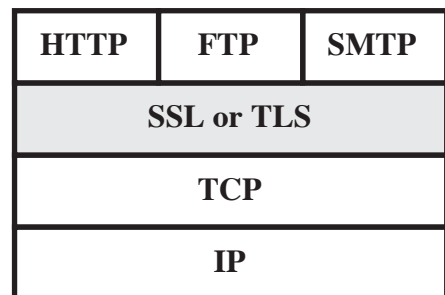
	Threats	Consequences	Countermeasures
<b>Integrity</b>	<ul style="list-style-type: none"><li>•Modification of user data</li><li>•Trojan horse browser</li><li>•Modification of memory</li><li>•Modification of message traffic in transit</li></ul>	<ul style="list-style-type: none"><li>•Loss of information</li><li>•Compromise of machine</li><li>•Vulnerability to all other threats</li></ul>	Cryptographic checksums
<b>Confidentiality</b>	<ul style="list-style-type: none"><li>•Eavesdropping on the Net</li><li>•Theft of info from server</li><li>•Theft of data from client</li><li>•Info about network configuration</li><li>•Info about which client talks to server</li></ul>	<ul style="list-style-type: none"><li>•Loss of information</li><li>•Loss of privacy</li></ul>	Encryption, web proxies
<b>Denial of Service</b>	<ul style="list-style-type: none"><li>•Killing of user threads</li><li>•Flooding machine with bogus requests</li><li>•Filling up disk or memory</li><li>•Isolating machine by DNS attacks</li></ul>	<ul style="list-style-type: none"><li>•Disruptive</li><li>•Annoying</li><li>•Prevent user from getting work done</li></ul>	Difficult to prevent
<b>Authentication</b>	<ul style="list-style-type: none"><li>•Impersonation of legitimate users</li><li>•Data forgery</li></ul>	<ul style="list-style-type: none"><li>•Misrepresentation of user</li><li>•Belief that false information is valid</li></ul>	Cryptographic techniques

# Diferentes soluciones en la pila TCP/IP

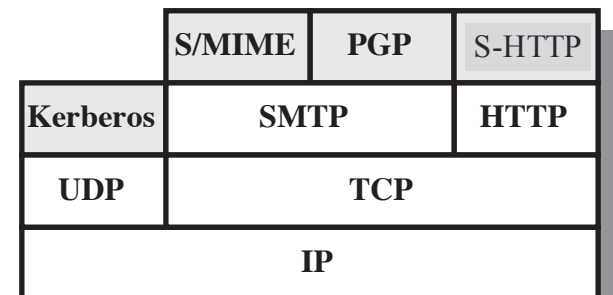
- De la tabla anterior se desprende que los posibles puntos de ataques son:
  - el cliente Web
  - el servidor Web
  - el tráfico de la red entre cliente y servidorinteresándonos en este momento el último de los tres
- Para dar una solución al problema del tráfico Web existen diferentes soluciones, dependiendo de la capa TCP/IP a considerar



(a) Network Level



(b) Transport Level



(c) Application Level

# Solución inicial: Modificar HTTP

- De hecho, el IETF formó a mediados de los 90 un Grupo de Trabajo denominado *Web Transaction Security (WTS)*
  - su objetivo: desarrollar los requisitos y las especificaciones para la provisión de servicios de seguridad en transacciones Web

## Web Transaction Security (wts)

(concluded WG)

[Documents](#) | [Charter](#) | [History](#) | [List Archive](#) » | [Tools WG Page](#) »

### Description of Working Group

The goal of the Web Transaction Security Working Group is to develop requirements and a specification for the provision of security services to Web transaction, e.g., transactions using HyperText Transport Protocol (HTTP). This work will proceed in parallel to and independently of the development of non-security features in the HTTP Working Group. The working group will prepare two documents for submission as Internet Drafts; an HTTP Security Requirements Specification, and an HTTP Security Protocol Specification. The latter will be submitted as a Standards Track RFC.

### Goals and Milestones

Jul 1995	HTTP Security Requirements finalized at the Stockholm IETF. Submit HTTP Security Specification proposal(s) as Internet-Drafts.
Dec 1995	HTTP Security Specification finalized at the Dallas IETF, submit to IESG for consideration as a Proposed Standard.
Done	HTTP Security Requirements submitted as Internet-Draft.

Note: The data for concluded WGs is occasionally incorrect.

#### Group

Name: Web Transaction Security  
Acronym: wts  
Area: Security Area (sec)  
State: Concluded  
Charter: [charter-ietf-wts-01](#) (Approved)

#### Personnel

Chair: [Charlie Kaufman <charlie\\_kaufman@notesdev.ibm.com>](#)  
Area Director: ?

#### Mailing List

Address: [www-security@nsmx.rutgers.edu](#)  
To Subscribe: [www-security-request@nsmx.rutgers.edu](#)  
Archive: [http://www.ns.rutgers.edu/www-security](#)

# SHTTP no terminó de funcionar ...

- Este grupo se centró en el desarrollo de una solución en la capa de aplicación, y diseñó el protocolo **SHTTP - Secure HyperText Transfer Protocol**, especificado en los documentos RFC que se observan abajo

## Web Transaction Security (wts)

**(concluded WG)**

[Documents](#) | [Charter](#) | [History](#) | [List Archive »](#) | [Tools WG Page »](#)

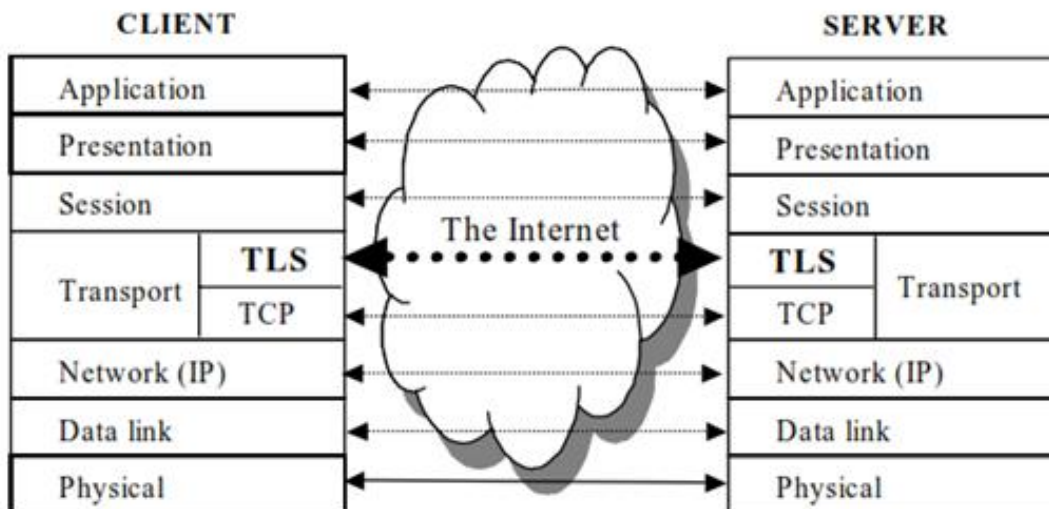
Document	Title	Date	Status	IPR	Area Director
RFCs					
<a href="#">RFC 2084</a> ( <a href="#">draft-ietf-wts-requirements</a> )	Considerations for Web Transaction Security	1997-01	RFC 2084 (Informational)		
<a href="#">RFC 2659</a> ( <a href="#">draft-ietf-wts-shtml</a> )	Security Extensions For HTML	1999-08	RFC 2659 (Experimental)		
<a href="#">RFC 2660</a> ( <a href="#">draft-ietf-wts-shttp</a> )	The Secure HyperText Transfer Protocol	1999-08	RFC 2660 (Experimental)		
Related Documents					
Related Documents	Title	Date	Status	IPR	Area Director



# Secure Sockets Layer - SSL



- Por otro lado, en las mismas fechas, los desarrolladores de Netscape abordaron el problema pero desde la capa de transporte
  - como una solución intermedia (ni en la capa alta ni en la baja)
- El resultado fue el protocolo **SSL - Secure Sockets Layer**, una especie de subcapa entre la de aplicación y la de transporte
  - más concretamente, SSL se sitúa por encima de TCP dado que este es orientado a la conexión y proporciona fiabilidad



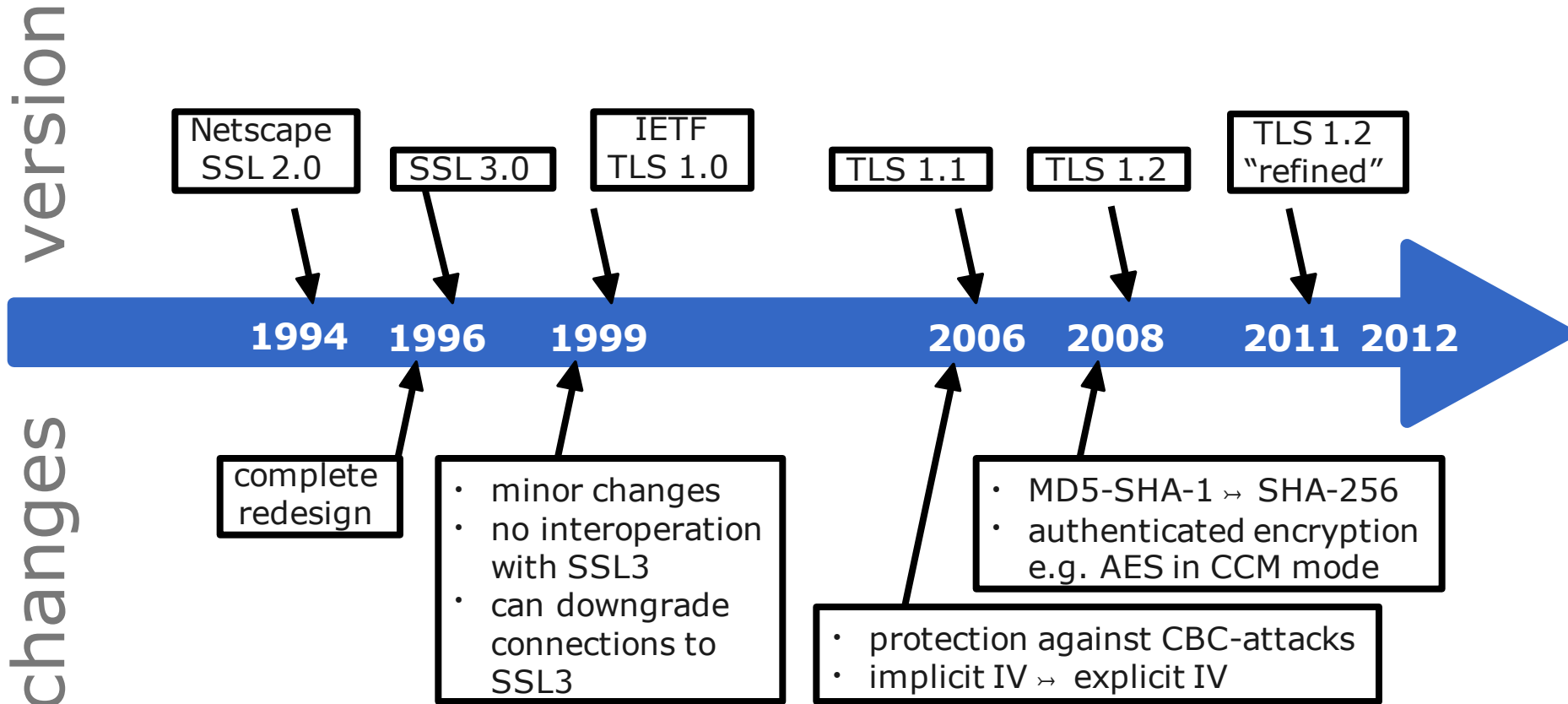
El objetivo del protocolo SSL es, por tanto, crear conexiones seguras y transmitir datos a través de esas conexiones.

# Transport Layer Security - TLS

- TLS es una iniciativa de IETF para estandarizar SSL
- Se definió por primera vez (TLS 1.0) en 1999, en el RFC 2246. Como se menciona en ese RFC:

“the differences between this protocol and SSL 3.0 are not dramatic, but they are significant to preclude interoperability between TLS 1.0 and SSL 3.0.”
- Versiones posteriores han sido TLS 1.1, publicada en 2006 en el RFC 4346, y TLS 1.2, publicada en 2008 en el RFC 5246
- La última de estas versiones sí introduce cambios más significativos, como la inclusión de AES en el cipher suite, la consideración de SHA-256, así como la consideración de criptografía de clave pública basada en curvas elípticas

# SSL -> TLS



**MAC** Message Authentication Code  
**IETF** Internet Engineering Task Force  
**CBC** Cipher Block Chaining  
**IV** Initialization Vector

**MD5** Message Digest Algorithm  
**SHA** Secure Hash Algorithm  
**AES** Advanced Encryption Standard  
**CCM** Counter with CBC-MAC

# Tecnologías empleadas en SSL

- El protocolo SSL es un protocolo cliente/servidor que proporciona los siguientes servicios de seguridad entre los puntos que se comunican:
  - Autenticación de entidades y de origen de datos
  - Confidencialidad de la conexión
  - Integridad de la conexión
- Más concretamente, SSL emplea:
  - **criptografía simétrica** para la **autenticación de los mensajes** y para la **confidencialidad** de los datos
  - **criptografía de clave pública** para la **autenticación de las entidades** y para el **establecimiento de clave**
    - básicamente, hay tres algoritmos de intercambio de clave en la especificación de SSL: **RSA**, **Diffie-Hellmann** y Fortezza
    - a pesar de la utilización de criptografía de clave pública, **no proporciona el servicio de no-repudio**
      - ni no-repudio de origen ni no-repudio de entrega



# Protocolo independiente de la aplicación

- Una ventaja del protocolo SSL es que es independiente del protocolo de la capa de aplicación
  - es decir, cualquier protocolo de aplicación basado en TCP se puede beneficiar de SSL (éste le dota de los servicios de seguridad mencionados)

Port Numbers Reserved for Application Protocols Layered over SSL/TLS

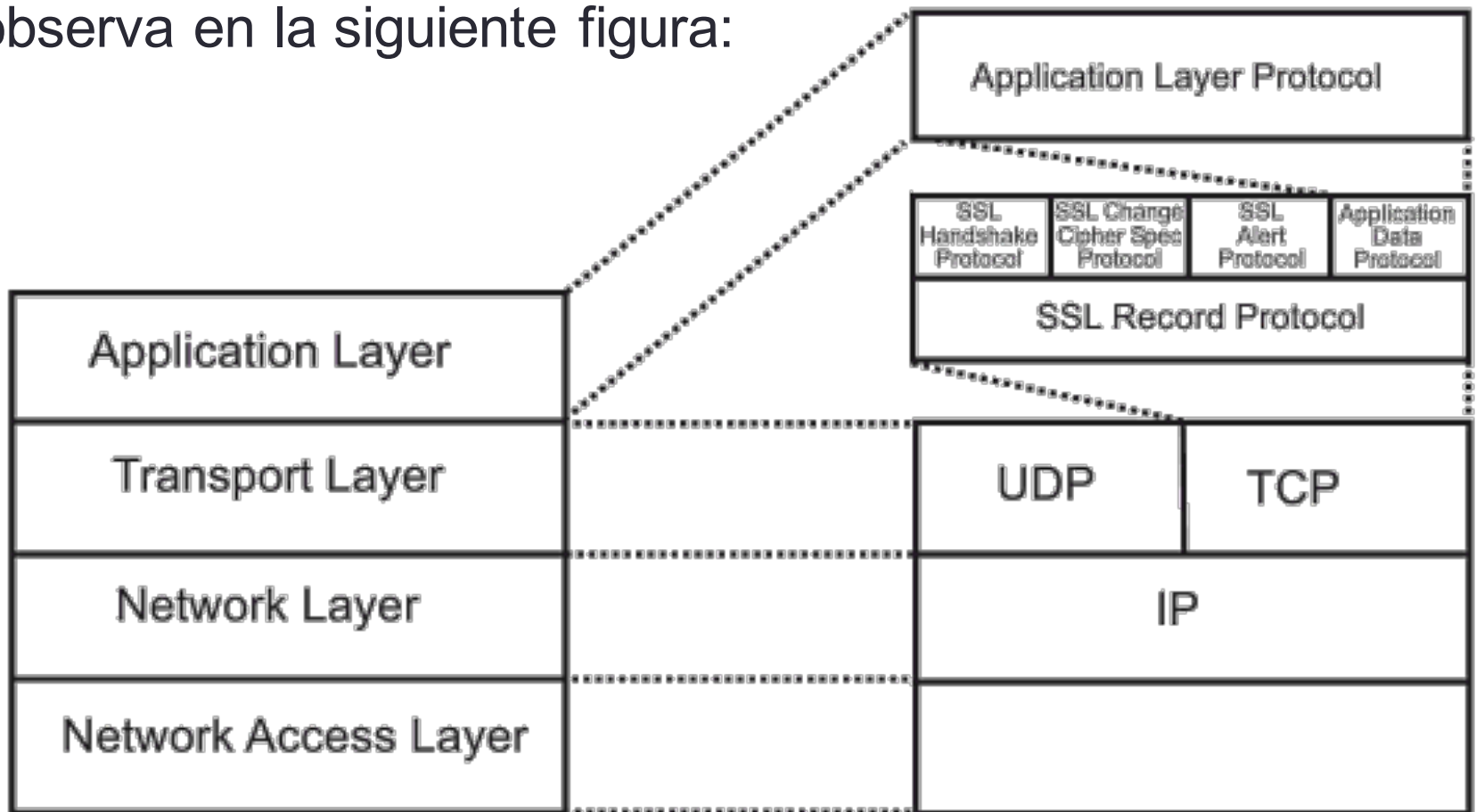
Protocol	Description	Port #
nsiops	IIOP Name Service over SSL/TLS	261
https	HTTP over SSL/TLS	443
nntps	NNTP over SSL/TLS	563
ldaps	LDAP over SSL/TLS	636
ftps-data	FTP Data over SSL/TLS	989
ftps	FTP Control over SSL/TLS	990
telnets	Telnet over SSL/TLS	992
imaps	IMAP4 over SSL/TLS	993
ircs	IRC over SSL/TLS	994
pop3s	POP3 over SSL/TLS	995
tftps	TFTP over SSL/TLS	3713
sip-tls	SIP over SSL/TLS	5061
...	...	...

# Tipos de mensajes en SSL

- El protocolo SSL emplea, entre otros, estos dos conceptos:
  - **Sesión SSL**: asociación entre el cliente y el servidor en la que se negocian los parámetros de seguridad para todas las conexiones de esa sesión
  - **Conexión SSL**: transmisión de datos entre el cliente y el servidor, protegida criptográficamente según lo negociado en la sesión
- El ámbito de la funcionalidad de SSL es doble, como se mencionó:
  1. **Establecer una conexión segura** (confidencial y autenticada) entre los puntos que se comunican
  2. Utilizar esa conexión para **transmitir de forma segura los datos** del nivel de aplicación entre el emisor y el receptor. Esta transmisión requiere a su vez de:
    - Dividir los datos en fragmentos más manejables
    - Procesarlos de forma individual, donde cada fragmento así tratado y preparado se denomina **SSL record**

# Diseño en dos capas

- Para llevar a cabo esa doble funcionalidad, SSL consta de dos subcapas y varios sub-protocolos, como se observa en la siguiente figura:



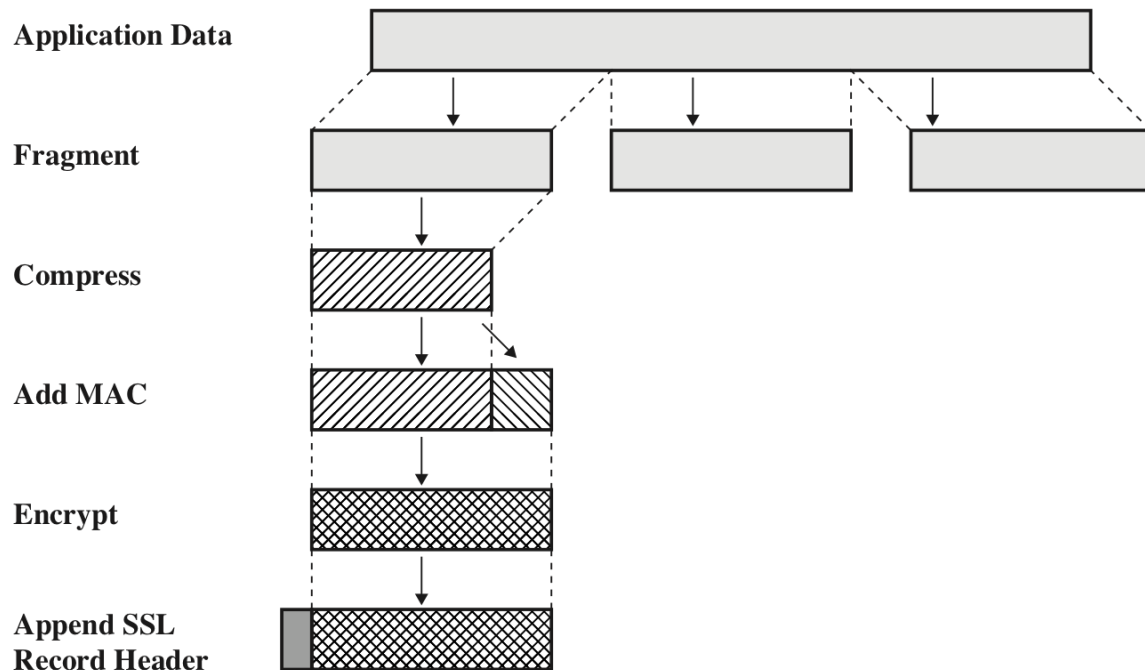
# Sub-protocolos de SSL

- De forma resumida, la subcapa baja contiene:
  - **SSL Record Protocol**: fragmenta los datos de la capa de aplicación y los procesa de forma individual
- Y la subcapa alta contiene:
  - **SSL Handshake Protocol**: permite que los puntos de comunicación se autenticuen mutuamente, y que además negocien un **cipher suite** y (opcionalmente) un método de compresión
  - **SSL Change Cipher Spec Protocol**: permite a los puntos de comunicación activar el cipher suite
  - **SSL Alert Protocol**: permite a los puntos de comunicación indicar posibles problemas potenciales e intercambiar los correspondientes mensajes de alerta
  - **SSL Application Data Protocol**: es el propio protocolo de la capa de aplicación (ej: HTTP), y alimenta al SSL Record Protocol



# SSL Record Protocol

- Toma los datos de la subcapa alta, los fragmenta en bloques manejables, los comprime de forma opcional, añade el MAC, cifra, y añade una cabecera
- El resultado final se transmite en un segmento TCP
- En recepción, los datos recibidos son descifrados, verificados, descomprimidos y re-ensamblados antes de entregarlos a la capa de aplicación



# SSL Record Protocol (II)

- Por lo tanto, este subprotocolo proporciona:
  - *Confidencialidad*: el SSL Handshake Protocol define una clave secreta compartida que es utilizada para el cifrado de los datos
  - *Integridad de datos*: el SSL Handshake Protocol también define una clave secreta compartida que es utilizada para formar un **MAC**
- En lo que a fragmentación se refiere, cada mensaje de la capa de aplicación se fragmenta en bloques de longitud  $2^{14}$  bytes o menor
- En lo que respecta al algoritmo de compresión, SSL no especifica ninguno
- En cuanto al código de autenticación de mensajes, se utiliza uno similar a HMAC

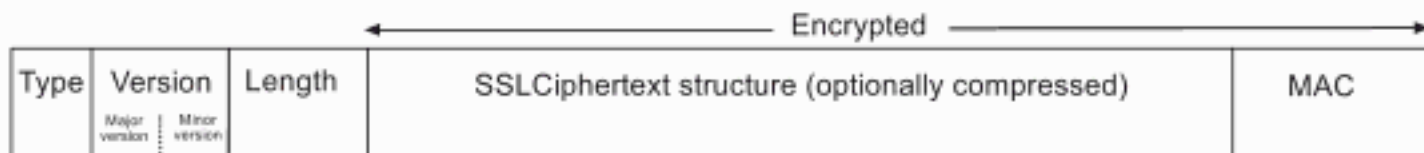
# Suites soportadas por SSL

- El mensaje comprimido y el valor MAC se cifran utilizando criptografía simétrica. Los algoritmos que se pueden utilizar se muestran en la tercera columna de la tabla:

CipherSuite	Key Exchange	Cipher	Hash
SSL_NULL_WITH_NULL_NULL	NULL	NULL	NULL
SSL_RSA_WITH_NULL_MD5	RSA	NULL	MD5
SSL_RSA_WITH_NULL_SHA	RSA	NULL	SHA
SSL_RSA_EXPORT_WITH_RC4_40_MD5	RSA_EXPORT	RC4_40	MD5
SSL_RSA_WITH_RC4_128_MD5	RSA	RC4_128	MD5
SSL_RSA_WITH_RC4_128_SHA	RSA	RC4_128	SHA
SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5	RSA_EXPORT	RC2_CBC_40	MD5
SSL_RSA_WITH_IDEA_CBC_SHA	RSA	IDEA_CBC	SHA
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA	RSA_EXPORT	DES40_CBC	SHA
SSL_RSA_WITH_DES_CBC_SHA	RSA	DES_CBC	SHA
SSL_RSA_WITH_3DES_EDE_CBC_SHA	RSA	3DES_EDE_CBC	SHA
SSL_DH_DSS_EXPORT_WITH_DES40_CBC_SHA	DH_DSS_EXPORT	DES40_CBC	SHA
SSL_DH_DSS_WITH_DES_CBC_SHA	DH_DSS	DES_CBC	SHA
SSL_DH_DSS_WITH_3DES_EDE_CBC_SHA	DH_DSS	3DES_EDE_CBC	SHA
SSL_DH_RSA_EXPORT_WITH_DES40_CBC_SHA	DH_RSA_EXPORT	DES40_CBC	SHA
SSL_DH_RSA_WITH_DES_CBC_SHA	DH_RSA	DES_CBC	SHA
SSL_DH_RSA_WITH_3DES_EDE_CBC_SHA	DH_RSA	3DES_EDE_CBC	SHA
SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA	DHE_DSS_EXPORT	DES40_CBC	SHA
SSL_DHE_DSS_WITH_DES_CBC_SHA	DHE_DSS	DES_CBC	SHA
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA	DHE_DSS	3DES_EDE_CBC	SHA
SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA	DHE_RSA_EXPORT	DES40_CBC	SHA
SSL_DHE_RSA_WITH_DES_CBC_SHA	DHE_RSA	DES_CBC	SHA
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA	DHE_RSA	3DES_EDE_CBC	SHA
SSL_DH_anon_EXPORT_WITH_RC4_40_MD5	DH_anon_EXPORT	RC4_40	MD5
SSL_DH_anon_WITH_RC4_128_MD5	DH_anon	RC4_128	MD5
SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA	DH_anon	DES40_CBC	SHA
SSL_DH_anon_WITH_DES_CBC_SHA	DH_anon	DES_CBC	SHA
SSL_DH_anon_WITH_3DES_EDE_CBC_SHA	DH_anon	3DES_EDE_CBC	SHA
SSL_FORTEZZA_KEA_WITH_NULL_SHA	FORTEZZA_KEA	NULL	SHA
SSL_FORTEZZA_KEA_WITH_FORTEZZA_CBC_SHA	FORTEZZA_KEA	FORTEZZA_CBC	SHA
SSL_FORTEZZA_KEA_WITH_RC4_128_SHA	FORTEZZA_KEA	RC4_128	SHA

# Formato de la trama

- El paso final del SSL Record Protocol es preparar una cabecera que consta de los siguientes campos:
  - *Content Type* (8 bits): protocolo de la subcapa alta de SSL de la que procede el fragmento:
    - 20 → SSL Change Cipher Spec Protocol
    - 21 → SSL Alert Protocol
    - 22 → SSL Handshake Protocol
    - 23 → SSL Application Data Protocol
  - *Major Version* (8 bits): versión de SSL en uso (para SSLv3, el valor es 3)
  - *Minor Version* (8 bits): versión menor en uso (para SSLv3, el valor es 0)
  - *Compressed Length* (16 bits): longitud en bytes del fragmento de texto en claro (o del fragmento comprimido si se ha utilizado compresión)



# SSL Handshake Protocol

- Se utiliza antes de transmitir ningún dato de la capa de aplicación
- Es la parte más compleja de SSL porque permite al servidor y al cliente autenticarse mutuamente, y negociar un algoritmo cifrado y una función MAC, así como las claves a usar para proteger los datos del SSL record
- Consta de una serie de mensajes intercambiados entre el cliente y el servidor con el formato:

1 byte	3 bytes	$\geq 0$ bytes
Type	Length	Content

- Por lo tanto, cada mensaje tiene 3 campos:
  - *Type* (1 byte): indica uno de 10 posibles mensajes (ver siguiente tabla)
  - *Length* (3 bytes): longitud del mensaje en bytes
  - *Content* ( $\geq 0$  bytes): parámetros asociados con el mensaje (ver siguiente tabla)

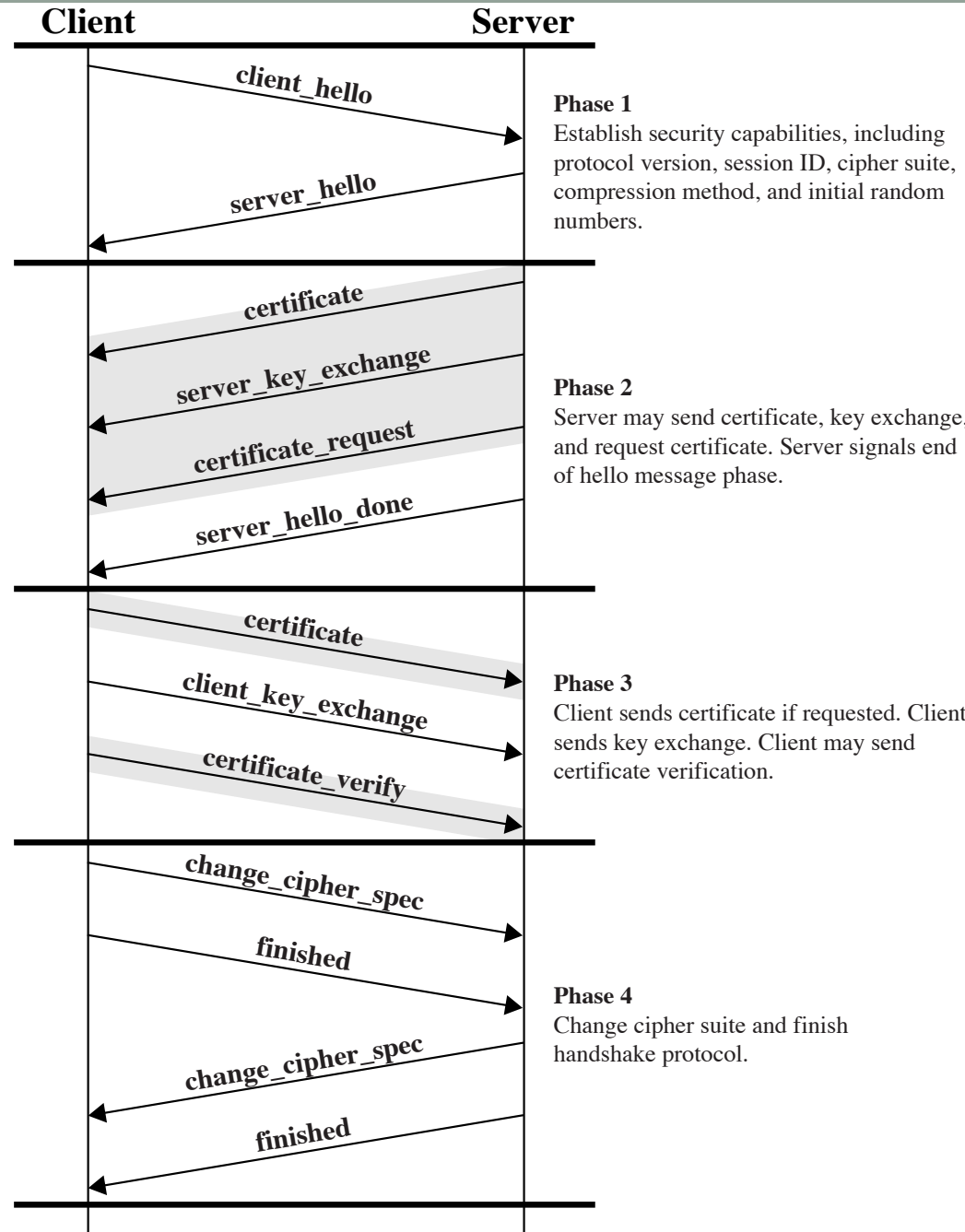
# Tipos de mensajes SSL Handshake

Message Type	Parameters
hello_request	null
client_hello	version, random, session id, cipher suite, compression method
server_hello	version, random, session id, cipher suite, compression method
certificate	chain of X.509v3 certificates
server_key_exchange	parameters, signature
certificate_request	type, authorities
server_done	null
certificate_verify	signature
client_key_exchange	parameters, signature
finished	hash value

# Pasos SSL

- La figura muestra el intercambio inicial necesario para establecer una conexión lógica entre cliente y servidor
- Se observan 4 fases

Time  
↓



# Resto de protocolos

- SSL Change Cipher Spec Protocol

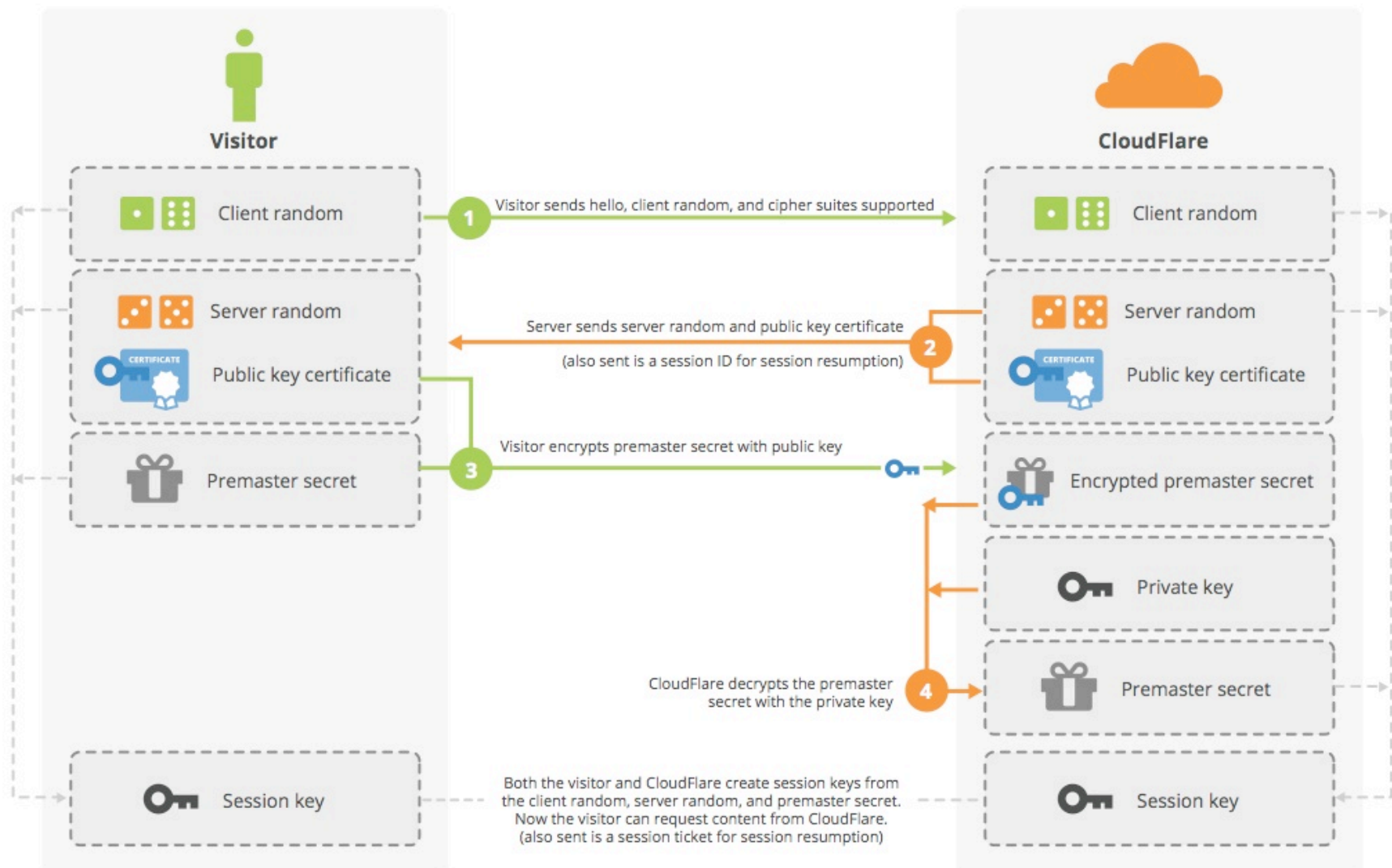
- Es un protocolo muy simple que consta de un solo mensaje de solo un byte con valor 1 que permite activar el cipher suite negociado previamente

- SSL Alert Potocol

- Se usa para comunicar al otro punto de comunicación las alertas relacionadas con SSL, y cada mensaje de este protocolo consta de 2 bytes
  - Estos mensajes también se comprimen y se cifran de acuerdo con lo establecido en la sesión
- El primer byte toma el valor 1 (warning) o 2 (fatal) para informar de la severidad del mensaje. Si el nivel es fatal SSL termina la conexión de forma inmediata
  - Otras conexiones de la misma sesión pueden continuar pero no se producen nuevas conexiones dentro de la misma sesión
- El segundo byte contiene un código que indica la alerta específica
  - Ejemplos: unexpected\_message, bad\_record\_mac, decompression\_failure, illegal\_parameter, ...



# Handshake SSL con RSA

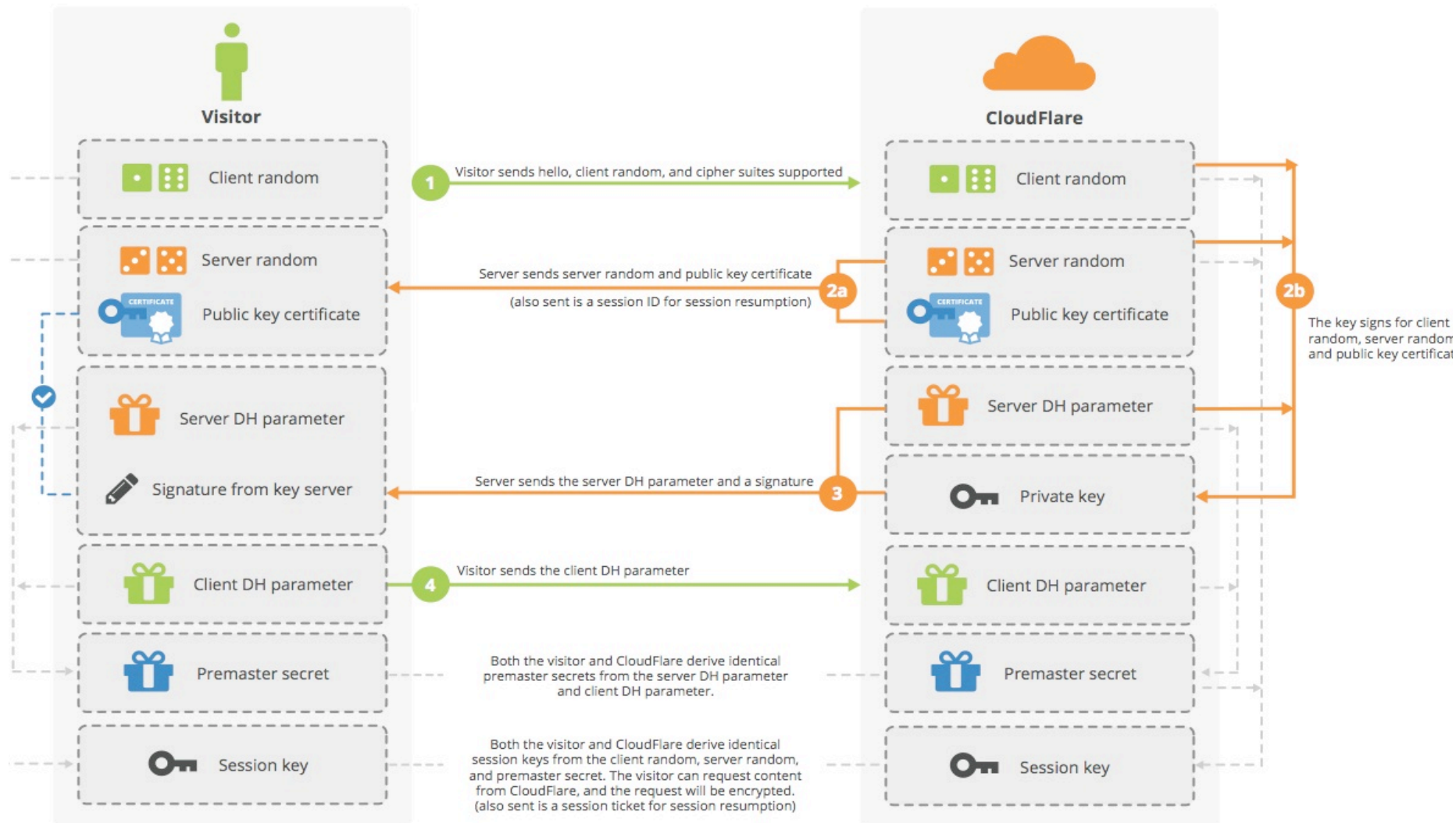


<https://blog.cloudflare.com/keyless-ssl-the-nitty-gritty-technical-details/>

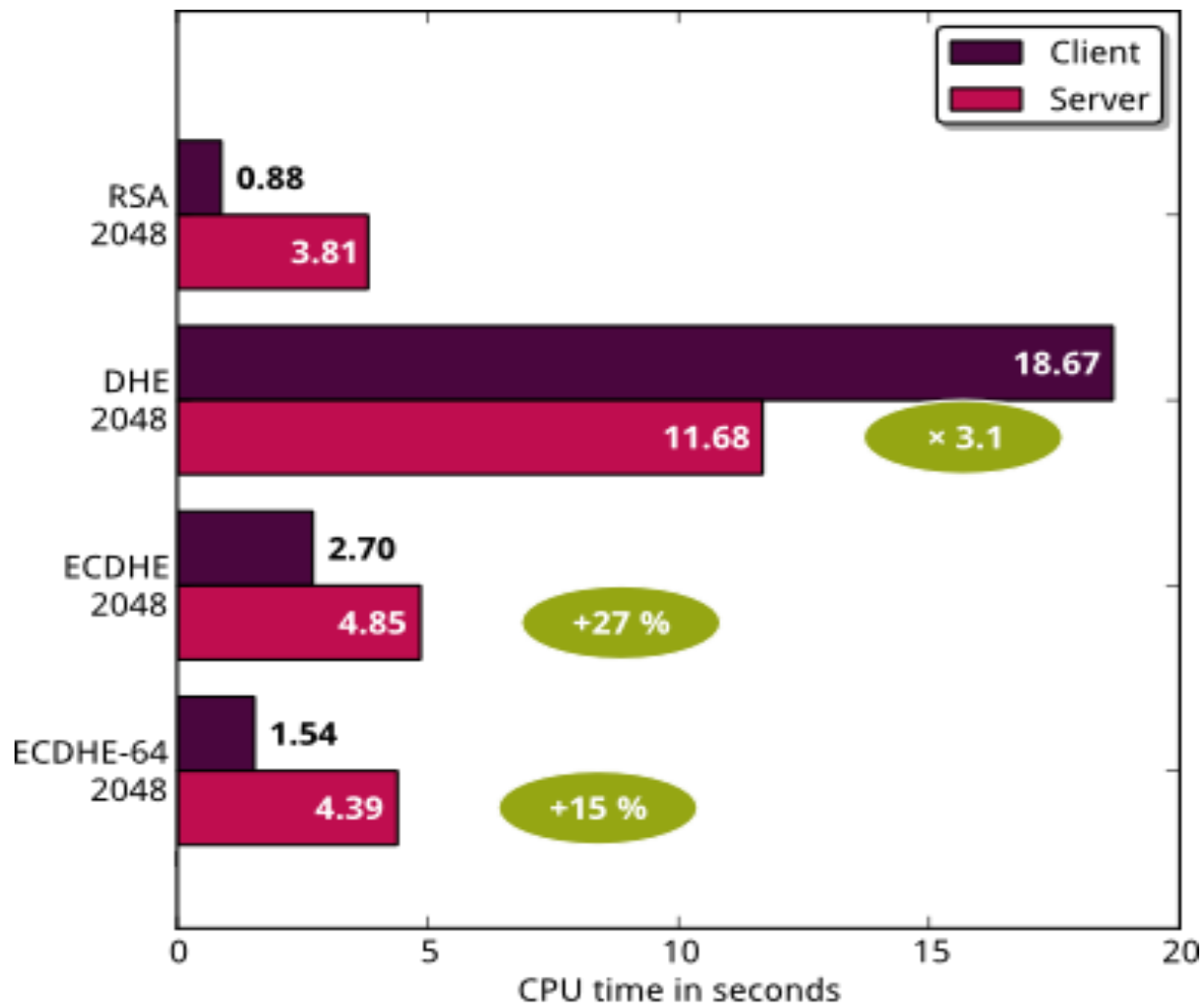
# Perfect Forward Secrecy (PFS)

- *“El descubrimiento de las claves utilizadas actualmente no compromete la seguridad de las claves usadas con anterioridad”*
- ¿Que ocurre si alguien se dedica a grabar todas las comunicaciones cifradas con un servidor dado durante años y en algún momento es capaz de comprometer el servidor y extraer la clave privada asociada al certificados SSL del servidor?
- ¿Se pueden descifrar las comunicaciones cifradas antiguas?
- **SI**, los secretos maestros usados para generar las claves de cifrado, IV y claves de autenticación se envían cifrados con la clave pública del servidor, así que son la privada podemos recuperarlos en cualquier momento.
- **Solución**, usar Diffie-Hellman con claves efímeras.

# Handshake SSL con Diffie-Hellman



# Rendimiento de SSL con RSA, DHE y ECDHE



# DTLS

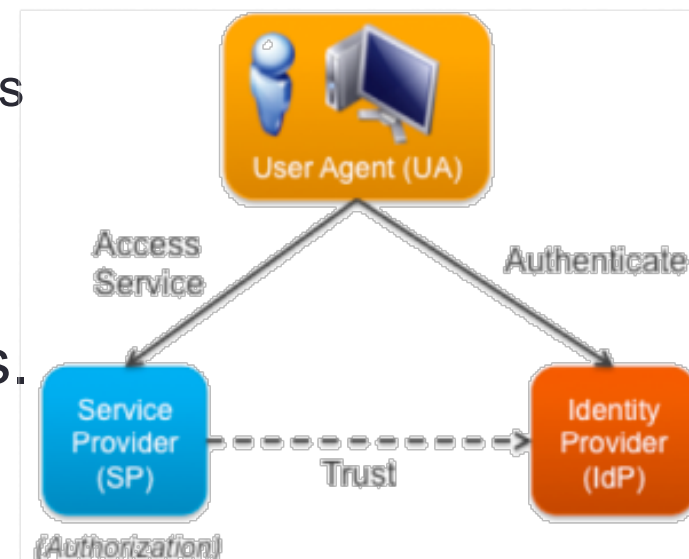
- Es conveniente comentar que existe un protocolo llamado **DTLS (Datagram Transport Layer Security)** definido en el RFC 6347
- Se utiliza para los protocolos basados en datagramas
  - Es decir, para los que se ejecutan por encima de UDP
- Se creó en 2006, aunque la última versión (1.2) es de Enero de 2012.
- Esta tomando un papel relevante en entornos restringidos (IoT) <https://datatracker.ietf.org/wg/dice/documents/>
- Se publicó un ataque en 2013 debido a que las mejoras de TLS 1.1 no se aplicaron de forma correcta en DTLS 1.1. <http://www.isg.rhul.ac.uk/~kp/dtls.pdf>

# CONTROL DE ACCESO EN LA WEB

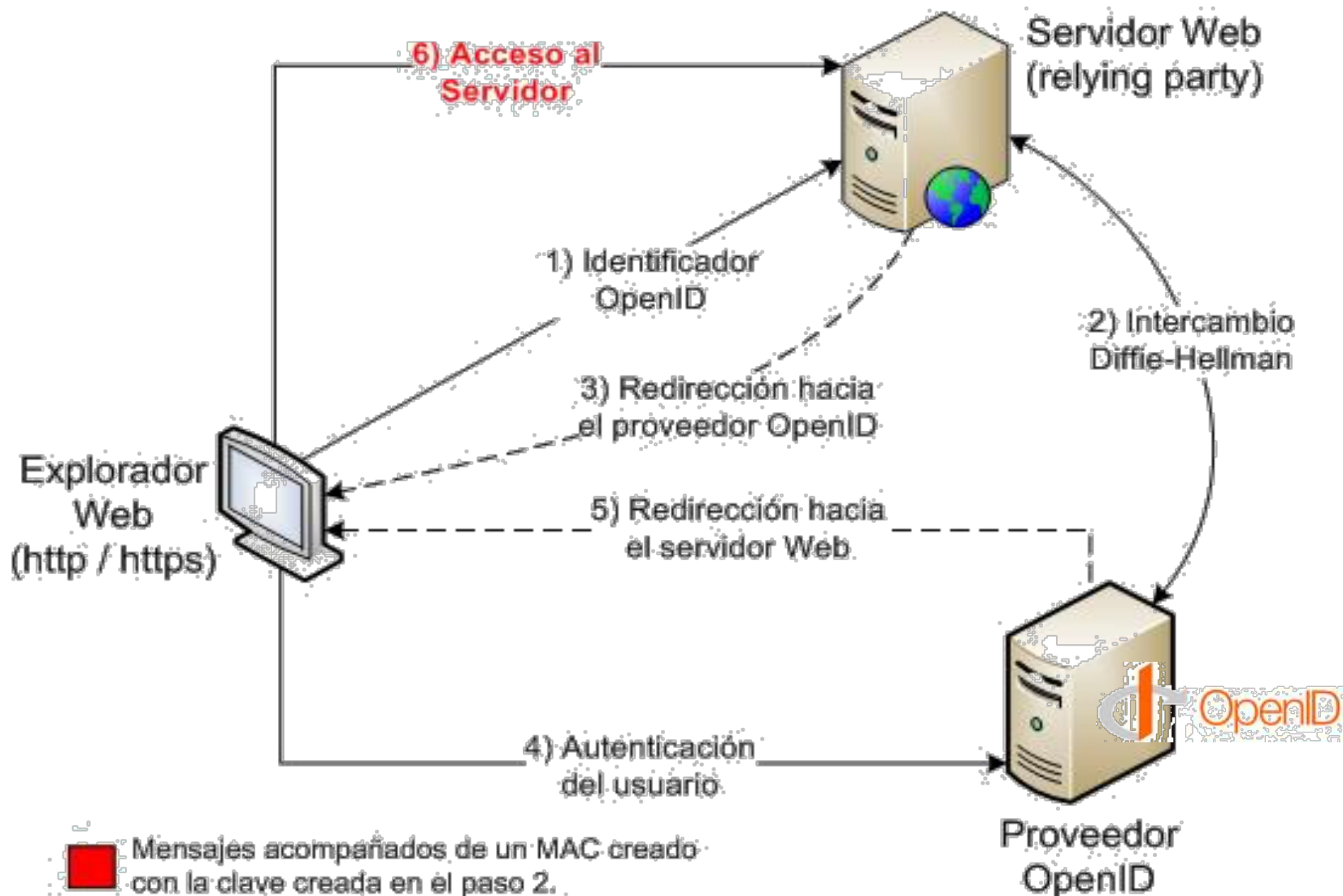
---

# Autenticación y Autorización en Internet

- Los sistemas de identificación/autenticación usuales presentan ***limitaciones en cuanto a la escalabilidad y usabilidad*** en entornos como Internet, además de problemas de seguridad
  - Ej: Un usuario y clave para acceder a cada/cualquier sitio web
- La tendencia actual es separar el control de acceso en dos fases
  - **Autenticación** Verificación de la identidad
  - **Autorización**. Verificación de los privilegios o derechos de acceso.
- Al trabajar en entornos distribuidos estos dos roles los toman entidades distintas que deben confiar entre ellas.



# OpenID





# Limitaciones OpenID

- El proveedor de OpenID tiene conocimiento de todas las webs que visitas
  - Falta de anonimato
- Vulnerable a ataques de Phishing
  - Se puede engañar a los usuarios para proporcionar las credenciales a sitios deshonestos haciéndose pasar por el proveedor de OpenID
- No resuelve el problema de la autorización
  - Solo resuelve el problema de verificar la identidad del usuario pero no entra en gestionar los permisos o derechos de acceso

# Oauth y OpenID

- Son estándares web **abiertos**, creados por personas que participan desinteresadamente en su desarrollo en todo el mundo.
- Usan una interfaz *REST*
- Buscan la **descentralización**, no hay ningún servidor central para OAuth ni OpenID.
- Ambos se basan en el **redireccionamiento** desde el sitio al que intentas acceder, “*consumidor*”, al sitio de un “*proveedor*”, y de vuelta, mientras ellos hablan entre si para verificar el proceso.
- El usuario puede controlar desde el proveedor qué sitios tienen acceso, pudiéndolo revocar en cualquier momento.

# OAuth

- OAuth permite **autorizar** que un sitio (**consumidor**) obtenga tu información personal desde otro (**proveedor**),
  - p.e. Puedes autorizar a que un servicio de impresión obtenga tus fotos desde una galería que hayas creado en otro servicio (como Flickr). El servicio de impresión te redirigirá a Flickr, donde se te preguntará si autorizas el acceso a los datos, y luego serás redirigido de vuelta.
- Se utiliza también en aplicaciones móviles para **no** tener que **almacenar credenciales** de usuario y poder revocar acceso.
- Adoptado por numerosos proveedores de servicio en Internet, p.e. Twitter, Google, Microsoft, Yahoo, etc.

# Elementos de OAuth 1.0

- **Proveedor de Servicios (SP):** Es el servicio en general donde se localizan los recursos restringidos (y que por tanto hay que proteger mediante un sistema de autenticación/autorización).
- **Usuario.** El usuario es el actor principal dentro de OAuth y es la persona que dispone de recursos privados que no quiere hacer públicos dentro del SP, pero quiere compartirlos.
- **Consumidor:** Se trata de la aplicación que está accediendo al recurso privado del usuario. Es por tanto, dicha aplicación la que mediante los permisos otorgados por el usuario el que accederá finalmente al recurso.

# Elementos de OAuth 1.0

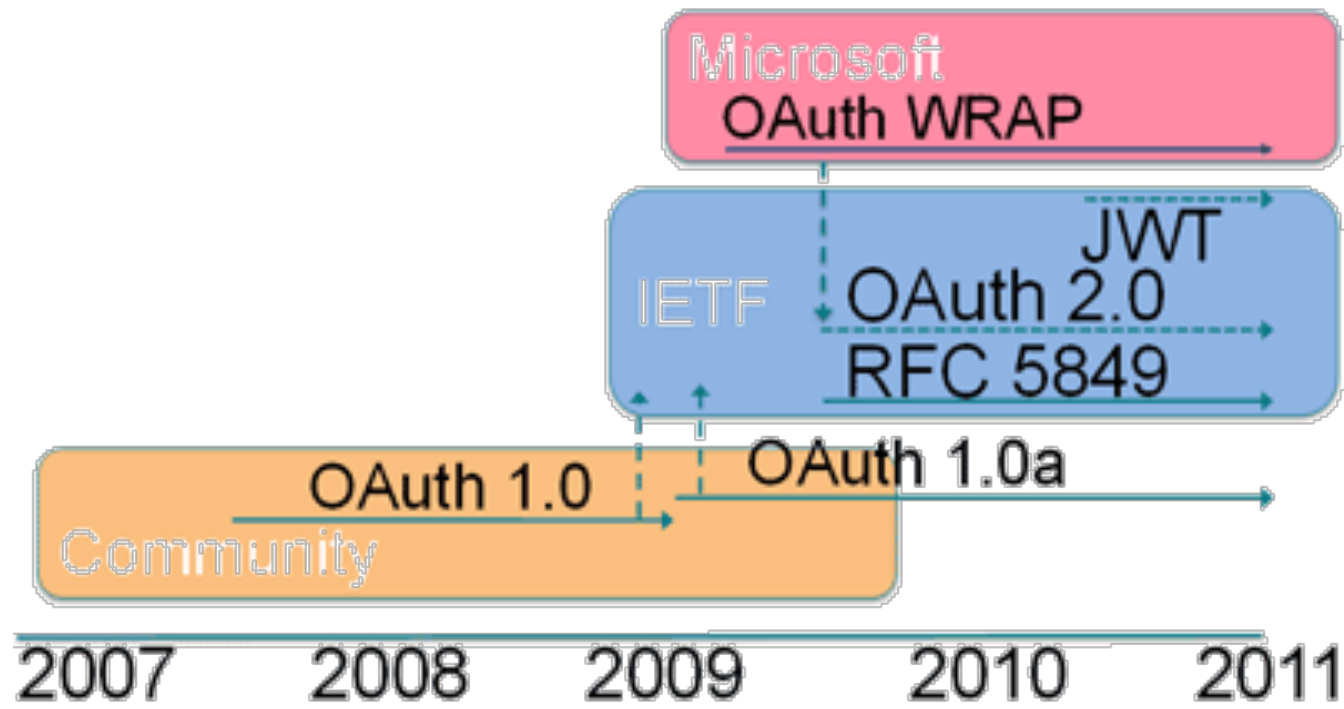
- **Recursos protegidos:** Aquellos recursos que son privados, y para los que se le otorgan permisos al Consumidor para que proceda a procesarlos adecuadamente: fotos, documentos, contactos, ...
- **Tokens:** Dentro del protocolo se usan tokens como sustituto a las credenciales del usuario. Se usan técnicas de *Firma Digital* para cifrar la información asociada al token y para garantizar la identidad de la persona/sistema que realiza la petición (Usuario/Consumer).
  - Dentro del protocolo hay dos tipos de tokens: *Request* y *Access*. Uno asociado con el Consumer y otro asociado con el Usuario.

# Medidas de seguridad

- Tanto los SP como los consumidores deben usar canales seguros o implementar HMAC-SHA1 ó RSA-SHA1.
- Proteger frente a Clickjacking y CSRF la web del SP en la que el usuario autoriza al consumidor.
- Utilizar un algoritmo con una alta entropía para generar los tokens y claves.
- Proteger frente accesos no autorizados la base de datos del SP ya que, a diferencia de las contraseñas de las que se recomienda almacenar tan sólo su función hash, las claves tienen que almacenarse en claro para poder implementar el protocolo.

# Evolución de OAuth 1.0 a 2.0

- **Tokens más simples** - OAuth 2.0 elimina las firmas y la criptografía a nivel de protocolo, basándose solo en HTTPS y cookies.
- **Renovación de Tokens** - Los tokens en OAuth 2.0 pueden expirar y deben ser refrescados lo que hace que las implementaciones de los clientes tengan que gestionar el estado de los tokens.



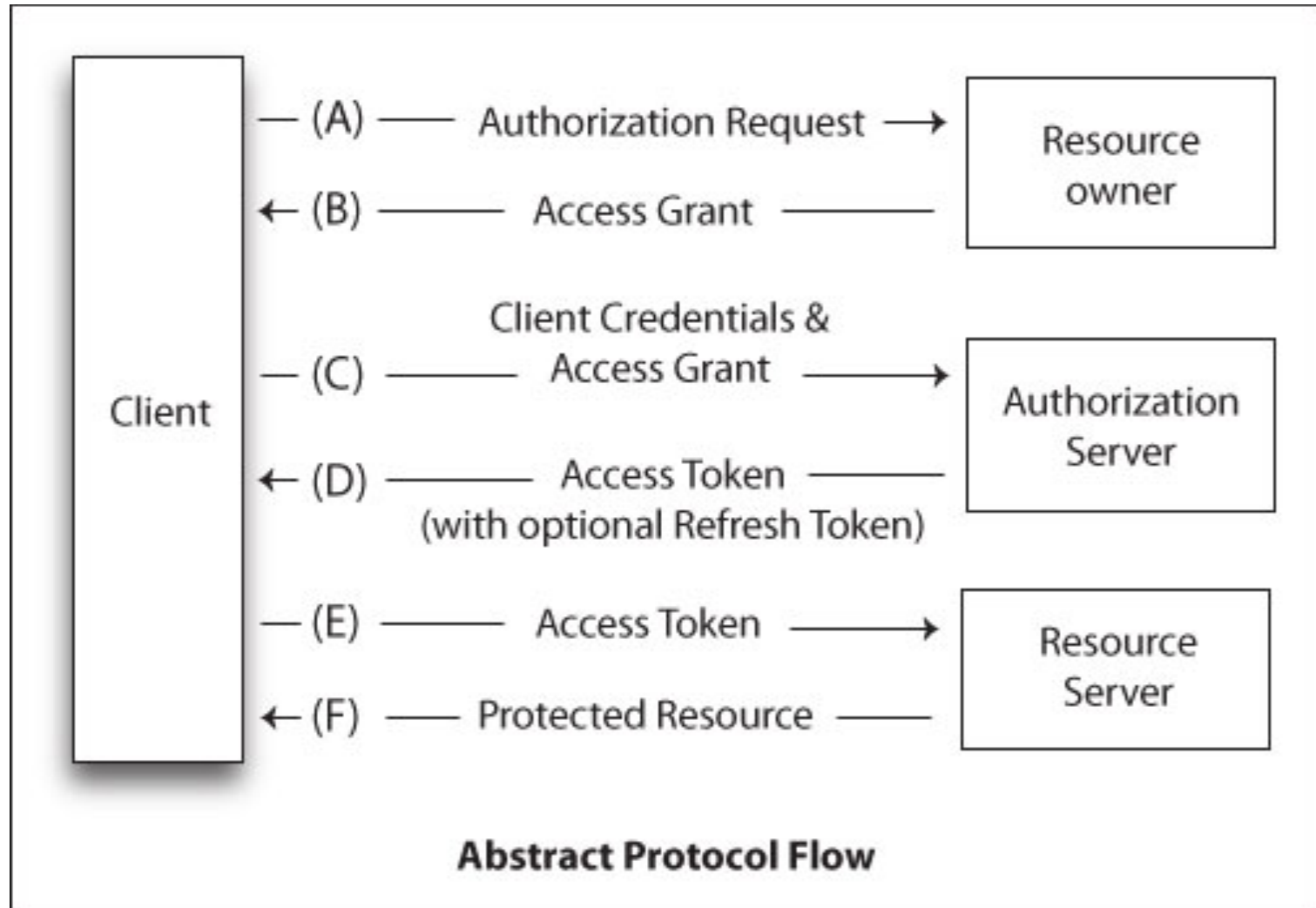
# "OAuth 2.0 - Looking Back and Moving On"



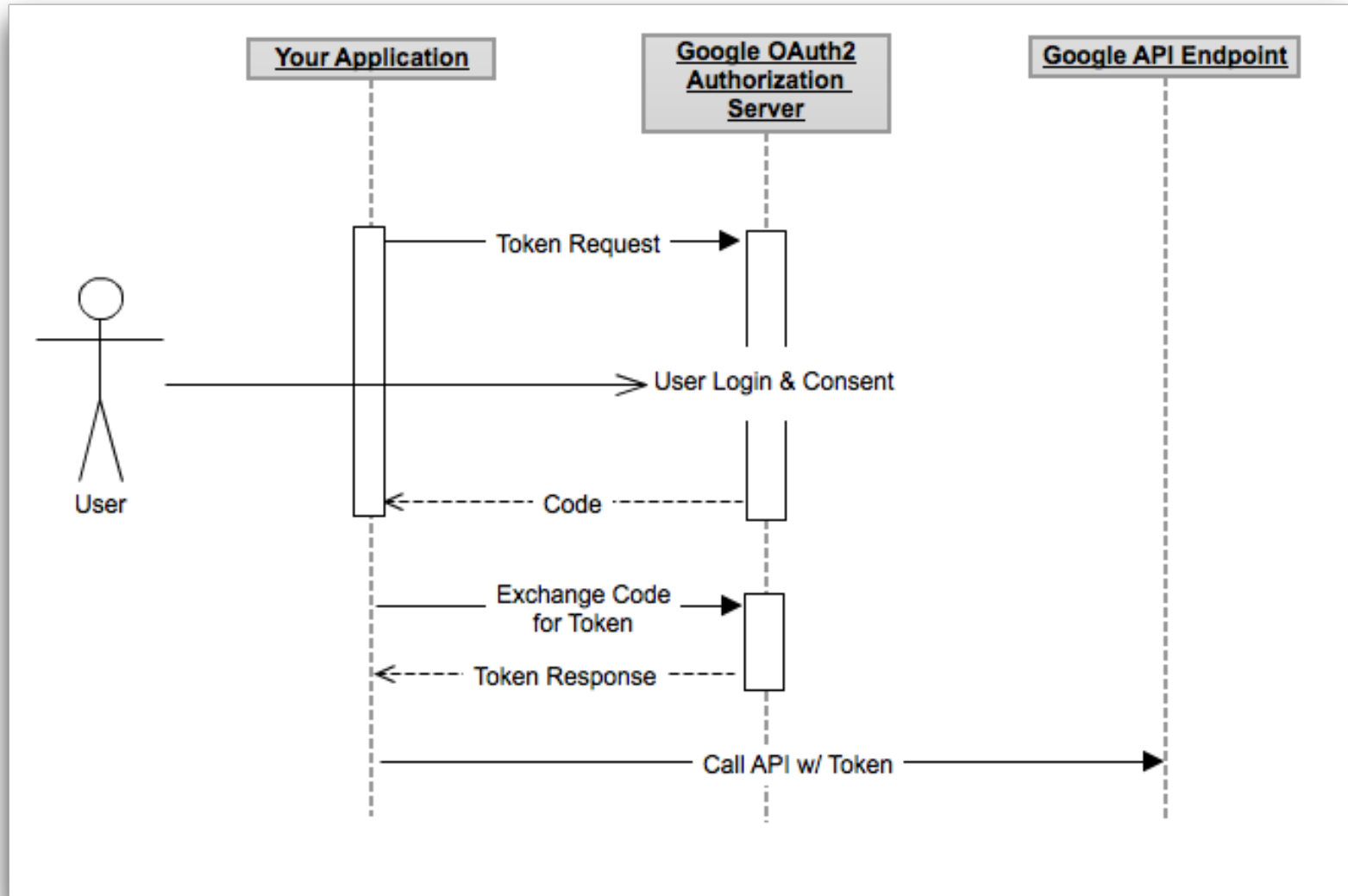
<http://vimeo.com/52882780>



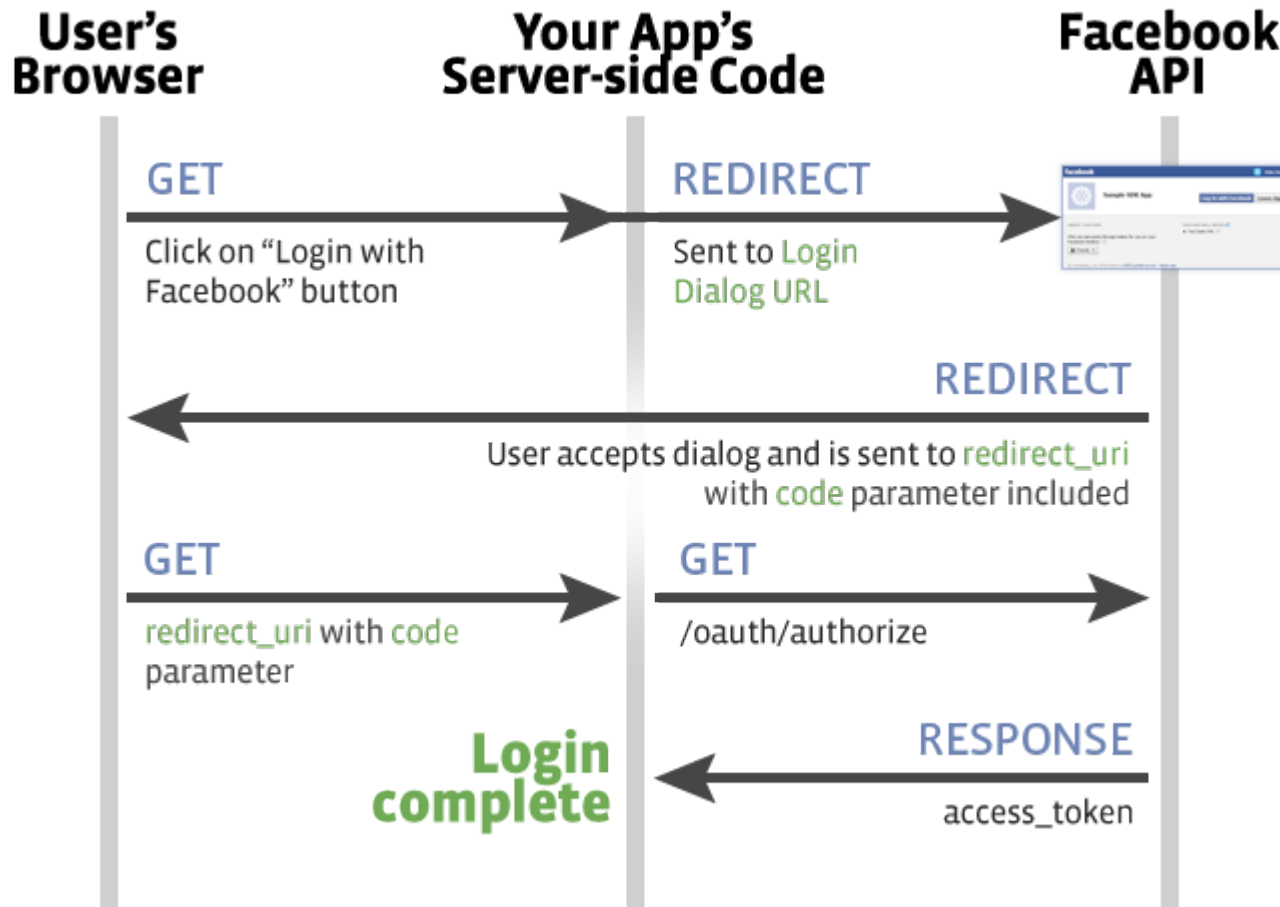
# OAuth 2.0 (IETF)



# OAuth Google

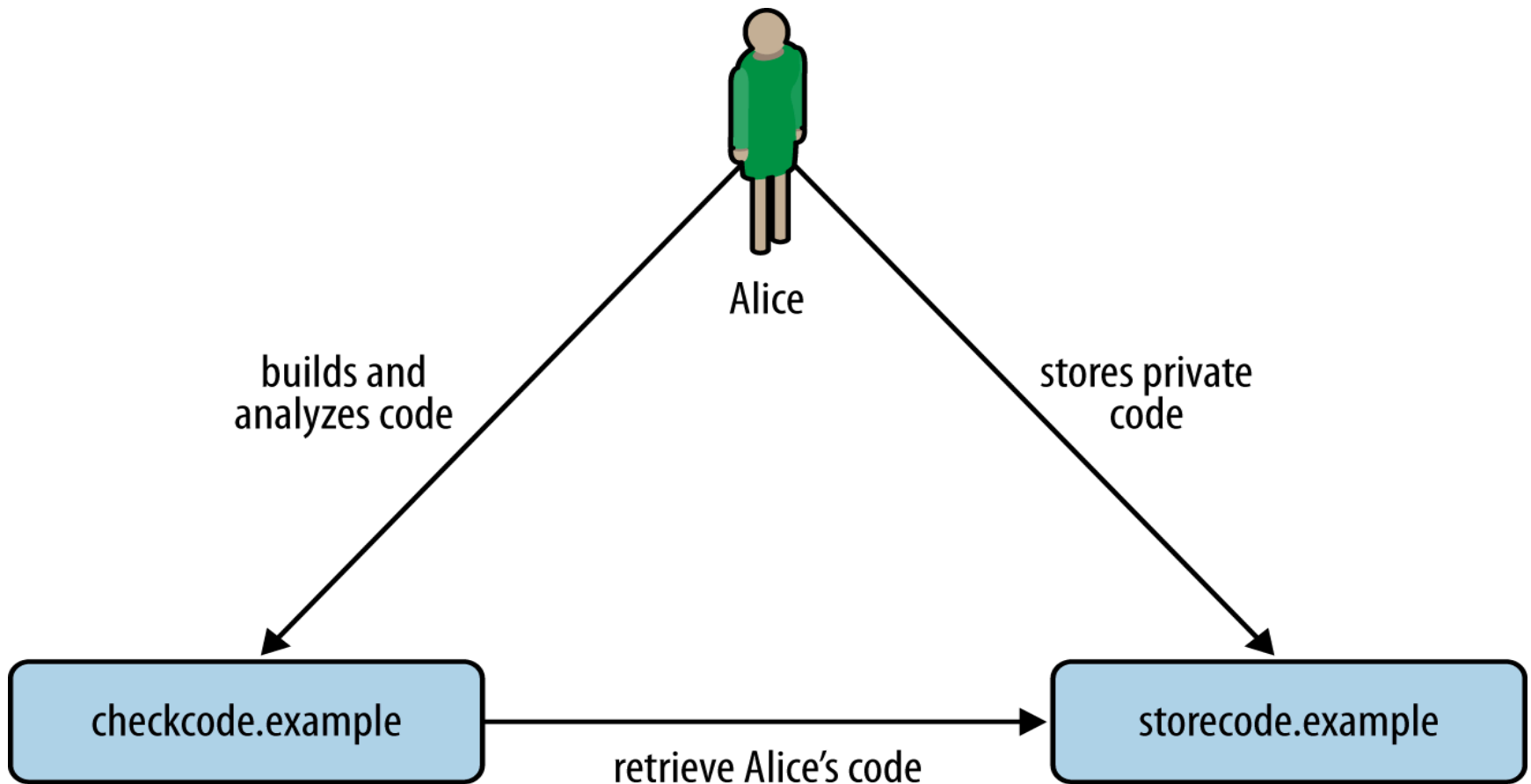


# OAuth Facebook



<http://developers.facebook.com/docs/concepts/login/login-architecture/>

# Ejemplo de escenario de uso de OAuth



# Roles OAuth 2.0 en el ejemplo

- *Alice* tiene el rol **resource owner**. Es la dueña (usuaria) y la encargada de dar acceso a los recursos protegidos
- *storecode.example* tiene el rol **resource server**. Es la entidad encargada de proporcionar el interfaz para acceder a los recursos protegidos.
- *checkcode.example* tiene el rol **client**. Es la aplicación que accede al recurso protegido en nombre del dueño o usuario.
- El cuarto rol es **authorization server**. Es el encargado de gestionar la autorización y los derechos de acceso. Si bien este rol lo suele jugar el *resource server*, OAuth 2.0 permite que sean roles separados.

# Aplicación cliente

- OAuth 2.0 permite el uso de diferentes tipos de cliente:
  - Aplicaciones web de servidor
  - Aplicaciones nativas, especialmente en móviles
  - Aplicaciones web cliente (usando Javascript)
- El servidor de autorización asigna un *client\_id* único a cada cliente. Algunos clientes también pueden recibir un *client\_secret* que permite autenticar a la aplicación frente al servidor de autorización. En este contexto tenemos dos tipos de cliente en OAuth 2.0:
  - Los clientes **confidenciales** son aquellos que pueden almacenar de forma segura el *client\_secret*. El ejemplo típico es una aplicación web de servidor, donde las credenciales están en el servidor.
  - Los clientes **públicos** son aquellos que no pueden almacenar de forma segura el *client\_secret*. Estos clientes solo tienen asignado un *client\_id*. Un ejemplo típico es una aplicación JavaScript.

# Registro de clientes

Antes de poder acceder a ningún recurso, los clientes deben registrarse en el servidor de autorización. En un escenario típico de OAuth 2.0 el dueño del cliente (desarrollador de la aplicación) proporciona la siguiente información:

- Información descriptiva, destinada a los usuarios, como nombre de la aplicación, logos, página web o información de la versión.
- Información técnica usada en el protocolo como URI de redirección los ámbitos de autorización.
- Los clientes se clasifican como confidenciales o públicos según la información proporcionada en el registro por el desarrollador.

# Ejemplo de registro de clientes en GitHub (2013)


Applications / **Register a new OAuth application**

Something users will recognize and trust

The full URL to your application's homepage

Your application's callback URL; read our [OAuth documentation](#) for more information

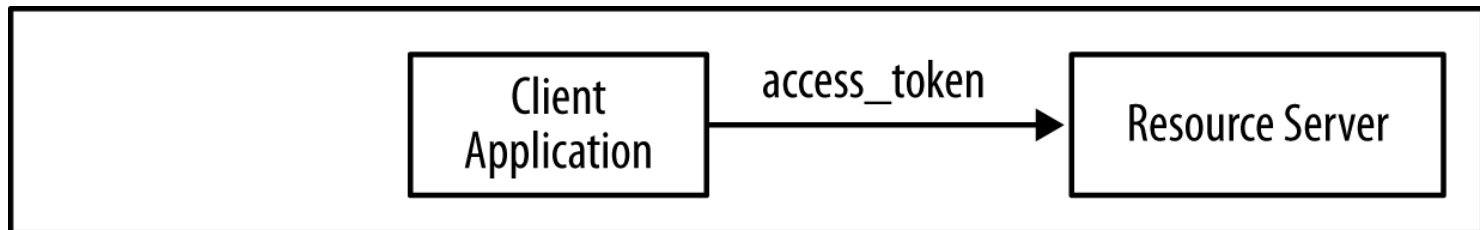
This is displayed to all potential users of your application

  
Drag & drop  
[or choose an image](#)



# Accediendo a recursos protegidos

- En OAuth 2.0 el cliente debe proporcionar un *access\_token* para poder acceder a un recurso protegido.
- Actualmente OAuth 2.0 solo permite el uso de “*Bearer*” *tokens*. Es decir, tokens que con solo adjuntarlos a los mensajes dan acceso al recurso y no hay que demostrar que somos los dueños.
- El uso de este tipo de token es la novedad de OAuth 2.0 frente a OAuth 1.0, y si bien hace mucho más simples los protocolos tiene desventajas de seguridad. En particular siempre se deben usar dentro de un canal de transporte seguro (SSL) y se debe asegurar que no se comparten con otros clientes.
- OAuth IETF esta trabajando en tokens basados en códigos MAC que permitan demostrar la posesión de un secreto compartido.



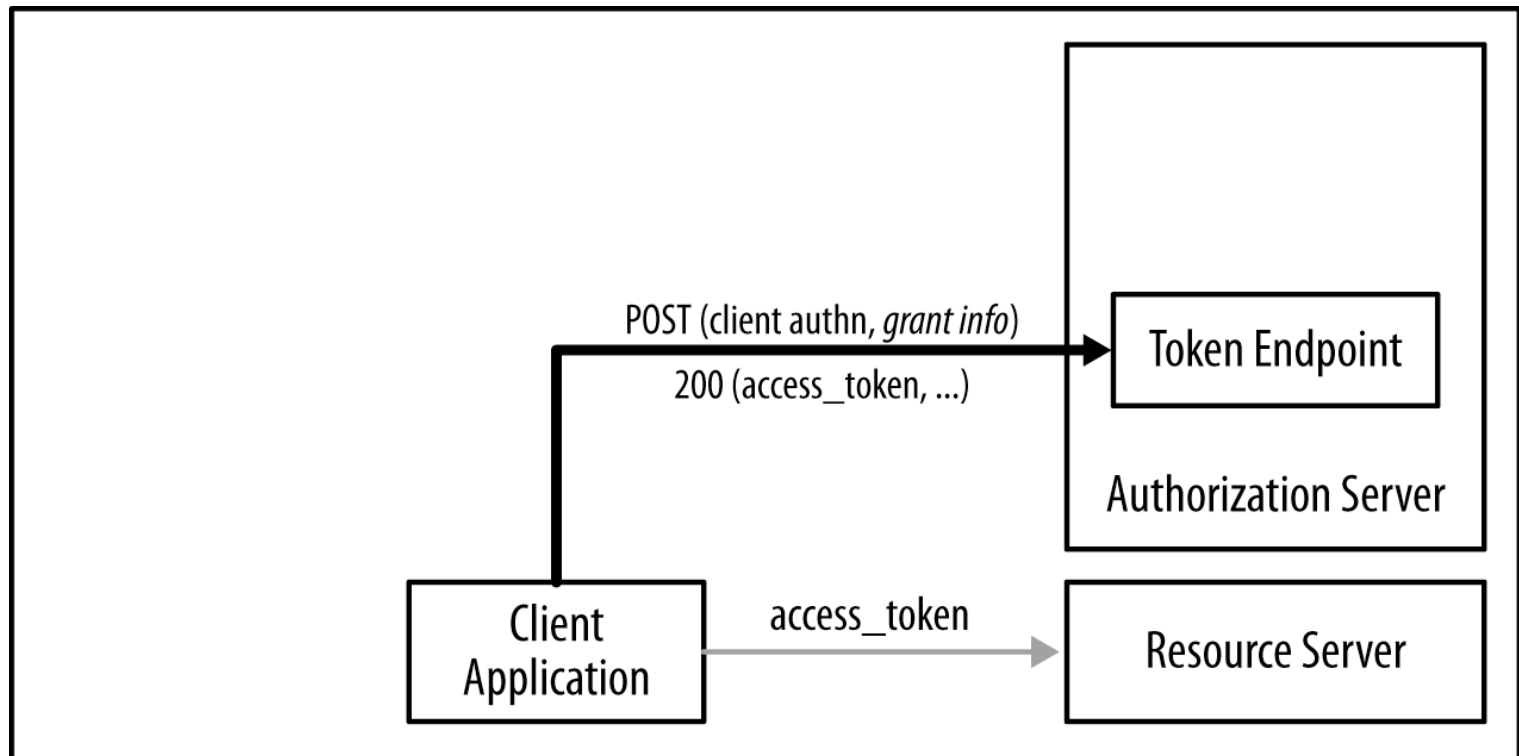
# Asociando token de acceso a peticiones HTTP

- La forma recomendada de asociar un token de acceso a una petición HTTP es usando la cabecera Authorization con el esquema Bearer:

```
GET https://storecode.example/resource HTTP/1.1  
Authorization: Bearer el.token.de.acceso
```

# Obteniendo los tokens de acceso

- La aplicación cliente consigue los tokens de acceso pidiéndolos al *token endpoint*, que es parte del servidor de autorización.
- La petición de token incluye un *authorization grant*, que es un concepto abstracto en el que se basa la concesión de derechos de acceso.



# Detalles técnicos

- La petición del token de acceso se realiza usando un POST a la URI del *Token Endpoint*. El cuerpo de la llamada se codifica como application/x-www-form-urlencoded y contiene el tipo de authorization grant y su valor. Cuando usamos un código de autorización el *grant type* es *authorization\_code* y el valor es el código de autorización:

```
POST https://authzserver.example/token_endpoint HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: authzserver.example
grant_type=authorization_code&code=the.authorization.code
```

- Si la petición es correcta la respuesta es application/json e incluye el valor del token de acceso, su tipo ( *bearer*), y la validez:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Cache-Control: private, max-age=0, must-revalidate

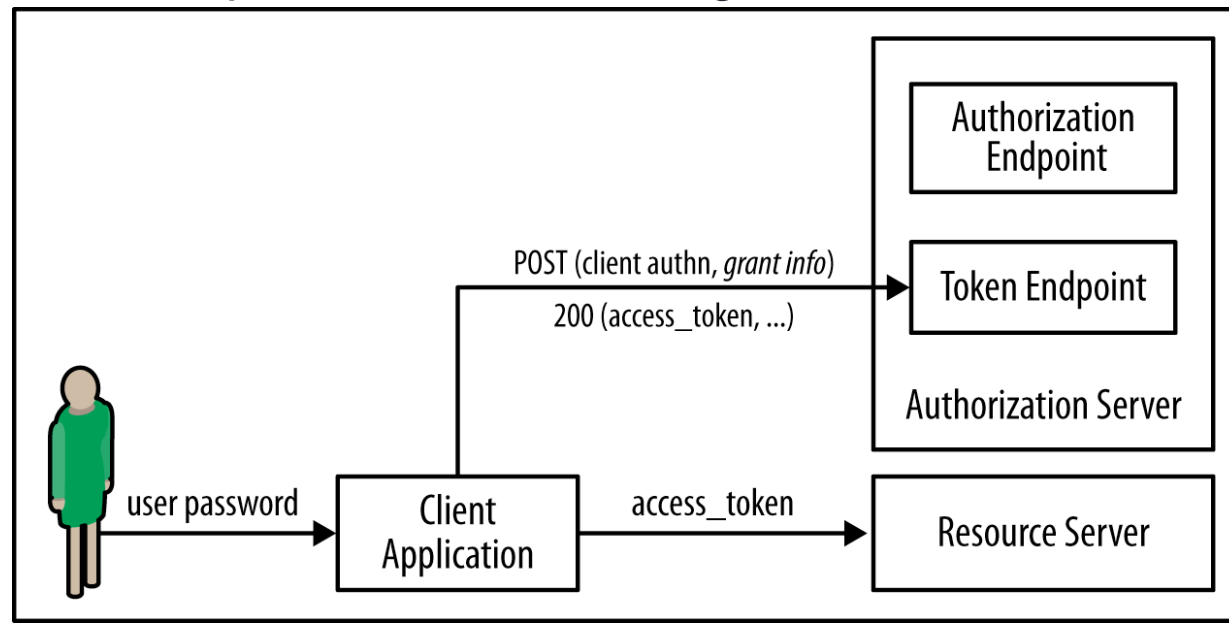
{"access_token":"the.access.token","token_type":"bearer",
"expires_in":3600, ... other info ...}
```

# Autenticación del cliente

- Si el cliente es confidencial (tiene un `client_secret`), la petición de token debe autenticar al cliente. Para ello OAuth 2.0 propone dos alternativas:
  - Usando el esquema de autenticación básica de HTTP, donde el `client_id` y el `client_secret` se utilizan como nombre de `usuario` y `contraseña` respectivamente.
  - Insertando el `client_id` y el `client_secret` como `parámetros` adicionales en el cuerpo de la petición POST

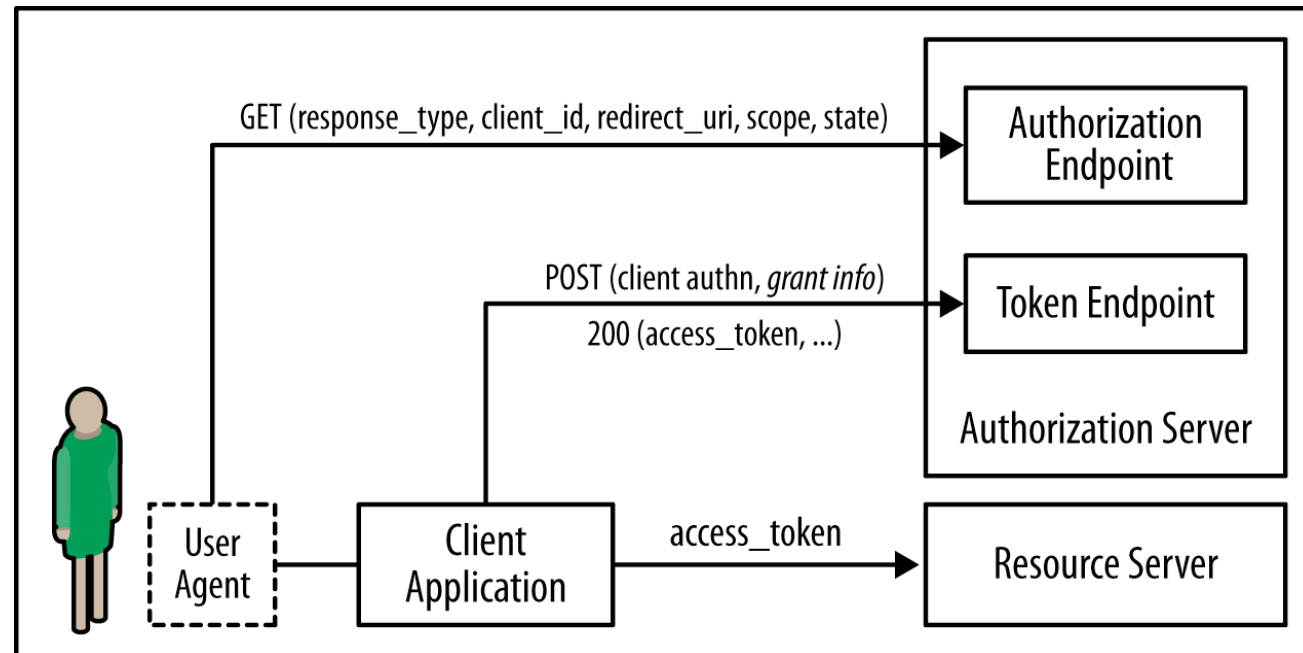
# Resource Owner Password Credentials Grant Flow

- En este caso el cliente utiliza directamente las credenciales del usuario para pedir el token de acceso.
- Tiene sentido en escenarios donde el cliente sea completamente confiable (entornos corporativos) o donde sea complicado usar otras opciones (aplicaciones móviles nativas).
- El password se puede eliminar una vez se ha conseguido el token de acceso. Si el usuario cambia la password el token sigue siendo válido



# Authorization Code Grant Flow

- Con esta opción el usuario puede delegar el acceso al cliente sin tener que pasarle sus credenciales. El usuario interactúa directamente con el authorization endpoint del servidor de autorización usando un user agent (p.e. Navegador web o web view).



# Detalles técnicos

- En un primer paso el cliente redirecciona el user agent al authorization endpoint e incluye en la petición los authorization request parameters
  - El parámetro *response\_type* define el tipo de authorization grant que se está pidiendo
  - El parámetro *scope* caracteriza el ámbito de la autorización que se está pidiendo.

```
https://authzserver.example/authorization_endpoint?  
client_id=the.client.id&  
scope=user+repo&  
state=crCMc3d0acGdDiNnXJigpQ%3d%3d&  
response_type=code&  
redirect_uri=https%3a%2f%2fclient.example%2fcallback&
```

CSRF



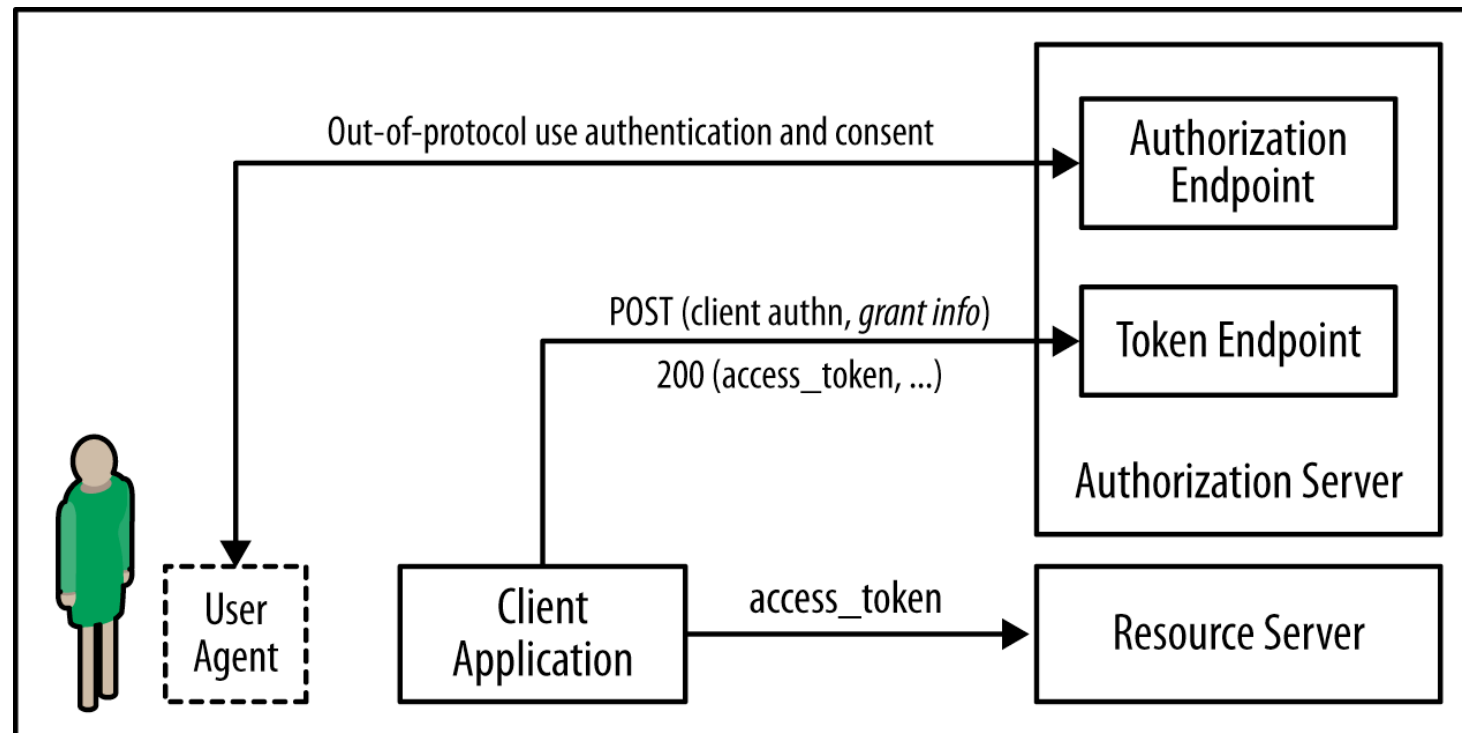
- Después de autenticar al usuario el servidor de autorización devuelve al usuario al cliente con el código de autorización *incluido* en la petición

```
https://client.example/callback?  
code=52...e4&  
state=cr...3D
```



# Autenticación frente al Servidor de Autorización

Al recibir la petición, el servidor de autorización inicia un protocolo propio (fuera de la especificación OAuth 2.0), con el objetivo de autenticar al usuario y pedir su consentimiento para el acceso los recursos por parte del cliente



# Refresh Tokens

- Si alguien captura un token de acceso puede acceder a los recursos del usuario por eso OAuth 2.0 utiliza los token de refresco para poder cambiar los token de acceso por unos nuevos.
- Cuando se cambia un authorization grant por un access token en el token endpoint, la respuesta puede contener también un refresh token.

```
HTTP/1.1 200 OK
```

```
Content-Type: application/json; charset=utf-8
```

```
Cache-Control: private, max-age=0, must-revalidate
```

```
{"access_token": "the.access.token", "token_type": "bearer",  
"expires_in": 3600, "refresh_token": "the.refresh.token"}
```

```
POST https://authzserver.example/token_endpoint HTTP/1.1
```

```
Content-Type: application/x-www-form-urlencoded
```

```
Host: authzserver.example
```

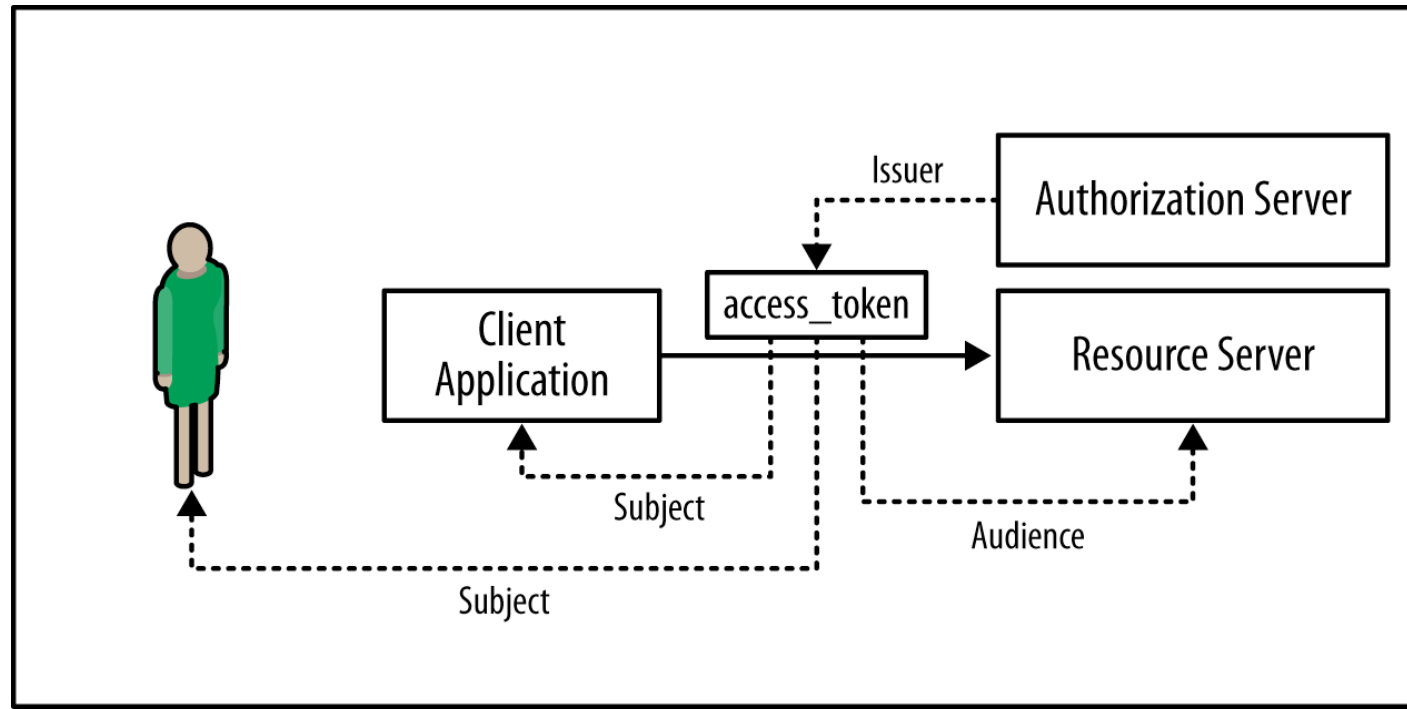
```
grant_type=refresh_token&  
refresh_token=the.refresh.token
```

# Access Token usando JSON Web Token

```
{  
  "exp": 1379284015,  
  "aud": "http://resourceserver.example",  
  "iss": "http://authzserver.example",  
  "role": [  
    "fictional_character",  
    "student"  
  ],  
  "client_id": "client2",  
  "scope": [  
    "scope1",  
    "scope2"  
  ],  
  "nbf": 1379280415,  
  "sub": "Alice"  
}
```

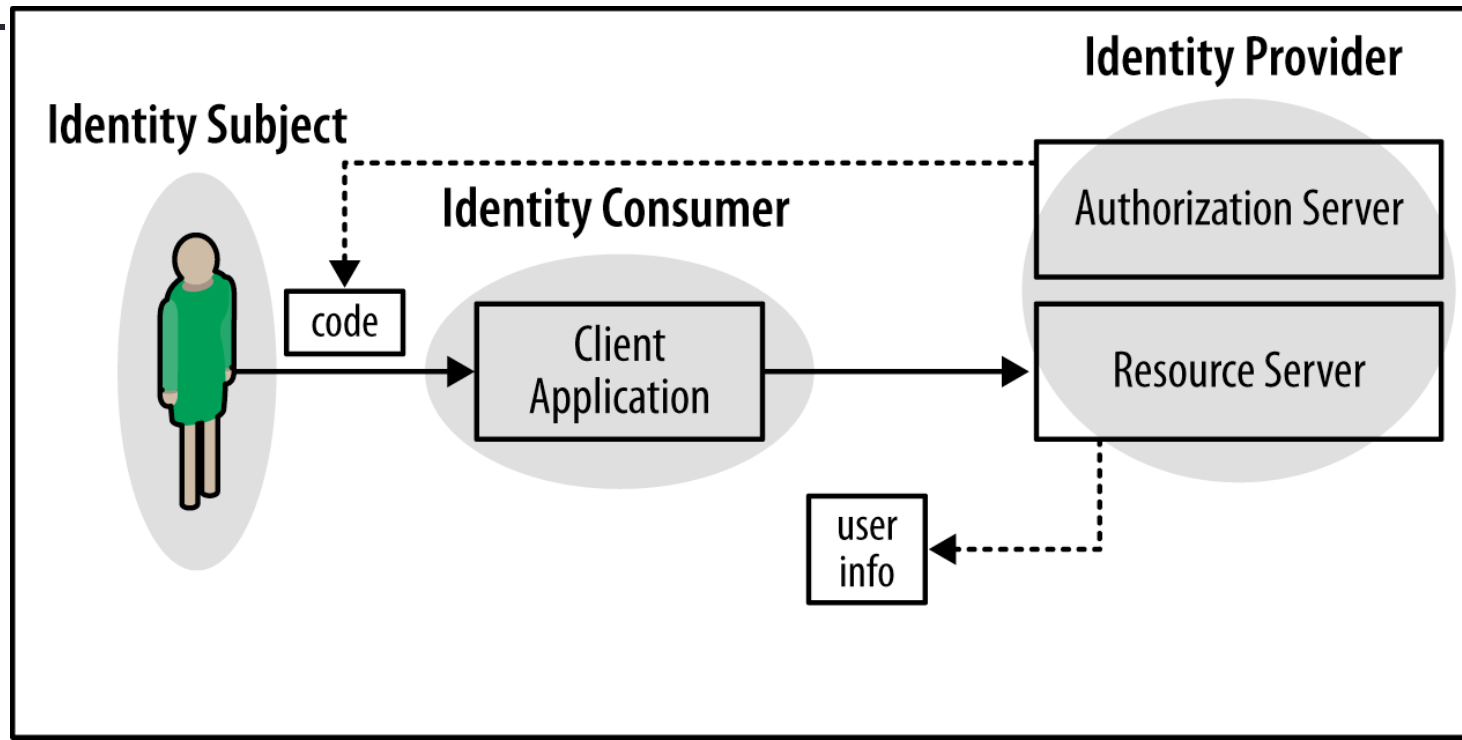
# Access Token como mecanismo de autenticación

- Las peticiones hechas por los clientes a los recursos contienen un token de acceso. El objetivo principal de ese token es probar al recurso que la petición está autorizada por el usuario para acceder al recurso protegido. Por tanto se autentica a la aplicación cliente y al usuario y se define el ámbito de la autorización.



# Autenticación con OAuth 2.0

- OAuth 2.0 se puede usar también para otro tipo de autenticación: El usuario frente a la aplicación cliente.
- La idea es usar recursos específicos con información sobre el usuario. De esta forma el cliente puede acceder a los datos de identidad del usuario de forma verificada por el servidor de autorización.



# OpenID Connect – *id\_token*

- OpenID Connect implementa este perfil de autenticación, para ello añade al token de acceso un *id\_token*

```
{ "access_token" : "ya..8s",  
  "token_type"   : "Bearer",  
  "expires_in"   : 3599,  
  "id_token"     : "eyJ..0Q" }
```

- El *id\_token* es un JWT firmado (JWS) que contiene la información de identidad del usuario

```
{  "sub": "104107606523710296052",  
    "iss": "accounts.google.com",  
    "email_verified": "true",  
    "at_hash": "G_...hQ",  
    "exp": 1380480238,  
    "azp": "55...ve.apps.googleusercontent.com",  
    "iat": 1380476338,  
    "email": "alice4demos@gmail.com",  
    "aud": "55...ve.apps.googleusercontent.com" }
```

# VULNERABILIDADES WEB Y PRUEBAS DE INTRUSIÓN

---

# Validación de la Entrada

- Éste tipo de ataque se produce en las siguientes circunstancias:
  - Una aplicación falla al reconocer una sintaxis de entrada incorrecta.
  - Un módulo de una aplicación acepta un tipo de entrada que no debería o bien falla al manejar campos vacíos en la entrada (format string attack).
- Este tipo de vulnerabilidad abre la puerta a otros ataques como
  - SQL injection
  - Ejecución de comandos del SO
  - Cross-site scripting



# SQL Injection

- Esta vulnerabilidad se da al construir consultas SQL dinámicas usando parámetros de entrada no validados.
- Se pueden alterar las consultas, modificar o borrar registros en la base de datos e incluso ejecutar comandos del sistema
- Ejemplo:

- Usuario: Pepe, Contraseña: Pepe123

- Consulta Honesta:

- Parametros user=Pepe, pass=Pepe123

```
"SELECT * FROM usuarios WHERE user = '"+user+"'" AND pass = '"+pass+"'" ;  
SELECT * FROM usuarios WHERE user = 'Pepe' AND pass = 'Pepe123'
```

- Devuelve el registro del usuario Pepe

- Consulta Maliciosa:

- Parametros user=' OR '1' = '1, pass = ' OR '1' = '1

```
"SELECT * FROM usuarios WHERE user = '"+user+"'" AND pass = '"+pass+"'" ;  
SELECT * FROM usuarios WHERE user = ' OR '1' = '1' AND pass = ' OR '1' = '1'
```

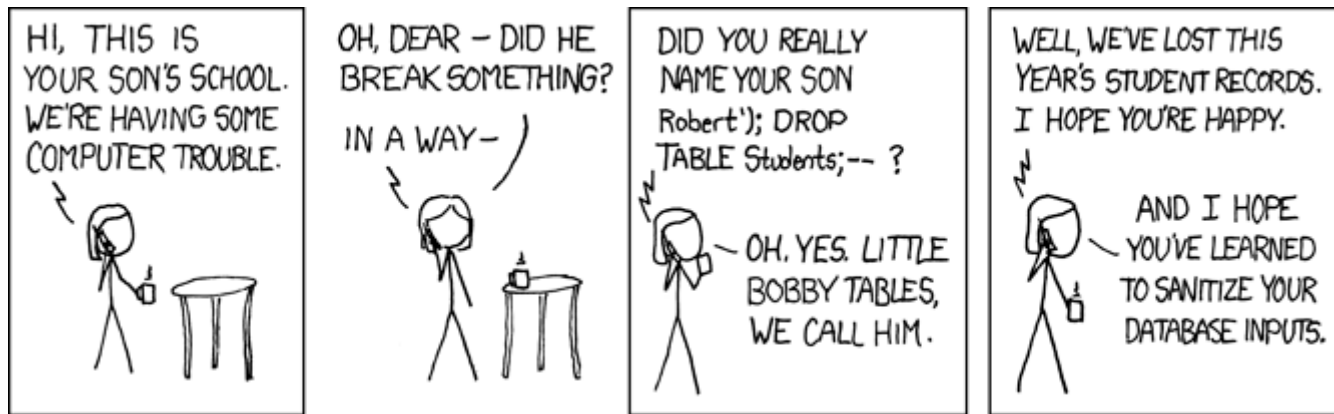
- Devuelve todos los usuarios de la tabla!!!

# SQL Injection

- `parametros pass = ' OR '1' = '1'; DROP TABLE users; --`

```
SELECT * FROM usuarios WHERE user = '' OR '1' = '1'  
AND pass = '' OR '1' = '1'; DROP TABLE users;
```

- Los guiones – eliminan la comilla simple final



<http://xkcd.com/327/>

# SQL Injection - Soluciones

- Consultas parametrizadas precompiladas
  - Usar Frameworks intermedios para acceder a la base de datos
  - Usar procedimientos almacenados
  - Evitar caracteres no alfanuméricos
  - Validar la entrada mediante whitelists
  - Validar la entrada mediante blacklists
  - ...
- 
- No hay una única solución

# Ejecución de comandos del SO

- Ejemplo PHP de código vulnerable

```
$userName = $_POST["user"];  
$command = `ls -l /home/' . $userName;  
System($command);
```

- El resultado esperado es un listado el directorio home del usuario pasado en el formulario web, pero si se envía un nombre de usuario como el siguiente,

```
;rm -rf/
```

- Se ejecutaría el siguiente comando,

```
Ls -l /home/;rm -rf /
```

- **Borrando todos los archivos del sistema de ficheros!!**

# Cross Site Scripting (XSS)

- Esta vulnerabilidad se da al construir páginas web dinámicas usando parámetros de entrada no validados.
- Como consecuencia un atacante podría modificar el aspecto de la web, transferir información sensible, interceptar la sesión, etc.
- Ejemplo:

test.jsp      `<% out.println("Hola "+request.getParameter("name")) ; %>`

Petición web **Honesta**

<http://www.servidor.es/test.jsp?name=Pepe>

```
<HTML>
<Body>
Hola Pepe
</Body>
</HTML>
```

Petición web **Maliciosa**

[http://www.servidor.es/test.jsp?name=<script>alert\("Attacked"\)</script>](http://www.servidor.es/test.jsp?name=<script>alert('Attacked')</script>)

```
<HTML>
<Body>
Hola <script>alert("Attacked")</script>
</Body>
</HTML>
```

# Cross Site Scripting (XSS) - Soluciones

- Filtrado de entrada (blacklist/whitelist)
- Filtrado de salida (blacklist/whitelist)
- Librerías de codificación de salida
  - Anti-XSS de Microsoft
  - Módulo de codificación OWASP ESAPI
- Reglas de codificación en función del contexto
- ...

# Cross Site Request Forgery (CSRF)

- El atacante puede hacer que el navegador realice peticiones web no deseadas en nombre del usuario que tiene la sesión iniciada.
- El ataque suele consistir en incluir en la página web una imagen o un marco apuntando a un sitio donde el usuario debe iniciar sesión para aprovechar que ya tiene la sesión abierta y que el navegador va a enviar las **cookies**
- Para que funcione el ataque se tienen que dar estos condicionantes:
  - Que la aplicación web atacada no compruebe las cabeceras HTTP Referer o HTTP Origin.
  - El atacante debe encontrar un formulario en la aplicación que quiere atacar y encontrar los valores correctos para los campos
  - La víctima tiene que tener la sesión abierta mientras es atacada

[http://www.youtube.com/watch?v=uycmHQM\\_h64](http://www.youtube.com/watch?v=uycmHQM_h64)

# Ej: Cross-Site Request Forgery

Alice quiere enviar dinero a Bob, a través de una plataforma web:

<http://bank.com/transfer.do?acct=BOB&amount=100>

María se da cuenta de que puede fabricar una petición en su beneficio:

<http://bank.com/transfer.do?acct=MARIA&amount=1000000>

María solo tiene que engañar a Alice para que haga click en un enlace:

```
<a href=http://bank.com/transfer.do?acct=MARIA&amount=1000000>Haz click aquí!</a>
```

O mejor aún, puede esconder la petición en una imagen de un fichero HTML:

```

```



# PRIVACIDAD Y ANONIMATO

---

# Privacidad

- La Privacidad puede definirse como:
  - El derecho de los individuos y entidades de proteger, salvaguardar y controlar el acceso, almacenamiento, distribución y uso de información sobre su propia persona
  - Confidencialidad no es equivalente a privacidad
  - La confidencialidad es relativa a los datos mientras que la privacidad es relativa a las personas
- ¿Qué información es necesario proteger?
  - Dependerá de lo que el usuario considere información privada. Además, también es relevante de quién se protege la información
    - Por lo general: identidad, localización, preferencias, ...
- A medida que aumenta el número de transacciones realizadas por los sistemas de información, las amenazas a la privacidad aumentan

# Anonimato

- Una definición de Anonimato:
  - “Es el estado de no ser identificable entre un grupo de sujetos, conocido como conjunto de anonimato”
- Las técnicas de anonimato tratan de hacer indistinguible a un individuo entre un conjunto suficiente de identidades con unas características similares
- Otros conceptos relacionados
  - **Unlinkability**: se refiere a la incapacidad de un atacante para relacionar dos sucesos o entidades observadas
  - **Unobservability**: se refiere a la imposibilidad de distinguir la presencia de sucesos.
    - No es posible distinguir si algún emisor o receptor envían o reciben mensajes
  - **Pseudoanonimato**: se refiere al uso de pseudónimos en lugar de los identificadores normales con el fin de proporcionar anonimato

# Mecanismos de prevención

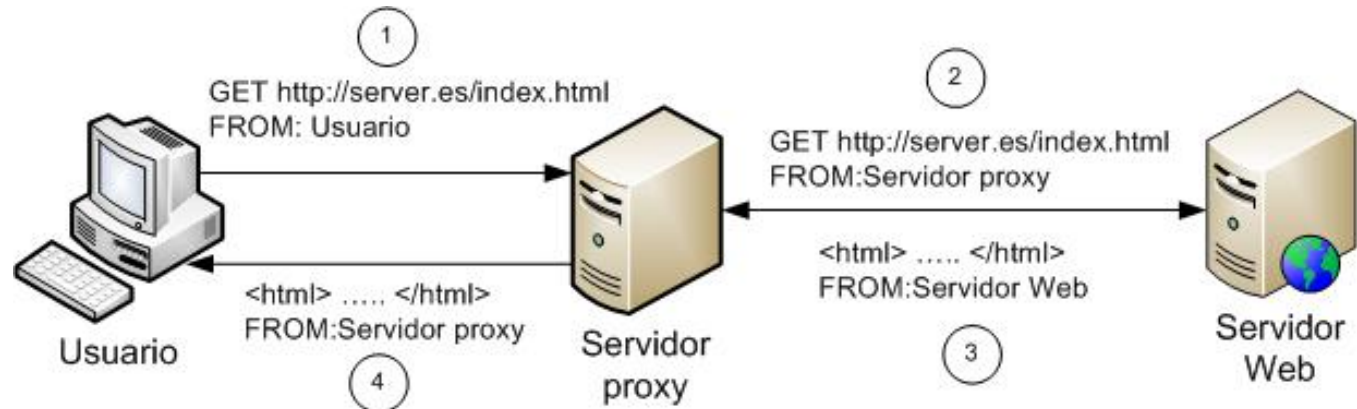
- Propuestas para proteger la privacidad de los usuarios
  - **Protocolos criptográficos y de enrutado**
    - Se utilizan para evitar que un atacante pueda acceder al contenido de los mensajes intercambiados así como para evitar que la dirección de red o el camino que siguen los paquetes puedan identificar a las partes comunicantes
  - **Técnicas de ofuscación**
    - Son mecanismos basados principalmente en la generalización o supresión de información para limitar la precisión de la información que se revela
  - **Técnicas de anonimato**
    - Tratan de impedir que un atacante conozca la identidad del usuario, para lo cual será necesario hacerlo indistinguible entre un número de individuos suficientemente grande

# Protocolos criptográficos y de enrutado

- Tratan de proteger el tráfico frente a entidades que se dedican a observar las comunicaciones
- Contramedidas básicas
  - Comunicaciones seguras utilizando técnicas criptográficas
  - Evitar el reconocimiento de la dirección del cliente
    - No sólo es necesario proteger la dirección del cliente frente a terceras partes, sino que en ocasiones también es necesario protegerla frente al propio proveedor de servicio

# Proxy

- Un servidor proxy hace de intermediario de la comunicación, acepta conexiones de los clientes y las reenvía.

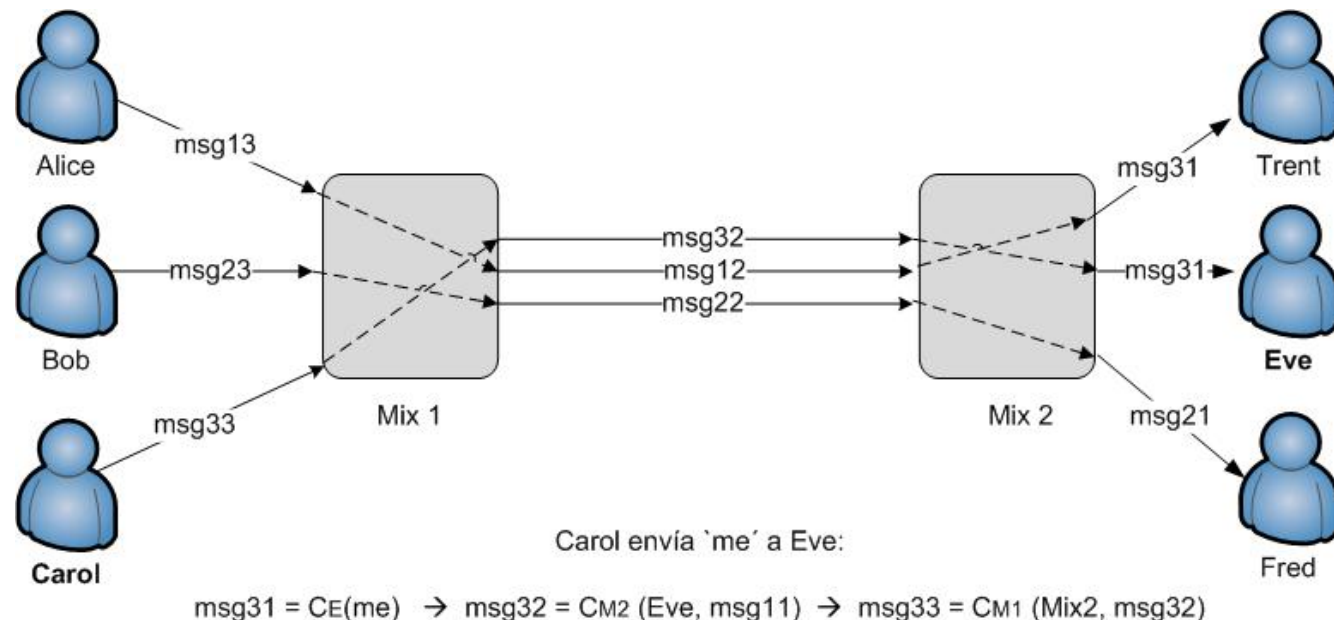


- El cliente solicita una página web al proxy y este origina una nueva petición, almacenando internamente el cliente y la petición que realizó para reenviarle la respuesta.
  - Se deben eliminar cualquier información de identidad de los datos del usuario y usar conexiones SSL entre el usuario y el servidor proxy
  - Cuando un usuario solicita una página web el proxy se puede usar una cache para guardar las peticiones junto a las de otros usuarios y así evitar que haya un orden prefijado en las peticiones.

# Mixers

Son dispositivos de almacenamiento y envío. Para ello el usuario cifra el mensaje en capas y lo envía a través de un conjunto de mixers determinado. Estos lo almacenan durante cierto tiempo con el fin de que pueda ser mezclado con otros mensajes recibidos, saliendo del mixer en un orden diferente

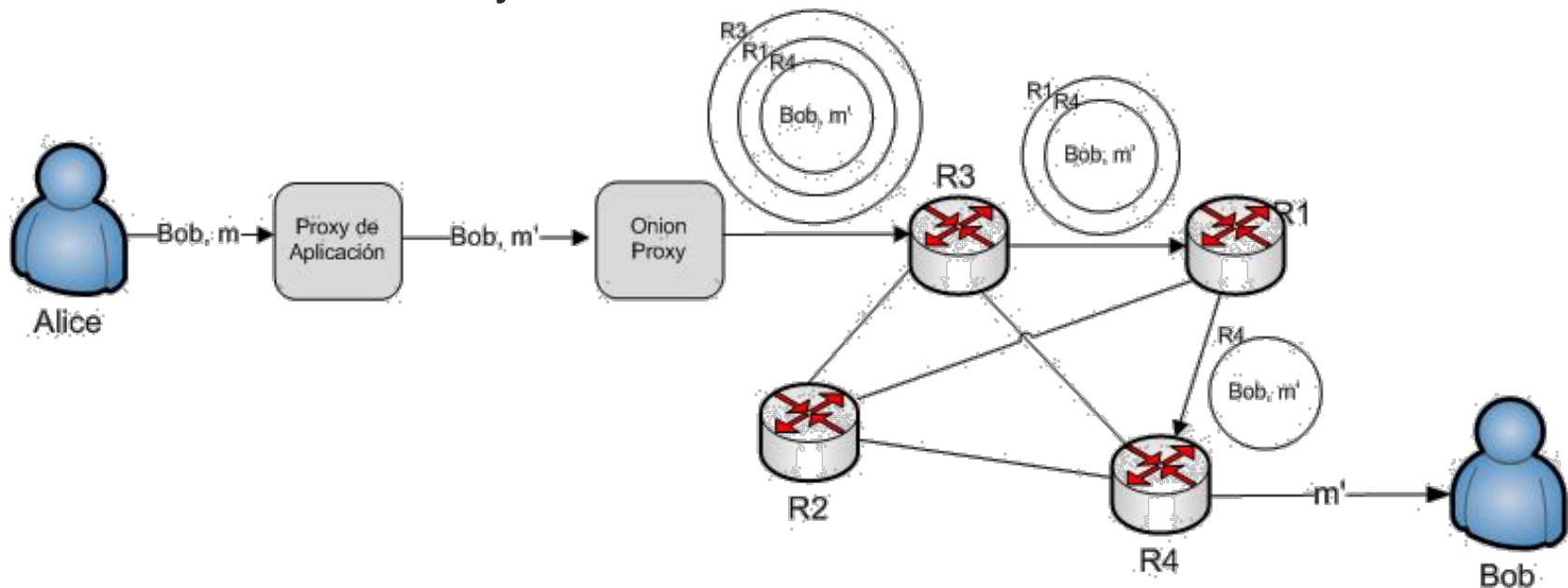
- Este esquema funciona sólo si el número de mensajes almacenados temporalmente por el mixer es lo suficientemente grande



# Onion Routing

Es un sistema similar al de mixnets orientado a la conexión, donde un proxy de aplicación filtra la información sensible a nivel de aplicación. Una vez filtrado el mensaje, el onion proxy crea un paquete cifrado en capas, que se irán pelando a medida que atraviese el camino de onion routers.

- **TOR** añade control de congestión, comprobación de la integridad y Perfect Forward Secrecy, entre otras funcionalidades.





# TOR

Tor: anonimato online - Mozilla Firefox

Archivo Editar Ver Historial Marcadores Herramientas Ayuda

https://www.torproject.org/

Más visitados Comenzar a usar Firefox Últimas noticias Master WSN Background Busqueda Multimedia

**Tor** Principal Perspectiva Descarga Documentación Colabora Gente Blog ¡Dona!

## Tor: Anonimato online

Tor es un proyecto software que le ayuda a defenderse contra el [análisis de tráfico](#), una forma de vigilancia de la red que amenaza la libertad personal y la privacidad, la confidencialidad en los negocios y relaciones, y la seguridad del estado. Tor le protege transmitiendo sus comunicaciones a través de una red distribuida de repetidores llevados por voluntarios de todo el mundo: evita que alguien que observa su conexión a Internet aprenda qué sitios visita, y evita que los sitios que visita aprendan su posición física. Tor funciona con muchas de sus aplicaciones existentes, incluyendo navegadores web, clientes de mensajería instantánea, acceso remoto, y otras aplicaciones basadas en el protocolo TCP.

Cientos de miles de personas de todo el mundo usan Tor para una gran variedad de razones: periodistas y bloggers, trabajadores por los derechos humanos, fuerzas del

Download Tor

Alice

Paso 2: El cliente Tor de Alice elige un camino aleatorio al servidor de destino. Los enlaces verdes están encriptados, los enlaces rojos van en claro.

Jane

Dave

Bob

Cómo funciona tor, pulse aquí para más información

**Apoya Tor: ¡dona!**

Terminado Una descarga activa (7 minutos restante(s)) www.torproject.org 150.214.56.135 Tor Desactivado

Panel de Control de Vidalia

Estado

Tor no está ejecutándose

Atajos de Vidalia

Start Tor

Ajustar Retransmisión

Ver la Red

Usar una Nueva Identidad

Gráfica de Ancho de Banda

Ayuda

Acerca de

Informe de Mensajes

Preferencias

Salir

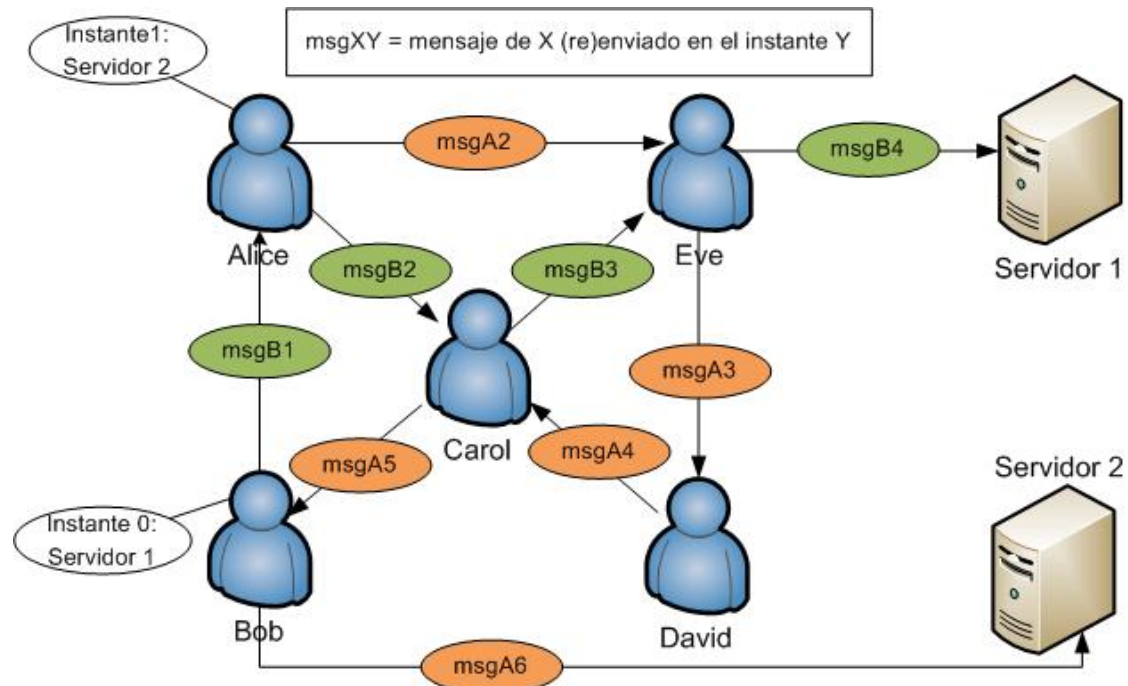
☒ Mostrar esta ventana al iniciar

Esconder

Dirección IP real

# Soluciones basadas en grupos

- Los emisores se agrupan creando una “multitud” en la que todos enrutan de manera aleatoria los paquetes recibidos. Cada nodo sólo conoce el destino y el nodo del que recibe el paquete. Los nodos comparten claves con sus vecinos para cifrar los mensajes



- El camino seguido por el mensaje es almacenado por un tiempo para saber enrutar de vuelta posibles respuestas