Universidad de Buenos Aires
Facultades de Ciencias Económicas,
Cs. Exactas y Naturales e Ingeniería
Carrera de Especialización en Seguridad Informática

Trabajo Final

Tema:
Navegación anónima

Título:
TOR, anonimato en internet

Autor: Ing. Harold David Chaparro Zuñiga

Tutor: Mg. Ing. Juan Alejandro Devincenzi

Año 2015

Cohorte 2013

Declaración Jurada de origen de los contenidos

“Por medio de la presente, el autor manifiesta conocer y aceptar el Reglamento de Trabajos Finales vigente y se hace responsable que la totalidad de los contenidos del presente documento son originales y de su creación exclusiva, o bien pertenecen a terceros u otras fuentes, que han sido adecuadamente referenciados y cuya inclusión no infringe la legislación Nacional e Internacional de Propiedad Intelectual”

RESUMEN

Palabras clave: Tor, SSL/TLS, anonimato, privacidad, censura.

Es evidente que día a día pasamos más tiempo interactuando con medios tecnológicos integrando el servicio de internet en toda actividad diaria, ligando nuestras identidades a las consultas, opiniones y publicaciones que realizamos en esta red.

Ya sea con fines académicos, periodísticos o políticos el anonimato es una poderosa herramienta para la distribución de información y libre expresión. Existen países en los cuales el seguimiento en internet, el análisis de tráfico y la censura de contenidos en base a esta información se realiza de forma legal. Por esta razón TOR brinda hoy día la posibilidad para aquellos que desea conservar un nivel de privacidad en internet y para aquellos que se les impide desarrollar y compartir sus ideales ofreciendo un medio de comunicación y difusión seguro sobre una red insegura como lo es internet.

TABLA DE CONTENIDOS

Lista de Figuras

Lista de Tablas

ABREVIATURAS

a|b: a concatenado con b

COUNTERLEN: tamaño del contador empleado en el algoritmo AES-CTR

DH: Diffie–Hellman

E(K,s): cifrado utilizando AES-CTR-128bits de la cadena s con la clave K

HASH_LEN: el tamaño de salida de una función hash

H(X): valor hash de la cadena de caracteres X

H(x,t): función HMAC del mensaje x con la clave t

K: clave simétrica

KEYLEN: tamaño de una llave K

MAC: Message Authentication Code

Nodo: OP o OR de acuerdo al escenario dentro de la red TOR

OP: Onion Proxy

OR: Onion Router

PK: Clave Pública

SK: Clave Privada

s[:n]: los primeros n bytes de la cadena s

s[n:]: los últimos n bytes de la cadena s

WR(n): n bytes de un generar de números aleatorio

1. INTRODUCCION

"SSL/TLS es un protocolo que brinda confidencialidad empleando claves simétricas e integridad controlada por el uso de MACs en los mensajes permitiendo que dos usuarios TCP puedan establecer los mecanismos de seguridad para entablar una comunicación segura".**[1]**

Tor es una red superpuesta a internet en la que un cliente construye un circuito por medio del protocolo SSL/TLS el cual permite enviar información a través de internet de forma cifrada con claves previamente establecidas con cada nodo agregando así una capa de cifrado para cada punto de enrutamiento, de aquí el nombre de enrutamiento cebolla. Debido a su diseño Tor logra que cada nodo solo conozca su predecesor y sucesor en este circuito logrando enviar tráfico a un destino sin exponer su origen.

El presente trabajo expone el funcionamiento del protocolo SSL/TLS aplicado en la creación de canales seguros y autenticación entre los nodos de la infraestructura de la red Tor. Pretende dar una visión de las problemáticas que ha enfrentado esta red a lo largo de los años y por ultimo comprender la metodología y el diseño de este protocolo para ofrecer anonimato a nivel de red en las consultas que realizamos diariamente en internet

2. PROTOCOLO SSL/TLS

2.1 Arquitectura del protocolo

El protocolo SSL/TLS permite crear un túnel de comunicación seguro mediante el uso de algoritmos criptográficos. Consta de dos etapas. La primera es denominada “SSL handshake protocol” en la que se inicia la negociación enviando un conjunto de parámetros que permiten configurar las medidas de seguridad para la transmisión, el formato de los datos y la generación de claves para el cifrado antes de enviar cualquier bit de información, y luego tenemos el “SSL record protocol“, encargado de recibir los datos de las capas superiores para así fragmentarse y configurarse en el formato establecido por el protocolo para ser enviados.

SSL/TLS permite agregar las siguientes características a una conexión:

Privacidad: Los datos son cifrados antes de ser enviados por medio de algoritmos criptográficos definidos en el SSL/TLS handshake.

Autenticación: Una de las partes de la comunicación debe autenticarse para poder establecer la identidad y confiar en este para el envío de datos.

Integridad: Los mensajes incluyen un MAC que permite verificar si este no ha sido manipulado antes de alcanzar el destino final.

El SSL/TLS handshake protocol es una de las etapas más críticas en el protocolo Tor ya que es un de las pocas donde se intercambia información entre nodos de la red en texto plano como el mensaje client_hello y el intercambio de certificados lo cual ha llevado a plantear un gran número de propuestas desde su creación hasta el día de hoy con el fin de minimizar el grado de exposición y de plantear un comportamiento que permita aumentar la similitud entre el protocolo TLS y TLS sobre Tor.

2.2 Cipher suite

Es un conjunto de algoritmos que se intercambian en la fase del handshake SSL/TLS, los cuales serán utilizados al momento de configurar y definir las características de seguridad en una conexión entre un cliente y un servidor. La suite está compuesta por 3 algoritmos:

- Algoritmo de autenticación
- Algoritmo de cifrado que incluye el modo si es cifrado por bloques
- MAC

3. TOR

3.1 Componentes

La infraestructura de la red Tor está formada por nodos con diferentes funciones: Los llamados “OP o onion proxy” que sirven como punto de entrada a la red, los “OR o onion routers, también conocidos como Tor-Relays” que son los nodos encargados de transmitir la información enviada a través de ellos hasta alcanzar el destino final y por último los “servidores de directorio” que se encargan de firmar y publicar la información de cada nodo utilizada por los OP en la creación de circuitos.

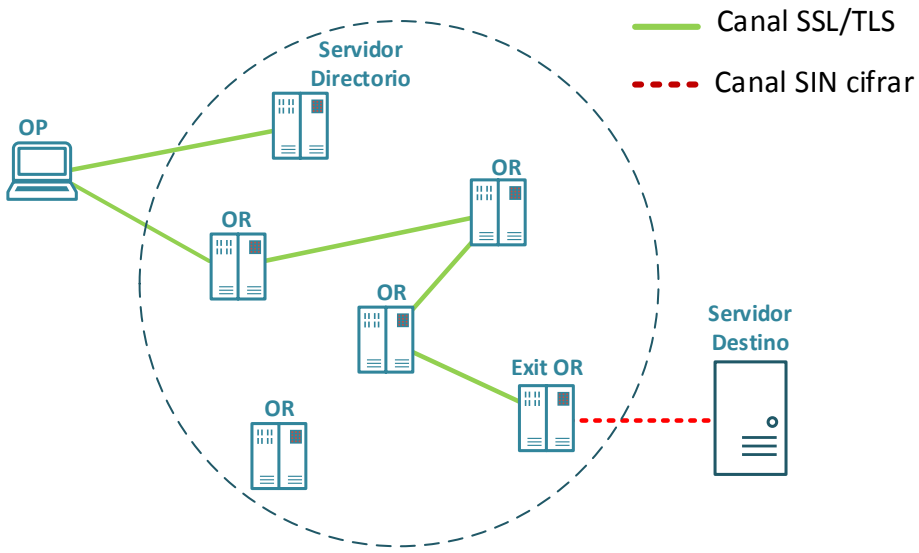

Figura 1. Infraestructura de la red Tor

El circuito por el cual se envía la información está construido por mínimo 3 OR. Los primeros dos denominados Middle-OR o Bridge-OR de acuerdo a su función y por último un Exit-OR antes de llegar al destino encargado de entregar el paquete al servidor que va dirigido.

Middle-OR: Nodos listados en los servidores de directorio, comúnmente reciben y pasan el tráfico a través del circuito.

Exit-OR: Este es el último nodo por el cual pasa el tráfico antes de llegar al destino, también se encuentra en los servidores de directorios para

ser empleados por los OP al momento de definir el punto de salida del circuito.

Bridge-OR: Este tipo de nodo no se encuentra publicado ni advierte su presencia en la red ya que su función es servir como entrada a la red Tor en países donde las IP de los OR listados por Tor se encuentran bloqueadas.

Cada OR posee múltiples pares de claves públicas y privadas:

Identity-key: PK utilizada para firmar documentos como los certificados de conexión e identidad utilizados en el handshake SSL/TLS además del "router-descriptor" que contiene información referente al nodo. Esta archivo se encuentra listado en los servidores de directorio.

Onion-key: PK utilizada para descifrar las capas del paquete al extender un circuito. Al momento de extender un circuito se envían un conjunto de datos en los payloads de las CELL el cual es cifrado con esta clave.

Connection-key: PK utilizada para negociar las conexiones TLS. (Cambia al menos una vez al día). Esta PK es enviada dentro del certificado de concesión SSL/TLS y posteriormente empleada para negociar las clave de sesión K.

Más adelante se explicará el uso y la fase en la que se emplea cada una de estas claves

A su vez se realizan varias operaciones de cifrado y hash empleando los siguientes algoritmos:

- Cifrador de llave pública: RSA 1024 con exponente fijo 65537.

- Cifrador de llave privada: AES 128 en modo CTR[1].
- Algoritmo de hash: SHA1.

3.2 Autoridades de directorio/ Servidores de directorio

La red cuenta con un grupo limitado de servidores oficiales que son distribuidos por defecto en el software Tor. Estos servidores ofrecen una lista de los ”network-status”, archivo que contiene el estado e información de conexión para cada nodo de la red (hash de su Identity-key, hash del router-descriptor, direcciones IP, puertos, etc) en la que se basa un OP o un OR para la creación y configuración de un circuito, además funciona como punto de información para validar las claves de cada nodo. Existen ORs que sirven como servidores de directorio cache con el fin de lograr redundancia de esta información y mejorar tiempos de respuesta.

Las Autoridades de directorio cuentan con una “Authority-Identity-key”, una de las claves más sensibles y críticas de la red ya que estas sirven para firmar los certificados de las “Authority-signing-key” de cada servidor, a su vez empleadas para firmar documentos como el router-descriptor.

3.3 Paquete CELL

Este paquete es el medio de comunicación entre OP-OR y OR-OR. Puede llevar tanto información del cliente como parámetros de configuración para el circuito. Es enviado de forma cifrada dentro del campo SSL/TLS records y está formado de acuerdo a la versión del handshake SSL/TLS utilizado para establecer la conexión entre los nodos y su estructura se define de la siguiente manera.

[1] Cifrado de bloque en modo contador utilizando una cadena generada aleatoriamente y un contador como entrada inicial que incrementa en 1 por cada bloque. Ambos son concatenados dando lugar a una cadena del mismo tamaño del bloque de texto a cifrar. Posteriormente son cifrados con AES utilizando la clave definida y por último se realiza la operación XOR entre esta cadena y el bloque de texto plano para dar lugar al bloque cifrado.

Estructura CELL en handshake Versión 1:

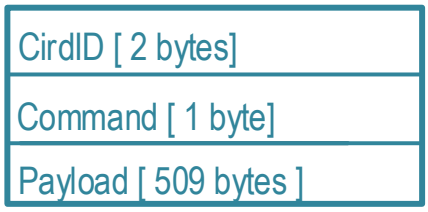

Figura 2. Estructura paquete CELL de tamaño fijo

Esta CELL es de tamaño fijo (512 bytes) y cumple una función de acuerdo al valor del campo “Command”. Los posibles valores para las CELL de tamaño fijo son los siguientes:

Valor	Comando	Función
0	PADDING	Permite mantener un circuito abierto cuando no hay tráfico de datos entre los nodos.
CELL de administración y configuración		
1	CREATE	Permite crear el circuito y definir las claves compartidas con cada OR.
2	CREATED	Permite confirmar la creación correcta del circuito por medio de CREATE CELLs.
3	RELAY	Permite transmitir información a través del circuito.
4	DESTROY	Permite cerrar un circuito.
7	VERSIONS	Permite negociar la versión del protocolo de enlace entre dos nodos.
8	NETINFO	Utilizada para enviar información de conexión (dirección IP de un nodo, puerto y un Timestamp).
0	CREATE2	Utilizada por las versiones más recientes de Tor para crear el circuito.
1	CREATED2	Permite confirmar la creación correcta del circuito por medio de CREATE2 CELLs.

Tabla 1. Funciones de un paquete CELL de tamaño fijo

Estructura de una CELL en handshake versión 2 o 3:

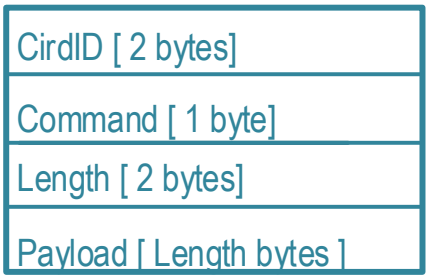

Figura 3. Estructura paquete CELL de tamaño variable

Este tipo de CELL varía de tamaño de acuerdo a la función establecida por el campo “Command” que puede tomar los siguientes valores solo disponibles para CELLs de tamaño variable:

Valor	Comando	Función
129	CERTS	Permite el envío de certificados entre los nodos.
130	AUTH_CHALLENGE	Reto enviado entre nodos para demostrar una identidad de una parte.
131	AUTHENTICATE	Respuesta al reto de autenticación.

Tabla 2. Funciones de un paquete CELL de tamaño variable

La variable CircID contiene el ID del circuito al que está asociada la CELL.

Al momento de enviar una CELL se debe elegir el identificador del circuito ID al que va a pertenecer, este valor es un número entero diferente a 0 elegido por el OP u OR que esté enviando la CREATE CELL y se define de la siguiente manera:

El nodo compara el módulo de las claves PK Identity-key. Si este es el menor valor entre los dos toma el MSB 0 (bit más significativo) de lo contrario toma el MBS 1.

Tener en cuenta que todas las CELL tiene la misma estructura excepto por el payload el cual varía de acuerdo a su función y es utilizado para el intercambio de información lo cual se verá en detalle más adelante.

Existen otros tipos de CELL con diferentes funciones que no se tendrán en cuenta debido a que no son utilizadas en la creación y configuración del circuito que es el objetivo de este documento.

4. CIRCUITO TOR

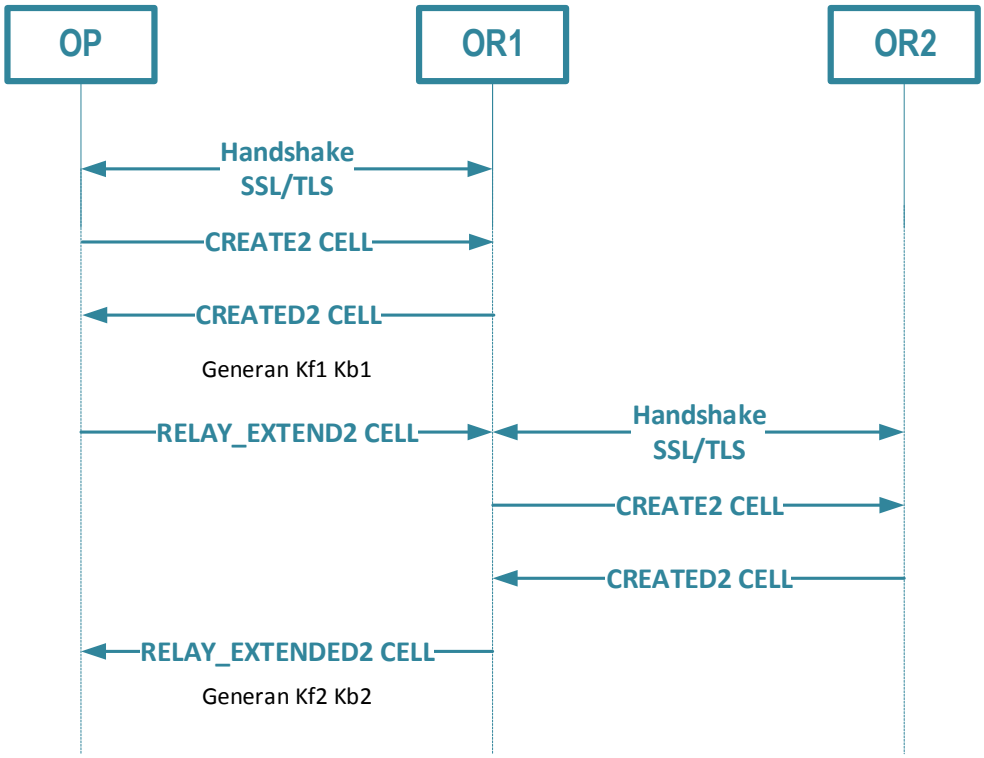

Figura 4. Creación del circuito Tor

El OP es el encargado de crear el circuito eligiendo x nodos por los cuales enviará la información, luego establece conexiones SSL/TLS con cada nodo y posteriormente negocia claves simétricas (capas de cebolla Kfn - Key forward n y Kbn - Key backward n) con cada uno por medio de paquetes CREATE2 y RELAY_EXTEND2 CELL. Este proceso se realiza enviado CELLs dentro de los SSL/TLS records.

A continuación se detalla la creación de un circuito:

- El OP elige N-1 nodos para formar el circuito.
- EL OP establece una conexión SSL/TLS con el primer nodo del circuito.
- El OP envía una CREATE2 CELL con su parte del DH para la obtención del secreto en común del cual se derivan las claves

simétricas Kfn y Kbn. Actualmente se obtienen por medio del protocolo Ntor (operaciones ECDH) y una función HKDF explicadas en detalle más adelante.

- El OP espera como respuesta una CREATED2 CELL por parte del servidor. El payload contiene la parte del Diffie-Hellman del servidor para generar la clave secreta. Luego por medio de la función HKDF se obtienen las claves Kf1 y Kb1 del circuito.

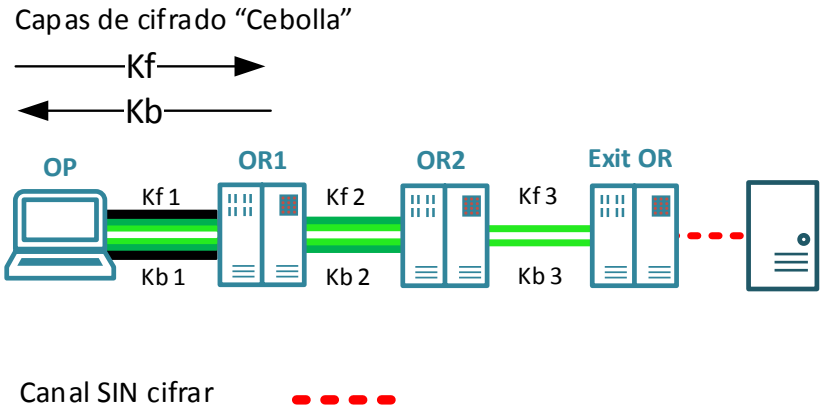

Figura 5. Capas de cifrado en un circuito Tor

- Luego el OP procede a enviar una RELAY_EXTEND2 CELL al OR1 que consiste en una solicitud de extender el circuito, en otras palabras le solicita al OR1 realizar el mismo proceso anterior con el próximo nodo (OR2) que le indica el OP.

- El OR1 luego de establecer una conexión SSL/TLS envía una CREATE2 CELL con una parte del DH del OP al OR2 y este contesta con una CREATED2 CELL con su parte del DH.

- Luego OR1 envía la información del OR2 al OP por medio de una RELAY_EXTENDED2CELL para obtener las claves Kf2 y Kb2. A este punto ha establecido claves cebolla con 2 nodos del circuito.

- Por último el OP continúa este proceso hasta N-1 definiendo todas las claves del recorrido.

4.1 CREATE2 / CREATED2 CELL

Para iniciar la creación del circuito el OP envía la información necesaria para ejecutar y verificar una operación Diffie-Hellman en el payload de una CREATE2 CELL con el fin de definir un valor en común entre ambos nodos empleado más adelante por la función HKDF para obtener las claves Kf y Kb del circuito.

La estructura de una CREATE2 CELL es la siguiente:

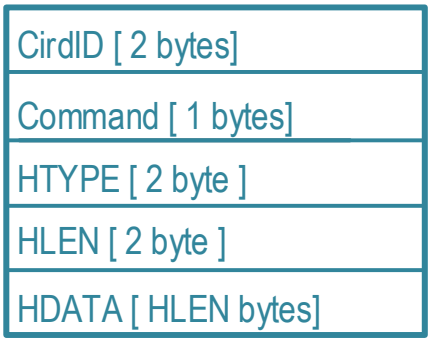

Figura 6. Estructura de un paquete CREATE2 CELL

La variable HDATA contiene los datos para la operación DH del Cliente y HTYPE el método para intercambiar la clave secreta, en este caso Ntor.

La estructura de una CREATED2 CELL es la siguiente:

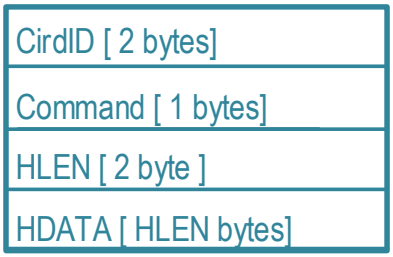

Figura 7. Estructura de un paquete CREATED2 CELL

La variable HDATA contiene los datos para el DH del servidor.

El valor HTYPE hace referencia al tipo de método empleado para este proceso. Tor define 3 tipos para realizar el DH entre nodos:

Método	HTYPE
TAP (Tor Authentication Protocol) primera versión de handshake empleado por Tor.	0x0000
Reserved.	0x0001
Ntor (curve25519 + sha256 handshake) utilizado actualmente por Tor. Implementado en la versión 0.2.4.8 alpha.	0x0002

Tabla 3. Métodos para un handshake entre nodos Tor

4.2 Handshake Ntor

En esta etapa, el protocolo Tor permite establecer un secreto en común entre dos nodos que pertenecen al circuito fijado por el OP por medio de una serie de handshakes empleando DH. Este secreto será utilizado como semilla de la función HKDF y será enviado en la variable “HDATA” de la CREATE2 CELL.

Este protocolo emplea una función de curva elíptica “Curve25519” propuesta por Daniel J. Bernstein. Para mayor detalle del diseño e implementación de la función ver el documento “Curve25519: new Diffie-Hellman speed records”. **[2]**

Se definen las siguientes variables:

H_LENGTH: 32 bytes
ID_LENGTH: 20 bytes
G_LENGTH: 32 bytes

KEYGEN(): función que retorna un conjunto de claves Pública/Privada obtenidas utilizando la función Curve25519.

PROTOID: “ntorcurve25519sha2561”, una cadena de caracteres.

t_mac: PROTOID | ":mac"
t_key: PROTOID | ":key_extract"
t_verify: PROTOID | ":verify"
m_expand: PROTOID | “:key_expand”

Antes de iniciar el envío de CELLs cada nodo ejecuta los siguientes pasos:

- El servidor genera una par de claves por medio de la función curve25519 (b,B), b= SK, B=PK (onion-key) publicada en el router-descriptor del nodo.

- El cliente genera un par de claves por medio de la función curve25519 (x,X) x= SK, X=PK (onion-key) e inicia el handshake enviando su parte del handshake en la variable HDATA de una CREATE2 CELL:

  NodeID: H(Identity-key), hash de la Identity-key del servidor.
  KEYID: PK B
  CLIENT: PK X

- El servidor genera un nuevo conjunto de claves por medio de la función curve25519 (y,Y) y= SK, Y=PK. Luego calcula los valores hash secret_input, verify, auth_input empleados como medio de verificación y el KEY_SEED utilizado para generar las demás claves del circuito:

  secret_input: EXP(X,y) | EXP(X,b) | ID | B | X | Y | PROTOID
  KEY_SEED: H(secret_input, t_key)
  verify: H(secret_input, t_verify)
  auth_input: verify | ID | B | Y | X | PROTOID | “Server”
  AUTH: H(auth_input,t_mac)

- Por último envía una CREATED2 CELL donde el campo HDATA contiene los siguientes valores:

  SERVER: PK Y
  Authe: [AUTH]

- El cliente calcula los valores hash secret_input, verify, auth_input.

  secret_input: EXP(Y,x) | EXP(B,x) | ID | B | X | Y | PROTOID
  KEY_SEED:H(secret_input, t_key)
  verify: H(secret_input, t_verify)
  auth_input: verify | ID | B | Y | X | PROTOID | “Server”

Luego si la función EXP() no produce un punto al infinito y si los hash calculados concuerdan con los enviados por el servidor finaliza el handshake dejando como resultado el intercambio de un valor privado conocido sólo por ambas partes “**KEY_SEED**” el cual será utilizado por la función HKDF para generar las Kfn y Kbn del nodo.

4.3 HKDF– SHA256

Esta función se emplea para derivar las claves Kf y Kb necesarias para cada nodo en el circuito. Su objetivo es tomar como fuente inicial una clave (KEY_SEED) y por medio de una función HMAC derivar una o más claves secretas criptográficamente fuertes. **[3]**

El valor de K se calcula realizando 3 operaciones de hash de la siguiente forma:

$$K = K\_1 \mid K\_2 \mid K\_3$$

K_1 = H(m_expand | INT8(1)**,** KEY_SEED)
K_2 = H(K_1 | m_expand | INT8(2)**,** KEY_SEED)

K_3 = H(K_2 | m_expand | INT8(3)**,** KEY_SEED)

La variable KEY_SEED es el valor resultante del handshake por medio del protocolo Ntor. La variable INT8(n) es un octeto con el valor n. El valor de K es una cadena de 96 bytes seccionada de la siguiente manera:

Figura 8. Segmentación del valor K

Para este caso HASH_LEN = 20 bytes y KEY_LEN = 16 bytes

- **Forward digest (Df)** = los primeros HASH_LEN bytes, utilizados en las RELAY_EXTEND2 CELL para la administración de la CELL.

- **Backward digest (Db)** = los siguientes HASH_LEN bytes, utilizados en las RELAY_EXTEND2 CELL para la administración de la CELL.

- **Forward Key (Kf)** = siguientes KEY_LEN bytes, Clave para el cifrado de información.

- **Backward Key (Kb)** = siguientes KEY_LEN bytes, Clave para el cifrado de información.

- **Nonce** = bytes sobrantes.

4.4 RELAY_EXTEND2/EXTENDED2 CELL

Luego de establecer las claves Kf y Kb con el primer nodo es necesario extender el mismo proceso para definir las claves con los demás nodos del circuito. Para esto el OP envía una RELAY_EXTEND2 CELL al último nodo hasta el momento en el circuito indicando cual es el siguiente

nodo, el modo de conexión, IP, puerto y la información necesaria para calcular un valor KEY_SEED con el segundo nodo.

La estructura de una RELAY_EXTEND2 CELL es la siguiente:

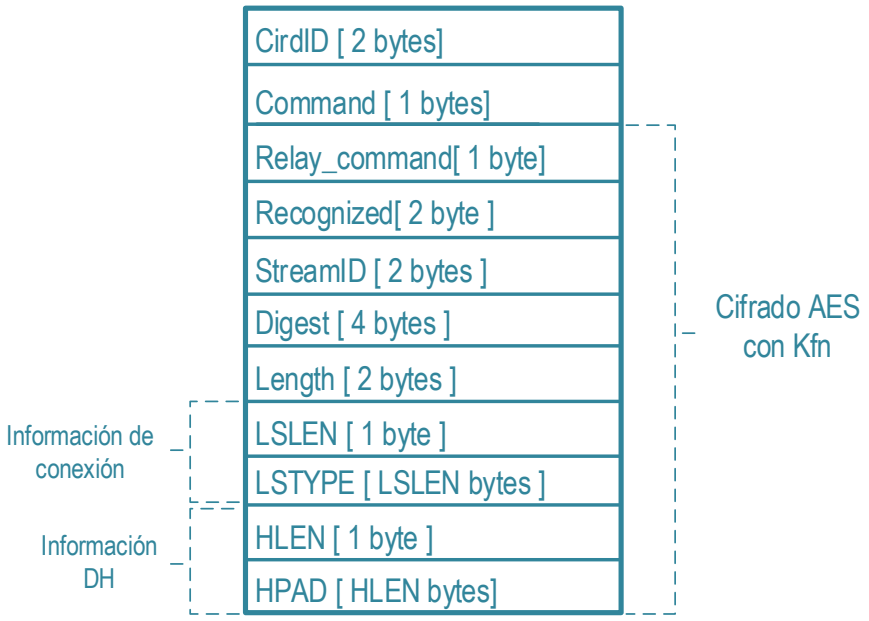

Figura 9. Estructura de un paquete RELAY_EXTEND2 CELL

El valor Relay_command de la RELAY CELL es la función que cumple, para este caso “extend”. El campo Recognized tiene el valor 0. El campo StreamID es un número arbitrario elegido por el OP.

El campo Digest son los primeros 4 bytes del valor Df o Db definido para esta parte del circuito. De acuerdo al sentido en el que se esté enviando la CELL se verifica el valor Forward o backward. Estos son empleados para determinar si el nodo debe descifrar la CELL o continuar su transmisión a través del circuito. De este modo si el campo recognized es 0 y el valor Digest concuerda con el generado para esta parte del circuito procede a descifrar el payload y ejecutar la función del valor command. De lo contrario continúa el envío de esta CELL luego de quitar su capa de cifrado.

El campo LSTYPE contiene el método y la información necesaria para realizar la conexión con el siguiente nodo del circuito. Los métodos son los siguientes:

LSTYPE	Detalle	Valor
SSL/TLS sobre TCP, IPV4	Dirección IPv4 (4 bytes) + Puerto del OR (2 bytes)	0
SSL/TLS sobre TCP, IPV6	Dirección IPv6 (16 bytes) + Puerto del OR (2bytes)	1
Legacy Identity	Hash de la PKCS#1 Identity-key del OR 20 bytes	2

Tabla 4. Métodos de conexión entre nodos

La estructura de una RELAY_EXTENDED2 CELL es la siguiente:

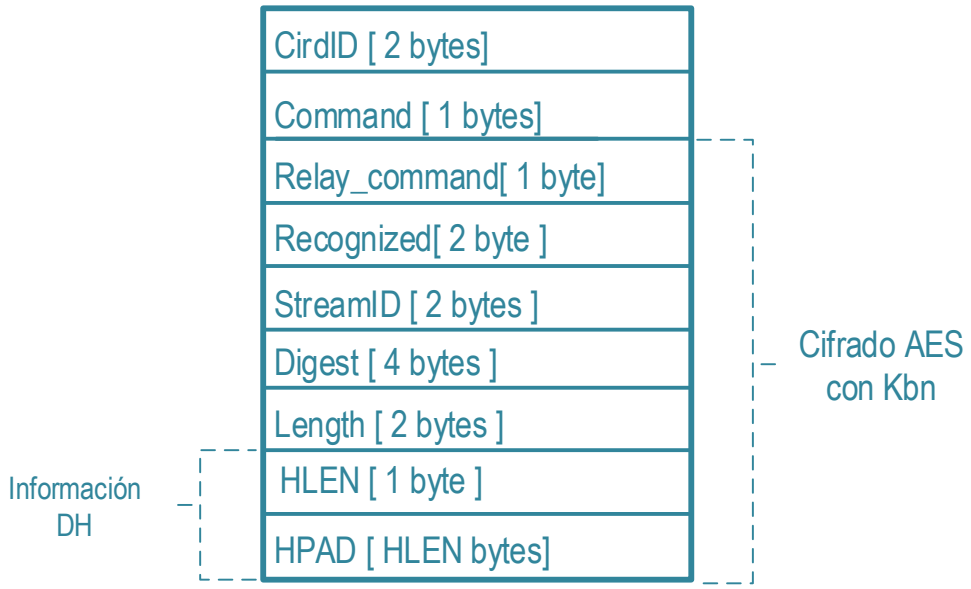

Figura 10. Estructura de un paquete RELAY_EXTENDED2 CELL

5. HANDSHAKE SSL/TLS ENTRE NODOS

El SSL/TLS handshake protocol permite que un OP inicie la creación de un circuito autenticando el primer nodo y posteriormente definiendo una clave simétrica sólo conocida entre ambas partes, agregando así la primera capa de cifrado sobre el tráfico que viajara en el circuito. Tor utiliza una variación del protocolo SSL/TLS sobre servicios comunes como HTTPS, haciendo foco en los mensajes client_hello donde se envían diferentes listas de cipher_suites y los certificados valiéndose de estos para definir la versión de handshake a realizar entre ambas partes.

El SSL/TLS record protocol permite enviar información para la configuración y administración de los circuitos por medio de paquetes tipo CELL entre nodos los cuales van cifrados con las claves establecidas en el handshake protocol.

Actualmente emplea la versión 3 llamada “In protocol” para la creación y autenticación del circuito entre nodos, sin embargo todas las versiones de Tor deben tener compatibilidad con la versión anterior “Renegotiation”. Ahora en adelante se define:

Cliente: Nodo que envía una solicitud de conexión.

Servidor: Nodo que responde la solicitud de conexión.

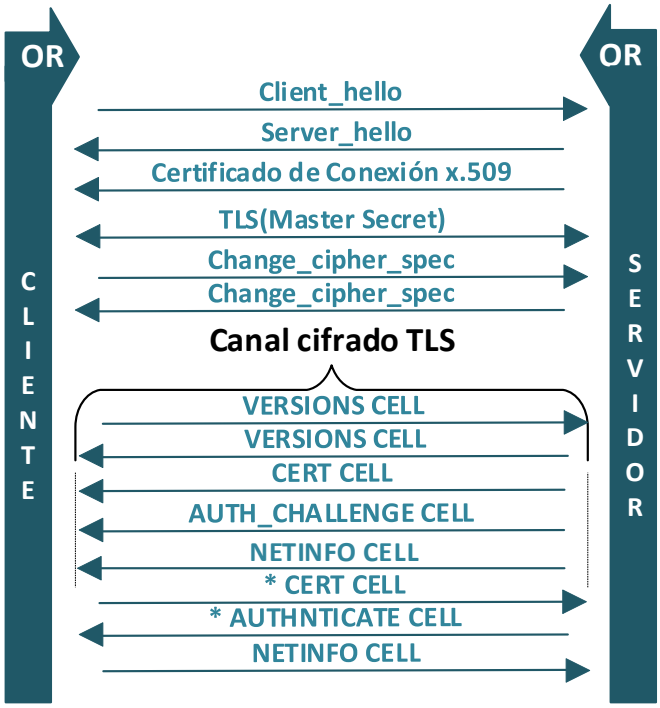

Figura 11. Handshake Versión 3 – In protocol

- El cliente inicia la primera etapa de la conexión enviando un mensaje client_hello. El servidor revisa la lista de cipher_suite enviada por el cliente, en base a esta determina si establece la conexión utilizando las versiones de handshake 1, 2 o 3.

- El cliente no envía certificados y finalizan el primer handshake de acuerdo al protocolo SSL/TLS.

- En esta etapa, cliente y servidor intercambian VERSIONS CELLs para determinar la versión del protocolo.

- El servidor envía una CERT CELL, AUTH_CHALLENGE CELL y una NETINFO CELL.

- Luego el Cliente debe contestar enviando una CERT CELL, AUTHENTICATE y NETINFO CELL. La función y contenido de estas CELLs se detalla más adelante.

Por último al revisar que los datos enviados en las CELL son los esperados finaliza dejando configurada una conexión SSL/TLS entre ambos nodos.

A continuación se describen los tipos de handshakes implementados desde la creación del proyecto Tor hasta la actualidad.

5.1 Versión 1 - Certificates up-front

Es la primera versión implementada por Tor para establecer una conexión SSL/TLS y posteriormente configurar y abrir un circuito sobre Tor. Este tipo de handshake es soportado por todas las versiones de Tor.

Figura 12. Handshake version 1 – Certificates up front

- El cliente inicia la conexión siguiendo el protocolo SSL/TLS enviando un mensaje tipo client_hello advirtiendo las cipher_suite soportadas para el intercambio de las claves. Esta lista solo debe contener las siguientes cipher_suites, de esta forma establece que desea realizar este tipo de handshake:

  Lista de cipher_suites enviada en la Versión 1 por el cliente:

- TLS_DHE_RSA_WITH_AES_256_CBC_SHA
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA
- SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA

- El servidor envía un mensaje server_hello eligiendo la primer cipher_suite de esa lista que sea soportada por el servidor.

- El servidor envía una cadena de 2 certificados, el primer certificado x.509 que contiene su Connection-key denominado “certificado de conexión” y firmado con la Identity-key del nodo, el segundo certificado x.509 que contiene su Identity-key denominado “certificado de identidad” y es autofirmado. Luego en base a las cipher_suites enviadas por el cliente solicita 2 certificados.

- El cliente envía el certificado de conexión y el certificado de identidad del nodo al servidor.

- Por medio de la información intercambiada, se genera el Master-secret.

- Se intercambian los mensajes Change_cipher_spec anunciando que ahora en adelante se enviaran mensaje utilizando la clave negociada.

Para verificar la autenticidad del certificado de identidad que se envía en el handshake, el nodo accede a una lista de certificados publicada por cada servidor de directorio oficial en la red Tor. Este servidor posee una PK denominada “directory-signing-key” utilizada para firmar la lista de certificados autofirmados por cada OR oficial con su identity-key, este contiene las claves a utilizar en el handshake SSL/TLS.

Una revisión a fondo dio como resultado los siguientes problemas en esta versión de handshake: Al iniciar una conexión por medio del protocolo SSL/TLS los certificados son enviados en texto plano y pueden ser vistos por

una persona ajena a la conexión. En una arquitectura cliente-servidor donde se accede a servicios HTTPS, normalmente estos certificados están firmados por un AC[2] raíz con el fin de verificar su validez y en este proceso solo el servidor presenta un certificado a diferencia del handshake visto anteriormente. Por esta cuestión en 2007 se generaron 2 propuestas **[4] [5]** que finalmente fueron aceptadas para implementar un handshake donde el cliente no tenga que enviar certificados el cual se describe a continuación.

Todas las propuestas mencionadas en este documento tienen el estado de "closed" lo cual significa que han sido aceptadas, implementadas e integradas a la documentación oficial sobre el diseño de Tor **[6]**.

5.2 Versión 2 –Renegotiation

En las siguientes versiones además del handshake SSL/TLS los nodos deben intercambiar un conjunto de paquetes denominados CELL para establecer la identidad del cliente frente al servidor y finalizar la creación del circuito debido a que el cliente no presenta un certificado. Más adelante se detalla la estructura y función de cada CELL intercambiada en esta versión.

A medida que utilizamos paquetes CELL en el handshake se explicará la estructura y función de cada una.

[2] Autoridad de Certificación. entidad en la cual se confía, encargada de emitir y revocar certificados digitales.

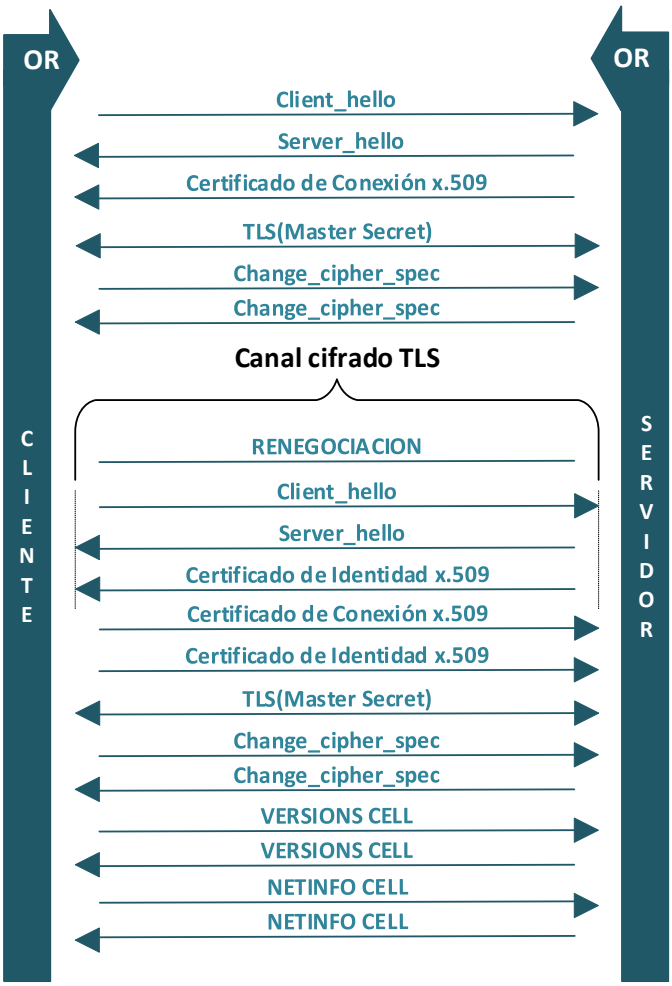

Figura 13. Handshake versión 2 - Renegotiation

- El cliente inicia la conexión siguiendo el protocolo SSL/TLS enviando un mensaje tipo client_hello advirtiendo las cipher_suites soportadas para el intercambio de las claves. Este debe incluir al menos una cipher_suite que no esté en la lista enviada en la versión 1, de esta forma establece que puede realizar este tipo de handshake. Tor recomienda la siguiente lista de cipher_suites con el fin de simular las enviadas comúnmente por un navegador. Esta lista se encuentra en los anexos.

- El servidor envía un mensaje server_hello que contiene las cipher_suites soportadas. Si la cipher_suite enviada en el cliente_hello es igual a la propuesta para a versión 1, este continúa con el comportamiento de la versión 1 enviando 2 certificados y solicitando estos mismos al cliente. De lo contrario el servidor

responde con un server_hello eligiendo la primer cipher_suite de esta lista.

- El servidor envía un único certificado de conexión x509.

- Por medio de la información intercambiada, generan el Master-secret.

- Por último los OR intercambian los mensajes Change_cipher_spec anunciando que ahora en adelante se enviaran mensajes cifrados utilizando la clave negociada.

Hasta aquí se ha establecido una conexión SSL/TLS, pero el cliente no se ha autenticado, por lo cual solicita una renegociación de la sesión.

El protocolo TLS por medio de la extensión “Renegotiation” **[7]**, permite a ambas partes solicitar una nueva negociación, intercambiando nuevos mensajes para configurar otro canal cifrado. Esta renegociación se realiza con el fin de enviar los certificados para la autenticación de ambas partes (como en la versión 1) por medio de un canal cifrado previamente establecido con el primer handshake, mejorando la similitud al comportamiento normal de una conexión SSL/TLS.

Luego de establecido el primer canal cifrado, se intercambian los mensajes client_hello y server_hello pero a diferencia del primer handshake, el servidor envía al cliente solo el certificado de identidad ya que el certificado de conexión fue empleado en el primer handshake y solicita los certificados de identidad y conexión al cliente en este orden. Luego continúa con el comportamiento establecido por el protocolo generando un nuevo canal de cifrado y finalizando el proceso con los mensajes change_cipher_spec.

En este punto tanto el servidor como el cliente verifican que las claves son las informadas en los router-descriptor de cada OR. De esta manera finaliza el handshake. A este punto el servidor y el cliente están

correctamente autenticados. Por último intercambian VERSIONS y NETINFO CELLs para configurar el circuito y establecer la versión del protocolo utilizada. Si se envían CELLs en diferente orden o si las claves no concuerdan con las dadas por el servidor de directorio el servidor termina la conexión.

Debido a la vulnerabilidad descubierta en el 2009 conocida como ataque de renegociación TLS por MarshRay y Steve **[8]**, Openssl deshabilito la renegociación por defecto en sus librerías y el soporte de esta. Esta vulnerabilidad sumado a otros factores planteados en la propuesta 169 **[9]**, motivaron el diseño de una nueva versión de handshake utilizando paquetes tipo CELL como medio para finalizar la etapa de autenticación.

5.3 Versión 3 – In Protocol

A continuación se detalla en profundidad el proceso para establecer una conexión SSL/TLS entre OR comentada al principio de la sección 4. El handshake lo dividiremos en 2 etapas.

Primera etapa: Conexión SSL/TLS.

- El cliente inicia la primera etapa de la conexión enviando un mensaje client_hello.

- El servidor revisa la lista de cipher_suites enviada, si esta lista corresponde a la mencionada en el versión 1 envía una cadena de 2 certificados y solicita al cliente la misma cadena como en la versión 1, de lo contrario envía un certificado de la forma versión 3 (se detalla a continuación) para indicar al cliente que puede realizar este tipo de handshake.

- El cliente no envía certificados finalizando el primer handshake de acuerdo al protocolo SSL/TLS y dejando como resultado una clave de sesión.

A continuación se presentan las diferencias entre los certificados empleados en el handshake versión 2 y versión 3:

Certificados Versión 2	Certificados Versión 3
- El certificado no es autofirmado, esto significa que se utilizó para la conexión el certificado de conexión.	-El certificado es autofirmado, significa que envía el certificado de identidad.
- Solo está definido el campo CN tanto en "Emisor" como en el "Sujeto" en el certificado enviado por el servidor y el sujeto debe terminar en sufijo ".net". Ej. xyz.net.	-El certificado contiene más valores en los campos de "Emisor" y "Sujeto". Además el CN del emisor y sujeto debe terminar en sufijo ".com". Ej. xyz.com.
- El módulo de la llave pública tiene como máximo 1024 bits.	- El módulo de la llave pública del certificado es mayor a 1024 bits.

Tabla 5. Diferencias entre certificados v2 y v3

Segunda etapa: Autenticación por medio de paquetes CELL.

Ahora se realiza la segunda etapa de autenticación intercambiando una serie de CELLs entre ambos nodos:

- Se intercambian VERSIONS CELLs para determinar la versión del protocolo que en este caso será 3.

- Luego el servidor envía una CERT CELL y AUTH_CHALLENGE CELL con el fin de autenticarse frente al cliente y una NETINFO CELL.

- El cliente responde con una CERT CELL y AUTHENTICATE CELL si este desea autenticarse frente al servidor de lo contrario solo enviara una NETINFO CELL.

Una vez establecida la conexión SSL/TLS utilizando la versión 3 ambas partes proceden a definir la versión del protocolo Tor, configurar el circuito definiendo direcciones IP y claves compartidas de cada nodo y terminar el proceso de autenticación. Esto se logra por medio del intercambio de paquetes de tamaño variable llamados CELLs enviados en los SSL/TLS record utilizando la clave simétrica obtenida por medio del handshake.

5.3.1 VERSIONS CELL

La primer CELL que se intercambia en los SSL/TLS records es una VERSIONS CELL con el fin de establecer la versión utilizada del handshake. Esta toma el mayor valor enviado por ambas partes en esta CELL. De acuerdo a esta versión se envían ciertos tipos de CELL para crear y administrar el circuito. Para este caso la versión es 3.

El valor de la variable “command” de este tipo de CELL es 7

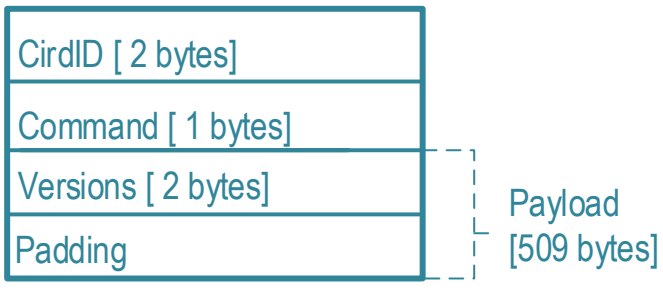

Figura 14. Estructura de un paquete VERSIONS CELL

5.3.2 NETINFO CELL

Esta CELL hace parte de la negociación y configuración de la conexión si el handshake utilizado fue la versión 2 o 3. El payload contiene las direcciones necesarias para construir el circuito más adelante.

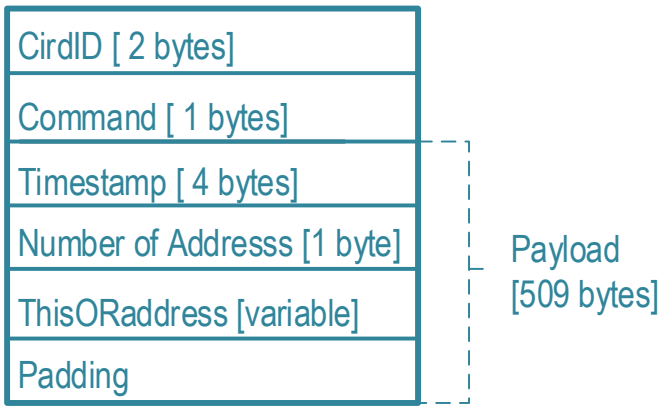

Figura 15. Estructura de un paquete NETINFO CELL

La dirección enviada en el campo ThisORaddress le permite saber a la otra parte de la conexión la IP de ese nodo y así poder verificar que se está conectando al OR indicado ya que esta IP esta listada en su router-descriptor.

La IP está formada de la siguiente manera:

Tipo | Tamaño | valor(IP)

El campo “Tipo” establece si es una dirección IPv4[3] o IPv6[4] tomando los valores 0x04 y 0x06 respectivamente.

El campo [timestamp] es un valor de tipo unix time.

[3] Internet protocol versión 4
[4] Internet protocol versión 6

5.3.3 CERT CELL

Este tipo de CELL contiene las claves necesarias para verificar la identidad de un nodo que ha establecido una conexión utilizando la versión 3 de handshake. El formato del payload es el siguiente:

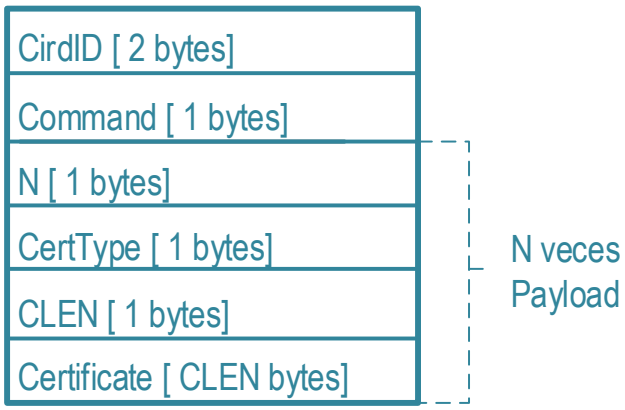

Figura 16. Estructura de un paquete CERT CELL

El campo CertType especifica el tipo de certificado que se envía en la CELL:

Función	CertType	Tipo
Certificado conexión firmado por un -identity-key-RSA de 1024 bits.	1	Link
Certificado de identidad que contiene una identity-key y esta autofirmado.	2	ID
Certificado de enlace AUTHENTICATE RSA 1024bits.	3	AUTH

Tabla 6. Tipos de certificados empleados en la autenticación por CELLs

El campo CLEN define el tamaño en bytes de los N certificados enviados en el payload. Los certificados deben estar formados según el estándar x509.

Los parámetros a verificar en la CERT CELL por parte del cliente son los siguientes:

- El campo N en la CERT CELL debe ser igual a 2, esto significa que esta presentado 2 certificados al cliente.
    - Un certificado tipo 1 con la clave Connection-key.
    - Un certificado tipo 2 con la clave Identity-key.
- Los certificados deben estar vigentes, esto significa que no haya superado la fecha definida en los campos "notBefore" y "notAfter".
- La PK dentro del certificado tipo 1 "conexión" debe ser igual a la PK (Connection-key) presentada en el certificado enviado por el servidor al momento de realizar el handshake SSL/TLS versión 3.
- La PK dentro del certificado tipo 2 ID es una clave RSA 1024 bits
- La PK definida en el certificado tipo 2 ID fue utilizada para firmar ambos certificados, esto significa que esta PK se utilizó para firmar el certificado Link y para autofirmar el certificado ID. Recordemos que esta clave Identity-key puede ser verificada en cualquier momento por un nodo accediendo a la lista publicada por los servidores de directorio oficiales de la red Tor.

Luego de verificar los datos enviados en la CERT CELL por parte del servidor y solo si estos cumplen las premisas anteriores, el cliente envía una NETINFO CELL.

Ahora en caso de que el cliente desee autenticarse frente al servidor, este debe contestar con una AUTHENTICATE CELL con la respuesta al reto enviado por el servidor y una CERT CELL en la que el servidor debe verificar los siguientes parámetros:

- El campo N en la CERT CELL debe ser igual a 2, esto significa que esta presentado 2 certificados al cliente.

- Un certificado tipo 3 AUTH, el cual es un certificado con una clave RSA de 1024 bits denominada -authenticate-key- la cual es empleada sólo en este tipo de handshake.

- Un certificado tipo 2 ID con la clave -Identity-key-

- Los certificados deben estar vigentes, esto significa que no haya superado la fecha definida en los campos "notBefore" y "notAfter".

- La PK definida en el certificado tipo 2 ID fue utilizada para firmar ambos certificados, esto significa que esta PK se utilizó para firmar el certificado AUTH y para autofirmar el certificado ID.

5.3.4 AUTH_CHALLENGE CELL

Este tipo de CELLs son empleadas para intercambiar un reto entre dos nodos en el cual uno desea autenticarse utilizando el handshake versión 3.

Estructura de una AUTH_CHALLENGE CELL:

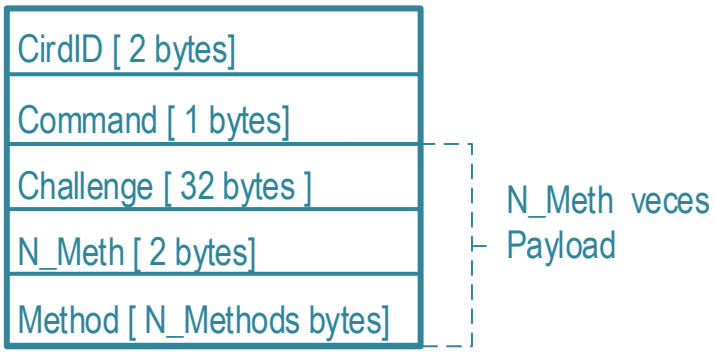

Figura 17. Estructura de un paquete AUTH_CHALLENGE CELL

El campo Challenge es una cadena de caracteres generada por el servidor utilizando un RNG[5] o PRNG[6] que ira inmerso en el hash de la respuesta enviada en la AUTHENTICATE CELL por el cliente.

El campo N_Meth contiene los diferentes métodos que utilizaría para el proceso de autenticación, actualmente solo está definido el método "authenticate" con el valor 1, descrito a continuación.

5.3.5 AUTHENTICATE CELL

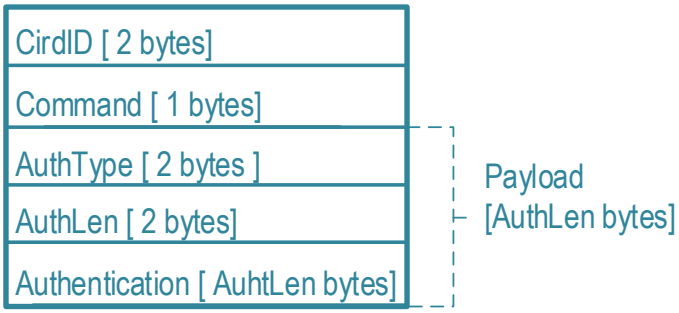

Figura 18. Estructura de un paquete AUTHENTICATE CELL

El cliente debe realizar las siguientes operaciones y enviarlas en el payload de una AUTHENTICATE CELL para finalizar el proceso de autenticación frente al servidor.

El campo AuthType contiene el valor 1, actualmente el único método para realizar este proceso. El campo AuthLen define el tamaño de los datos enviados en el campo Authentication como parte del reto el cual soluciona utilizando una clave RSA (authentication-key), realizando operaciones de hashing con la función SHA256 y haciendo uso del "TLS secret" intercambiado en la etapa del handshake SSL/TLS.

[5] Random Number Generator
[6] PseudoRandom Number Generator

A continuación se detalla la operación:

El cliente inicia definiendo los siguientes valores:

TYPE: [1 byte] contiene la cadena de caracteres “AUTH0001”

CID: [32 bytes] el valor hash de la PK Identity-key del cliente.

SID: [32 bytes] el valor hash de la PK Identity-key del servidor.

SLOG: [32 bytes] el valor hash de todas las CELLs enviadas por el servidor al cliente hasta aquí, esto significa el hash de la concatenación de los siguientes bytes:

H (NETINFO | VERSIONS | AUTH_CHALLENGE | CERT)

CLOG: [32 bytes] el valor hash de todas las CELLs enviadas por el cliente al servidor hasta aquí, esto significa el hash de la concatenación de los siguientes bytes:

H (NETINFO | VERSIONS | CERT)

SCERT: [32 bytes] el valor hash del certificado empleado por el servidor para establecer la conexión SSL/TLS, este es el certificado de identidad.

TLSSECRETS: [32 bytes] el valor obtenido al calcular un HMAC utilizando el “TLS secret” como clave K de la cadena M compuesta de la siguiente forma:

M = client_random | server_random | proto

proto: es la cadena de caracteres “Tor V3 handshake TLS cross-certification”.

Los valores client_random y server_random fueron definidos previamente en la etapa del handshake SSL/TLS.

RAND: [24 bytes] un valor aleatorio generado por el cliente.

SIG: [32 bytes] el valor resultante de firmar del valor hash SHA256 de todos los valores anteriormente definidos utilizando la -authenticate-key-.

Estos valores son enviados en el campo [Authentication] de la AUTHENTICATE CELL.

El servidor luego de recibir la AUTHENTICATE CELL verifica que los datos enviados sean los esperados. Por último calcula y verifica el valor del campo SIG. De esta forma finaliza el proceso y el servidor ha autenticado correctamente al cliente, en caso contrario el servidor termina la conexión.

6. BRIDGES RELAYS

La censura de contenidos, el control sobre consultas y difusión de opiniones entre otros son prácticas que se llevan a cabo en algunos países por parte del gobierno acabando con la idea misma de internet como herramienta para la libre distribución y acceso a la información. Fundamentándose en promover la estabilidad cultural o política del país se implementan sistemas de censura y vigilancia masiva, un ejemplo es el "Great firewall of China" puesto en marcha por el gobierno de china en el año 2003 y que continúa vigente al día de hoy. Estos gobiernos que controlan los ISP del país mantienen la tarea de bloquear el acceso a cierto contenido de internet filtrado URLs, listas de IP, filtrando resoluciones de nombres de dominio o analizando los paquetes en busca de palabras clave o protocolos que vayan en contra de las políticas de uso de internet en ese país.

Anteriormente se expuso el funcionamiento de Tor, como agrega anonimato a nuestras comunicaciones y permite acceder a una gran diversidad de contenidos en internet, dificultado la labor de controlar la navegación de estos usuarios por lo cual es una de las principales herramientas objeto de censura en países que ejercen esta actividad. Ahora recordando su funcionamiento es necesario llevar a cabo un primer paso al contactar un servidor de directorio para obtener la lista de relays públicos y dar comienzo a la creación del circuito por el cual enviaremos todo nuestro tráfico. Debido a esto es imperativo que la primera conexión no sea bloqueada por el ISP y aquí entran en acción los “Bridge relays”. Estos son nodos similares a los Tor-relay con la función de servir como puente hacia la red Tor, con la única diferencia que no publican sus “server-descriptor” por lo cual no están listados en los servicios de directorio y se configuran de forma manual en el navegador Tor.

6.1 Métodos de ofuscación para el tráfico Tor

"Deep Packet Inspection" o DPI, es uno de los métodos empleados por los gobiernos para inspeccionar, filtrar y determinar la acción a tomar sobre una conexión de un usuario. Esta técnica consiste en inspeccionar el contenido de los paquetes que se envían a internet y que deben pasar por el ISP clasificándolos por protocolo. Debido a esto y como vimos anteriormente Tor hace uso de SSL/TLS para sus conexiones, el proyecto Tor desarrolló una técnica para ofuscar este tráfico (haciendo más difícil la tarea de DPI) denominada "pluggable transports". Esta técnica consiste en definir un protocolo para modificar el tráfico entre el OP y el bridge con el cual intenta establecer comunicación.

6.1.1 Obfsproxy

Consiste en un proxy desarrollado por George Kadianakis, integrado por defecto en el paquete de instalación Tor que transforma el tráfico inicial desde el OP hacia el bridge redirigiéndolo primero hacia un servidor Obfsproxy del proyecto Tor, utilizando un protocolo de ofuscación.

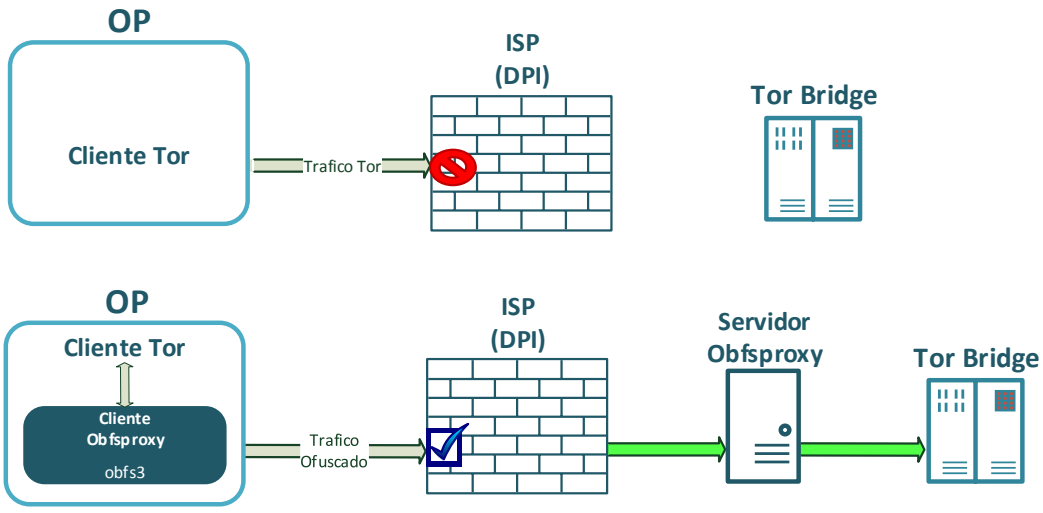

Figura 19. Escenario conexión Obfsproxy

Obfsproxy soporta una variedad de “pluggable transports protocols”[7], Algunos de los recomendados por el proyecto Tor son ScrambleSuite, FTE y Obfs3 del cual detallaremos su funcionamiento a continuación.

6.1.2 Obfs3

Denominado Obfs3 o “The Threebfuscator” es un protocolo con una única función de transformar el tráfico desde el cliente-obfs al servidor-obfs, enviado los datos cifrados con AES-CTR acompañado de relleno aleatorio para dificultar su detección. La parte de autenticación se continúa desarrollando por medio de TLS ofuscado en los mensajes cifrados. El protocolo lo explicaremos en 4 pasos. Se define:

Cliente: parte que inicia la conexión
Servidor: parte que acepta la conexión
MAX_PADDING: 8194bits
KEYLEN: 16 bytes
COUNTERLEN: 16 bytes
PADLEN: Número (relleno) entre los valores [0, MAX_PADDING/2]

Paso 1. Definición de claves

El cliente elige un PADLEN, calcula los valores a utilizar en el DH con g = 2, un primo p y un número al azar x elegidos de acuerdo al RFC 3526 “More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)” **[17]**. Por último se calcula X= g^x(mod p) así mismo el servidor define un PADLEN y calcula el par de claves y,Y.

[7] Protocolos utilizados para ofuscamiento de tráfico de red.

Paso 2. Intercambio de parámetros

Ambas partes intercambian los mensajes:

Cliente: X | WR(PADLEN)
Servidor: Y | WR(PADLEN)

Paso 3. Diffie-Hellman y claves de sesión

Luego intercambiar los valores X y Y, proceden a terminar el DH generando un secreto en común denominado “SHARED_SECRET”. En base a este valor se calcula la clave de sesión a utilizar por el algoritmo AES-CTR de la siguiente forma:

INIT_SECRET = HMAC(SHARED_SECRET, "Initiator obfuscated data")

RESP_SECRET = HMAC(SHARED_SECRET, "Responder obfuscated data")

INIT_KEY = INIT_SECRET[:KEYLEN]

INIT_COUNTER = INIT_SECRET[KEYLEN:]

RESP_KEY = RESP_SECRET[:KEYLEN]

RESP_COUNTER = RESP_SECRET[KEYLEN:]

INIT_ hace referencia al iniciador de la conexión (cliente) y RESP_ al que acepta la conexión (servidor).

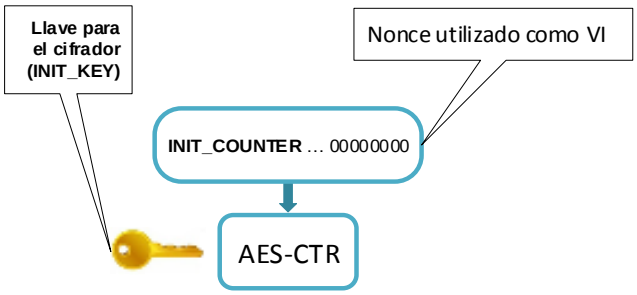

Figura 20. Clave y nonce, algoritmo AES-CTR

Paso 4. Envío de mensajes ofuscados

La primera vez que se intercambia tráfico entre ambas partes se calcula un nuevo PADLEN2 y se forma el mensaje de la siguiente manera:

MAGIC_ FLAG_INIT= HMAC(SHARED_SECRET, "Initiator magic")

MAGIC_ FLAG_RESP= HMAC(SHARED_SECRET, "Responder magic")

Cliente – Servidor:

Mensaje: WR(PADLEN2) | MAGIC FLAG_INIT | E(INIT_KEY, DATA)

Servidor – Cliente:

Mensaje: WR(PADLEN2) | MAGIC FLAG_RESP | E(RESP_KEY, DATA)

DATA son los mensajes TLS cifrados con AES_CTR. Por último el valor “MAGIC_FLAG_INIT” Y “MAGIC_FLAG_RESP” le permite al receptor establecer en que parte termina el relleno y comienzan los datos enviados, así al terminar esta cadena de 32 bytes debe comenzar a descifrar los siguientes bytes.

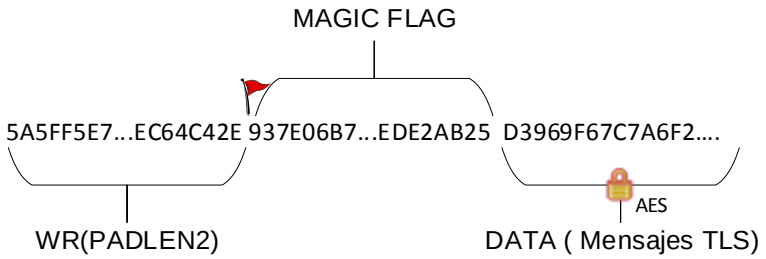

Figura 21. Mensaje Obfs3

Luego del primer mensaje y ya teniendo identificado el “Magic flag” se continúa enviado tráfico cifrado de la siguiente forma:

Cliente – Servidor: E(INIT_KEY, DATA)

Servidor – Cliente: E(RESP_KEY, DATA)

6.2 Obtener y configurar Bridges

El software Tor es distribuido con una lista de bridges configurados por defecto por lo cual antes de iniciar el proceso de configuración es recomendable intentar acceder ya que si bien estos son públicos es posible que el ISP no los tenga bloqueados aun. De lo contrario se deben solicitar por medio de BridgeDB.

BridgeDB está conformado por varios servidores del proyecto Tor que tienen la función recolectar y listar los estados de cada bridge en la red por medio de los netwrok-status y bridge-descriptor. Estos documentos le permiten aprender información sobre cada bridge como el propósito, nickname, IP, puerto, fingerprint, fecha de publicación entre otros campos necesarios para validar y así mantenerlos en la lista de bridges a distribuir.

Una línea de configuración de un bridge está formada de la siguiente manera

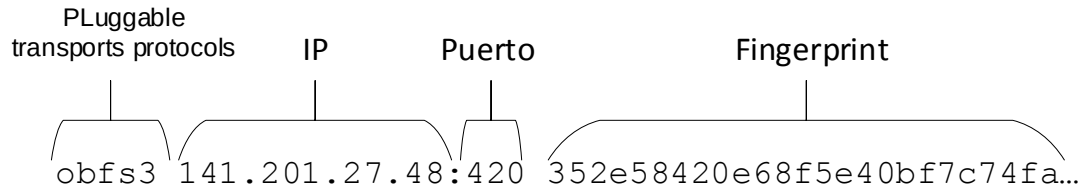

Figura 22. Línea de configuración para un Bridge

Pluggable transports protocol (Opcional): Puede tomar varios valores como Obfs2, Obsf3, ScrambleSuite, FTE.

IP: dirección IP del nodo Bridge

Port: puerto del bridge

Fingerprint (Opcional): Huella que identifica al Bridge

Existen 2 formas de obtener bridges. La primera es visitando la pagina web “https://bridges.torproject.org/” donde podemos solicitar un bridge para configurar de forma manual en el navegador Tor, además da la opción de obtener una línea normal de configuración (1) o una avanzada utilizando un pluggable-transports protocol de los mencionados anteriormente(2):

Figura 23. Pantalla BrigeDB

De acuerdo al modo seleccionado carga la siguiente pantalla con los bridges.

Figura 24. Bridges distribuidos por BridgeDB

La segunda forma es enviando un email al servidor BridgeDB a la dirección bridges@bridges.torproject.org y en el cuerpo del email “get transport obfs3”, este último se puede cambiar por el protocolo de ofuscación

que desee utilizar. El proyecto Tor establece que obligatoriamente se debe enviar el correo desde una cuenta en Gmail, Riseup o Yahoo.

Para ingresar los Bridge obtenidos del BridgeDB, seleccionamos la opción “configure” al iniciar el navegador:

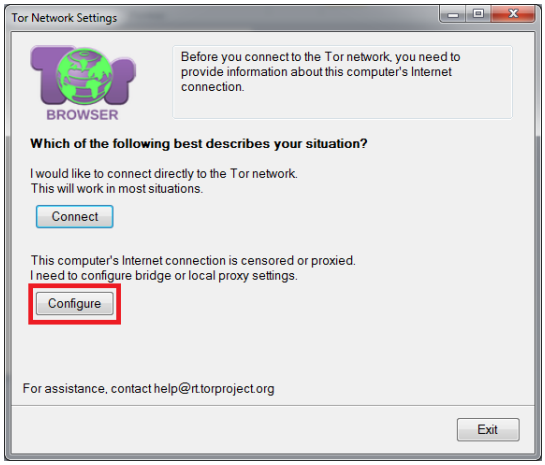

Figura 25. Primera pantalla de configuración de bridge en el navegador Tor

Luego seleccionamos la opción “Does your Internet Service Provider (ISP) block or otherwise censor connections to the Tor Network” para informar al navegador que nuestro ISP o algún otro sistema bloqueas las conexiones hacia Tor:

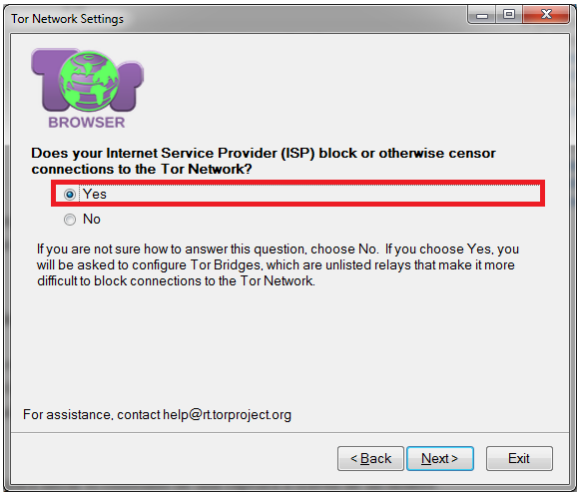

Figura 26. Segunda pantalla de configuración de bridge en el navegador Tor

Seguida de esta pantalla nos muestra 2 opciones:

La primera “connect with provided bridges” establece la conexión utilizando los bridges que vienen configurados por defecto y nos permite elegir el protocolo de ofuscación:

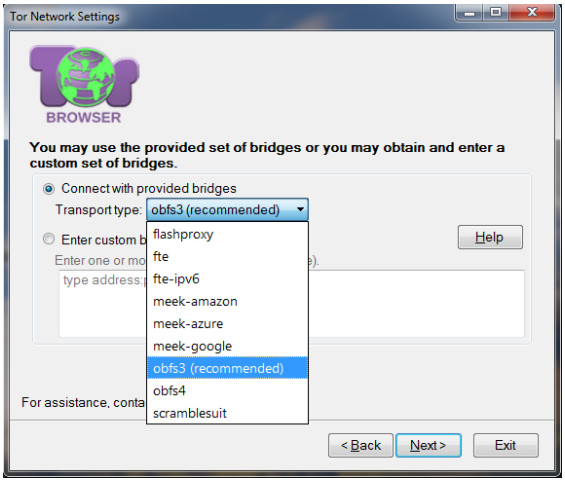

Figura 27. Tercera pantalla de configuración de bridge en el navegador Tor

La segunda opción permite ingresar bridge de forma manual:

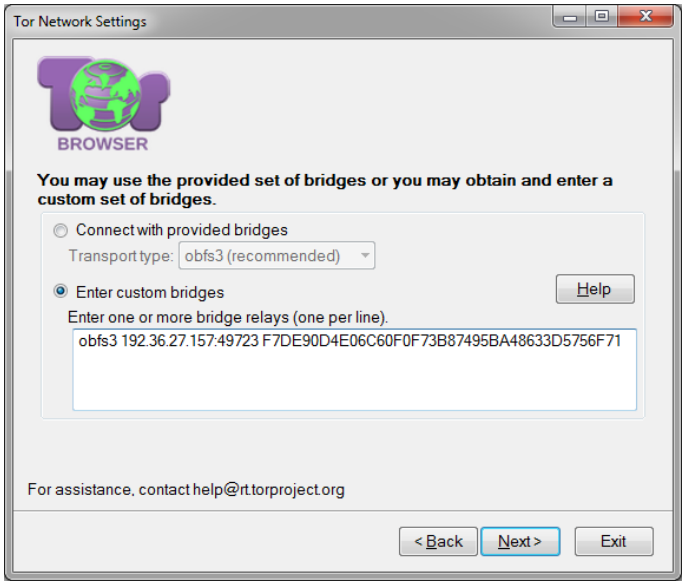

Figura 28. Cuarta pantalla de configuración de bridge en el navegador Tor

Seguido nos pregunta si nuestra conexión hace uso de un proxy para acceder a internet. Si es así procedemos a configurarlo de lo contrario seccionamos la opción “No” y así finalizamos la configuración del bridge dado inicio al proceso de conexión hacia la red Tor.

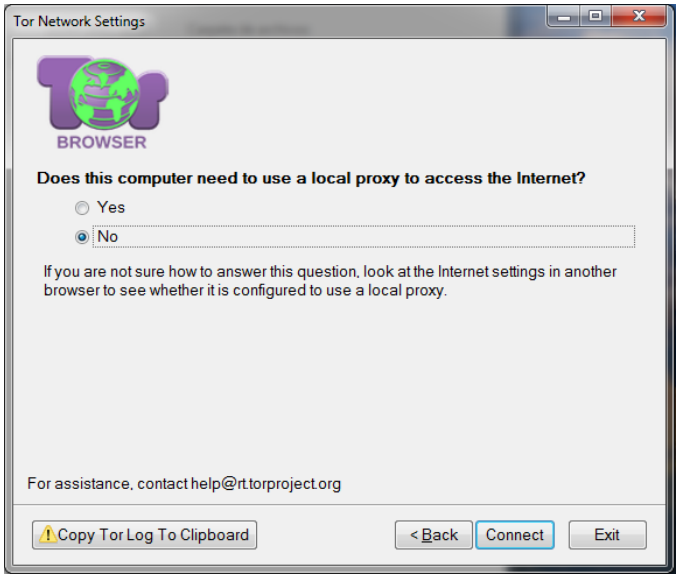

Figura 29. Quinta pantalla de configuración de bridge en el navegador Tor

7. ANÁLISIS PRÁCTICO DEL TRÁFICO DE RED TOR

Anteriormente mencionamos DPI como una de las técnicas de censura que se basa en patrones establecidos por cada protocolo. Para observar el comportamiento del OP al iniciar una conexión y con el fin de determinar las características que hacen único este tráfico frente a otros servicios SSL/TLS se desarrolló un laboratorio utilizando los navegadores Tor, Google chrome, Mozilla Firefox, Opera e Internet Explorer para generar dicho tráfico. Para la captura y análisis del tráfico de red hacemos uso de la herramienta Wireshark[8]. Por último se empleará un bridge utilizando Obfs3 como protocolo de ofuscación para establecer la conexión y comparar el tráfico de ambas conexiones.

[8] Software para captura y análisis de trafico de red

Como primer paso se realizaron consultas HTTPS a diferentes servidores que implementan SSL/TLS con el fin de identificar las variantes con respecto a un OR.

A continuación se detallan los hallazgos:

Al iniciar el navegador Tor vemos como se intercambian los mensajes propios de un Handshake TLS para establecer la primera conexión entre el OP y un OR identificado con la IP 154.35.175.225.

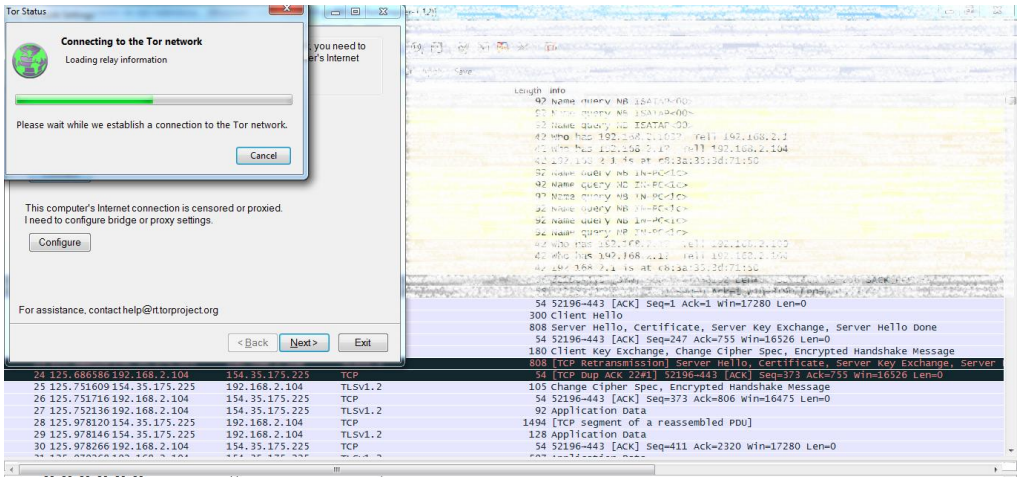

Figura 30. Tráfico de red generado al iniciar una conexión Tor

Luego al realizar una consulta con el navegador Tor se inicia el intercambio de tráfico cifrado empleando el protocolo TLS:

No.	Time	Source	Destination	Protocol	Length	Info
19	125.248156	192.168.2.104	154.35.175.225	TLSv1.2	300	Client Hello
20	125.513082	154.35.175.225	192.168.2.104	TLSv1.2	808	Server Hello, Certificate, Server Key Exchange, Server Hello Done
22	125.522683	192.168.2.104	154.35.175.225	TLSv1.2	180	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
23	125.686526	154.35.175.225	192.168.2.104	TLSv1.2	808	[TCP Retransmission] Server Hello, Certificate, Server Key Exchange, Server
25	125.751609	154.35.175.225	192.168.2.104	TLSv1.2	105	Change Cipher Spec, Encrypted Handshake Message
27	125.752136	192.168.2.104	154.35.175.225	TLSv1.2	92	Application Data
29	125.978146	154.35.175.225	192.168.2.104	TLSv1.2	128	Application Data
31	125.979268	192.168.2.104	154.35.175.225	TLSv1.2	597	Application Data
33	126.289610	192.168.2.104	154.35.175.225	TLSv1.2	597	Application Data
35	126.521198	154.35.175.225	192.168.2.104	TLSv1.2	597	Application Data
37	126.521925	192.168.2.104	154.35.175.225	TLSv1.2	597	Application Data
38	126.756455	154.35.175.225	192.168.2.104	TLSv1.2	597	Application Data
40	126.756987	192.168.2.104	154.35.175.225	TLSv1.2	597	Application Data
45	127.003554	154.35.175.225	192.168.2.104	TLSv1.2	1255	Application Data
51	127.222379	154.35.175.225	192.168.2.104	TLSv1.2	1494	Application Data
57	127.235042	154.35.175.225	192.168.2.104	TLSv1.2	1494	Application Data
63	127.245431	154.35.175.225	192.168.2.104	TLSv1.2	1494	Application Data
69	127.455831	154.35.175.225	192.168.2.104	TLSv1.2	1494	Application Data
75	127.480048	154.35.175.225	192.168.2.104	TLSv1.2	1494	Application Data
81	127.490604	154.35.175.225	192.168.2.104	TLSv1.2	1494	Application Data
83	127.491031	192.168.2.104	154.35.175.225	TLSv1.2	597	Application Data
86	127.753388	154.35.175.225	192.168.2.104	TLSv1.2	1494	Application Data
90	127.860122	154.35.175.225	192.168.2.104	TCP	1494	[TCP segment of a reassembled PDU]
92	127.860638	154.35.175.225	192.168.2.104	TLSv1.2	1494	Continuation Data
94	127.863855	154.35.175.225	192.168.2.104	TLSv1.2	1494	Continuation Data
96	127.864279	154.35.175.225	192.168.2.104	TLSv1.2	1494	Continuation Data
99	128.081042	154.35.175.225	192.168.2.104	TLSv1.2	1494	Continuation Data
101	128.086872	154.35.175.225	192.168.2.104	TLSv1.2	1494	Continuation Data
103	128.088794	154.35.175.225	192.168.2.104	TLSv1.2	1494	Continuation Data

Figura 31. Tráfico TLS entre el OP y un OR.

Para comprobar que esta IP pertenece a un OR de la red Tor, la verificamos visitando la página https://atlas.torproject.org/ en la cual están listados todos los Tor-relays que están activos actualmente.

Realizamos la consulta de la IP 154.35.175.225:

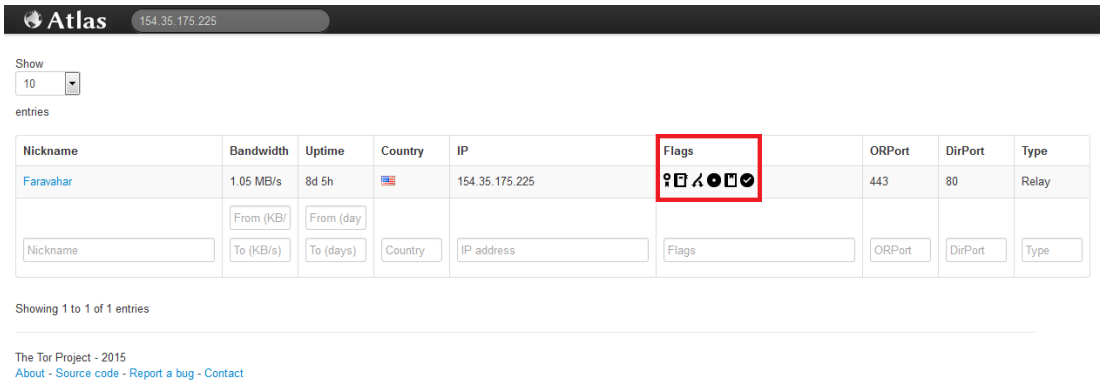

Figura 32. Consulta de IP en atlas.torproject.org

La consulta nos devuelve una pantalla con información del Relay como Nickname, ancho de banda, tiempo activo y un cuadro denominado "flags" que nos indica que esta activo y estable. Además de estos datos es posible obtener más información sobre los relays. Para el propósito de esta práctica con estos datos es suficiente.

El servidor Tor presenta un único certificado, además tiene una validez de 13 días lo cual hace único su comportamiento frente a otros servicios del mercado como se muestra a continuación:

Validez del certificado Tor:

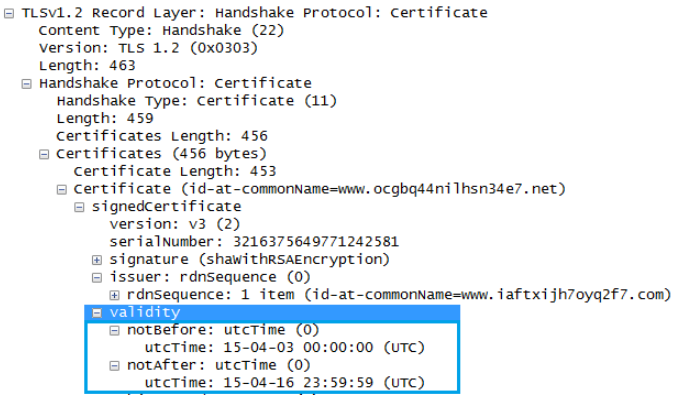

Figura 33. Certificado Tor

Servicio	Versión	Número de certificados	Validez del certificado
Servidor Tor	TLS v.1.2	1	13 días
https://www.torproject.org	TLS v.1.2	2	2 años,7 meses
https://duckduckgo.com	TLS v.1.2	2	11 meses
https://www.verisign.com	TLS v.1.0	2	2 años
https://www.eff.org	TLS v.1.2	2	1 año, 2 meses
https://www.gmail.com	TLS v.1.2	3	3 meses
https://www.ietf.org	TLS v.1.2	4	1 año, 1 mes
https://www.comodo.com	TLS v.1.2	3	2 años
https://www.twitrer.com	TLS v.1.2	2	2 años
https://www.facebook.com	TLS v.1.2	2	1 año, 2 meses

Tabla 7. Detalle certificados SSL/TLS

Con respecto a los certificados logramos verificar que están construidos de acuerdo a la documentación oficial y en este caso empleando la versión 2. De esta forma el servidor presenta un certificado con el campo CN terminado en el sufijo ".net" (www.ocgbq44nilhsn34e7.net) el cual es una de las formas que utiliza para determinar qué tipo de handshake ejecutar. Además el cliente envía en el mensaje client_hello, una cipher_suite de la siguiente forma: (SCSV) "TLS_EMPTY_RENEGOTIATION_INFO_SCSV", la cual no es una suite de cifrado, esta sirve para notificar al servidor que puede realizar una renegociación TLS, simulando el campo "renegotiation_info". A su vez el servidor envía en el server_hello el campo "renegotiation_info", (en este caso 0 ya que es el primer handshake de conexión).Así determinamos que empela un handshake versión 2 para la conexión.

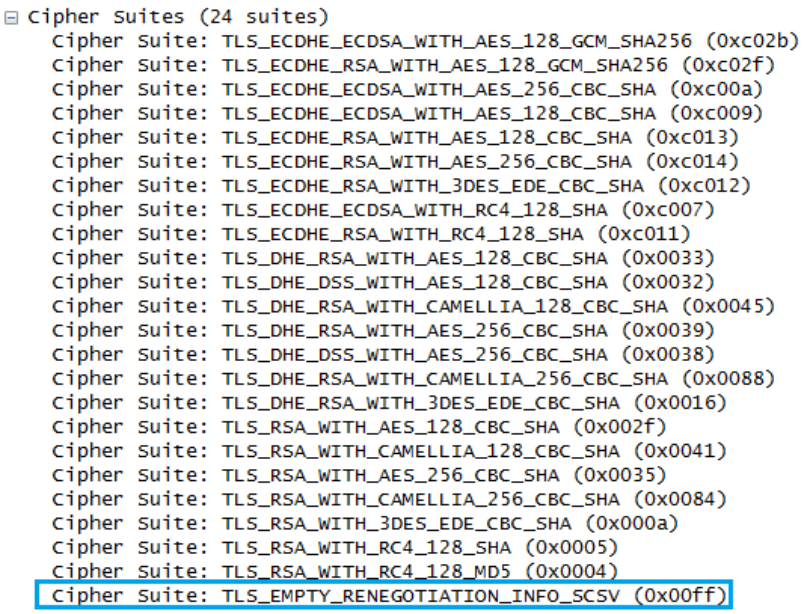

Figura 34. Cipher suite enviada por Tor browser

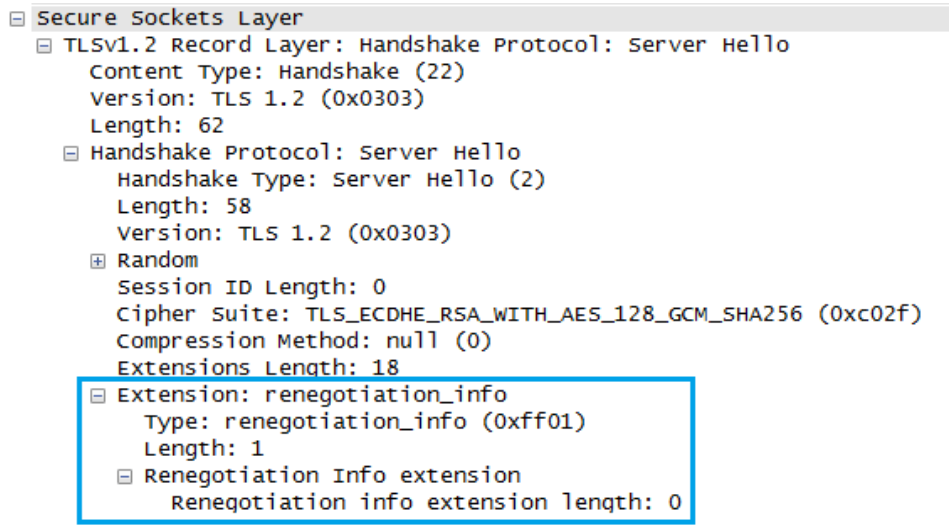

Figura 35. Extensión "Renegotiation" en mensaje Server_hello

Además identificamos que el navegador Tor por medio del client_hello envía una cantidad de cipher_suites diferente a la enviada por algunos de los navegadores más usados del mercado como se muestra a continuación:

Navegador	Versión	Cipher_suites
Tor Browser	4.5.1	24
Internet Explorer	11.09	26
Opera	29.0.1795.5460 0	21
Google Chrome	43.0.2357.65 m	17
Mozilla Firefox	38.0.1	11

Tabla 8. Detalle cipher_suites en mensaje Client_hello

Para finalizar procedemos a capturar el tráfico desde el OP hacia un OR utilizando los Bridge relay. Para esto solicitamos al BridgeDB una lista de bridges y se configuran en el navegador Tor:

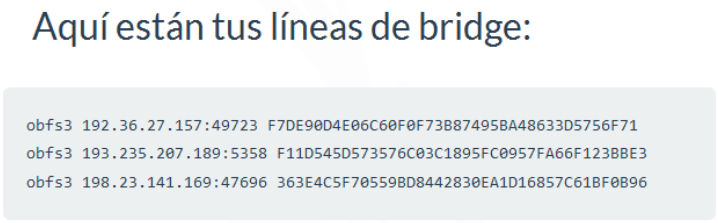

Figura 36. Bridges distribuidos por BridgeDB

Figura 37. Configuración de Bridges en navegador Tor

Agregamos un el filtro “ip proto\tcp” en Wireshark para capturar solo tráfico TCP entrante y saliente así podemos observar con más precisión los mensajes generados por la conexión entre el Bridge y el OP.

Conociendo las direcciones de los bridges, filtramos el tráfico de esas IPs.

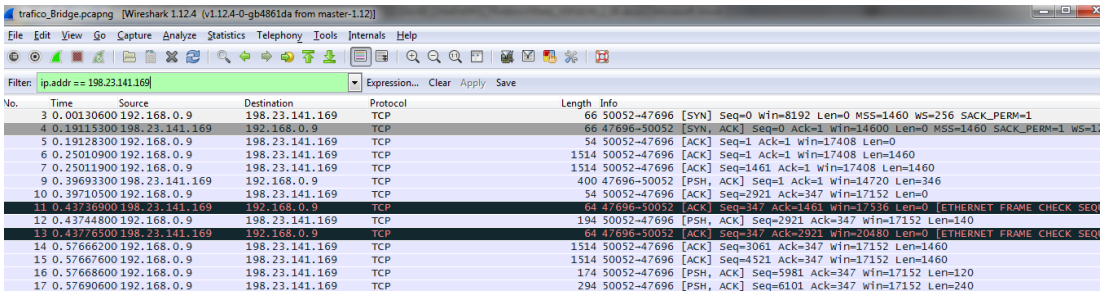

Figura 38.Tráfico de red Tor empleando Bridges

Por último logramos observar que al finalizar la conexión con éxito, no se envió tráfico SSL/TLS, solo se evidencia tráfico TCP con contenido aparentemente aleatorio (tráfico ofuscado que contiene los mensajes TLS cifrados con AES_CTR como vimos en la sección 6.1.2 Obfs3) sin alguna relación al tráfico capturado anteriormente sin el uso de bridges.

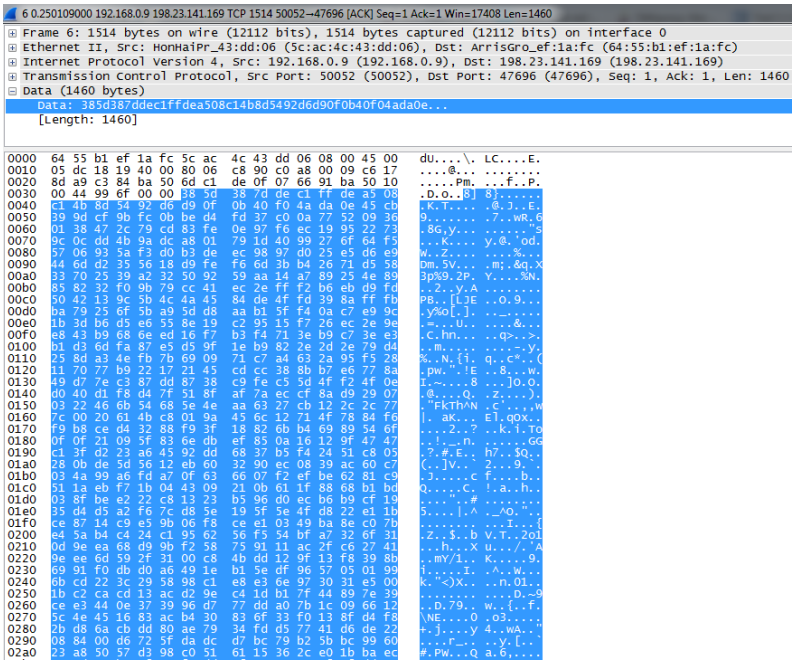

Figura 39. Datos transmitidos empelando Obfs3

Al finalizar el laboratorio observamos lo siguiente:

Tor permite realizar la primera conexión a la red empleando simples ORs o por medio de un bridge como entrada inicial o puente. Ambos tráficos van cifrados solo que los bridges utilizan los protocolos de ofuscación para evitar que esta conexión sea bloqueada por los ISP que conocen con anterioridad las IP de la red Tor y las características de este tráfico de red. Lo comprobamos viendo el primer tráfico (Figura 31) generado por el navegador Tor utilizando TLS 1.2 y luego el tráfico TCP (ofuscado) (Figura 38) configurando un bridge como entrada a la red.

Logramos establecer varios patrones que diferencian el handshake SSL/TLS que realiza el navegador Tor hacia un OR con respecto al realizado por diferentes navegadores y servicios, dando como resultado los siguientes patrones:

- Número de certificados (1).
- Certificado único en cuanto a su tiempo de validez y AC emisora.
- Cipher_suites presentadas por el navegador Tor (24).

Por último comprobamos que la conexión utilizando Bridge NO presentó los patrones mencionados anteriormente, esperados en una conexión normal de Tor utilizando un OR como nodo inicial, cumpliendo el objetivo de minimizar la probabilidad de ser bloqueada la conexión por parte de un ISP:

	Sin Bridge	**Con bridge**
Tráfico de red	TLS1.2	TCP
Certificados	1	No aplica
Cipher_suite	24	No aplica

Tabla 9. Comparación tráfico al usar Bridges

8. CONCLUSIONES

El presente análisis sobre la red Tor buscó dar a conocer en detalle el protocolo y la arquitectura que la conforman, pero a medida que se desarrolló la investigación se han publicado constantes noticias sobre Tor siendo esta red un tema de tendencia en los últimos años, debido a esto fue inevitable conocer trasfondo de estas tecnologías que prestan este servicio en internet.

Nuestras acciones en el mundo virtual impactan cada días más en la vida real, creando un delicado lazo entre estos dos escenarios, desarrollando así una identidad virtual y una real. Un escenario que desde el 2013 gracias a las declaraciones del ex-agente de la NSA Edwar snowden **[10] [11]** sabemos que no es neutral y que funciona de acuerdo a los intereses de algunas empresas o países que vienen desarrollando una vigilancia masiva. Debido a esto podemos determinar con certeza que el anonimato en escenarios que no garantizan la libre expresión se ubica como la principal herramienta para la distribución de información y opiniones salvaguardando la identidad y el bienestar de los usuarios. Internet no debe estar absenta de poder garantizar los mismos derechos en cuanto a privacidad y libre expresión que tiene una persona fuera de este.

Ahora en cuanto a privacidad, ¿Son los usuarios consientes del costo que tiene hoy en día el uso de internet?, lamentablemente no, las redes sociales, wikis, blogs, servicios de alojamiento de videos como youtube, vimeo entre otros han ubicado al usuario en un papel protagónico dándole la posibilidad publicar contenido en internet. Esto sumado a que día a día nos vemos más obligados a estar presentes en este mundo virtual ya que el comercio, las oportunidades de trabajo y hasta la diversión o encuentros con amigos son pactados por estos medios. Son los usuarios los que facilitamos las tareas de espionaje.

Tor cuenta con una gran acogida en diversos sectores, por lo cual es mantenida y presenta un gran número de grupos encargados de ayudar al crecimiento de esta red, aportando investigaciones sobre posibles fallos, vulnerabilidades, mejoras en el diseño y algoritmos utilizados o bien traduciendo y dando soporte en otros idiomas. Ya sea por temas judiciales, políticos o por menciones gracias a su labor, Tor continúa creciendo tanto en usuarios como en reputación a nivel mundial llegando a más de 2 millones de usuarios **[12].**

Es evidente la búsqueda y filtrado de los paquetes característicos de la red Tor por países como China o Russia **[13]**, que ofrecen millonarias sumas de dinero por revelar técnicas que permitan identificar usuarios en la red Tor. Es por esta razón que el proceso de autenticación por medio de TLS en la red Tor ha ido cambiando con el tiempo buscando siempre tener más similitud con el comportamiento normal en cuanto a certificados y handshake estándar en un escenario cliente-servidor. Aun así luego de capturar tráfico de red generado por Tor y observar en detalle su funcionamiento encontramos que en la etapa del handshake SSL/TLS se intercambian mensajes en texto plano dejando en evidencia el uso de este protocolo y vemos que aún es único su comportamiento sobre Tor sin el uso de protocolos de ofuscación, facilitando identificar la creación de una sesión por un usuario.

En esta carrera contra la censura el proyecto Tor a ha desarrollado varias soluciones orientadas a la modificación de la apariencia del tráfico de red que genera. Gracias a la librería “pyptlib” de pyton encargada configurar la comunicación entre el navegador Tor y el proxy, es posible no solo integrar Obfsproxy sino cualquier aplicación que cumpla la función de ofuscación de tráfico logrando así que un gran número de desarrolladores se vinculen a este proyecto sin la necesidad en tener un completo entendimiento del funcionamiento interno de Tor. Así mismo Obfsproxy puede ser implementado independientemente de Tor para ofuscar otro tipo de tráfico objetivo de censura como lo es SSH.

Por último vimos que el uso de los protocolos de ofuscación permiten evitar técnicas avanzadas de censura de tráfico de red como el DPI al enviar el tráfico con una capa extra de cifrado y relleno aleatorio, agregando un grado de dificultad a las tareas de reconocimiento de patrones, además al no seguir la cadena de mensajes establecidos para un handshake TLS evita el bloqueo de este tráfico por técnicas de “fingerprint protocols”.

9. ANEXOS

9.1 Cipher_suite enviada en las versiones 2 y 3

TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_ECDSA_WITH_RC4_128_SHA
TLS_ECDHE_RSA_WITH_RC4_128_SHA
TLS_DHE_RSA_WITH_AES_128_CBC_SHA
TLS_DHE_DSS_WITH_AES_128_CBC_SHA
TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA
TLS_DHE_RSA_WITH_AES_256_CBC_SHA
TLS_DHE_DSS_WITH_AES_256_CBC_SHA
TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_CAMELLIA_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_CAMELLIA_256_CBC_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_RC4_128_SHA
TLS_RSA_WITH_RC4_128_MD5
TLS_EMPTY_RENEGOTIATION_INFO_SCSV

9.2 Configuración óptima para la navegación anónima en navegador Google Chrome

Como primer paso debemos establecer que el intercambio de información es peligroso en páginas que no implementan certificados para verificar su identidad. Enviar información a este tipo de páginas no presenta seguridad.

Historial web de Google: Comúnmente al trabajar con el navegador Google chrome se vincula nuestra cuenta de correo Gmail para acceder a diferentes servicios con la cual Google almacena todas las búsquedas realizadas.

Para desactivar esta opción entramos al link “google.com/history”, nos dirigimos a configuración haciendo click en el icono ⚙ y luego en la opción “Turn off web history”.

Navegación de incognito: Este modo de navegación evita que las páginas visitadas y las descargas realizadas sean archivadas en el historial de navegación. Las nuevas cookies generadas por las páginas visitadas en modo incógnito se borran al cerrar el navegador.

En la ventana de configuración del navegador Google chrome verificar los siguientes puntos:

- Marcar la función "Comprobar la revocación del certificado del servidor", esto permite que al obtener un certificado de una página permita ver si este ha sido revocado o continúa vigente.

- Advertir si cambia de modo seguro a un modo no seguro.

- Verificar el uso de la última versión del protocolo TLS (v 1.2)

9.3 Configuración óptima para la navegación anónima en navegador Mozilla Firefox

Al iniciar la conexión segura con el servidor se envían un conjunto de parámetros entre los cuales está la versión mínima y máxima del protocolo SSL/TLS para implementar en la conexión. Se debe comprobar la configuración de estos en el navegador.

En la barra de direcciones escribe la cadena about:config el cual nos permite ver la configuración del navegador, encontrar los campos security.tls.version.max y security.tls.version.min y verificar que este con los valores 1-3

- Valor **1** equivale a TLS v1.0 como mínima o máxima versión.
- Valor **2** equivale a TLS v1.1 como mínima o máxima versión.
- Valor **3** equivale a TLS v1.2 como mínima o máxima versión.

Existen funciones ocultas en página que permiten redireccionar nuestra navegación a páginas con código malicioso por lo cual se recomienda habilitar la opción “Alertar cuando un sitio web intenta redireccionar o recargar la página”

9.4 HTTPS everywhere

Es una extensión creada en conjunto por el proyecto Tor y la EFF[9], desarrollada para los navegadores Firefox (versión estable), Google Chrome (Beta) y Opera (Beta). Se encarga de redireccionar la conexión de una página a la versión segura de esta forzando el uso de ssl/tls. HTTPS everywhere depende de la configuración del servidor que aloja el recurso solicitado (debe estar configurado para dar respuesta a solicitudes empleando el protocolo ssl/tls) y debe estar incluido en la lista de reglas que trae por defecto esta extensión. Dicho esto, cuando se ingresa una url, HTTPS everywhere evita se envié una solicitud http al servidor, en cambio consulta su lista de reglas para luego solicitar su versión segura si cumple con las condiciones mencionadas anteriormente. Si un sitio cuenta con soporte para ssl/tls y no está incluido en la lista de reglas, se puede crear una regla fácilmente siguiendo el instructivo disponible en la página EFF.

[9] ElectronicFrontierFoundation

Luego de instalar la extensión vemos que el uso de la página "wikipedia" nos devuelve la versión segura https://es.wikipedia.org/wiki/Wikipedia:Portada

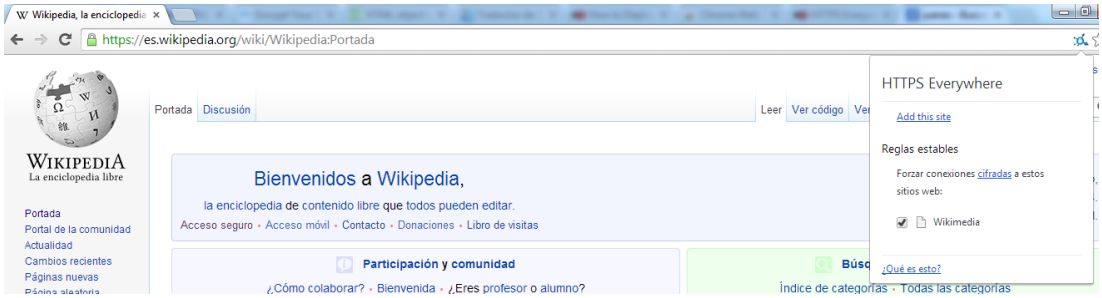

Figura 40.HTTPS everywhere en navegador Google Chrome

Figura 41. HTTPS everywhere en navegador Mozilla Firefox

9.5 NoScript

Es una extensión de código abierto que permite bloquear contenidos de javascript, java, flash que se encuentran embebidos y que se ejecutan al ingresar a una página sin nuestra autorización, comúnmente ejecutados como parte de publicidad o códigos maliciosos independientes de la página que estamos visitando. Desarrollada por Mozilla y disponible en "https://addons.mozilla.org/es/firefox/addon/noscript/" en su última versión 2.6.8.19, funciona para navegadores basados en Mozilla Firefox. Al momento de instalarse se agrega un icono en el navegador y este bloquea automáticamente todas las ejecuciones que no se encuentren en la "whitelist" (lista de las páginas de confianza) la cual trae por defecto algunas

ya configuradas por ejemplo: www.google.com, www.gmail.com, www.yahoo.com, www.youtube.com, entre otros. Por otro lado tenemos la "blacklist" que contiene todas las páginas de las cuales queremos bloquear estos comportamientos. Al ingresar a una página NoScript nos advierte el intento de ejecución de estos y así podremos tomar la decisión de agregarla a la blacklist o whitelist.

Al ingresar a una página NoScript realiza un conteo de todas la etiquetas <script> y <object> en el código HTML de la página y bloquea la ejecución de estos si la dirección de origen se encuentra en la "blacklist".

La etiqueta HTML <script> permite ejecutar un conjunto de instrucciones o apuntar a un script externo por medio de un atributo "src" (especifica una URL de un script externo), usualmente empleado para insertar imágenes, validación de datos o cambios dinámicos de contenido en la página web. **[14]**

La etiqueta HTML <object> permite incluir dentro en una página web contenido de audio, video, ActiveX, PDF, Java applets, Flash u otra página. **[15]**

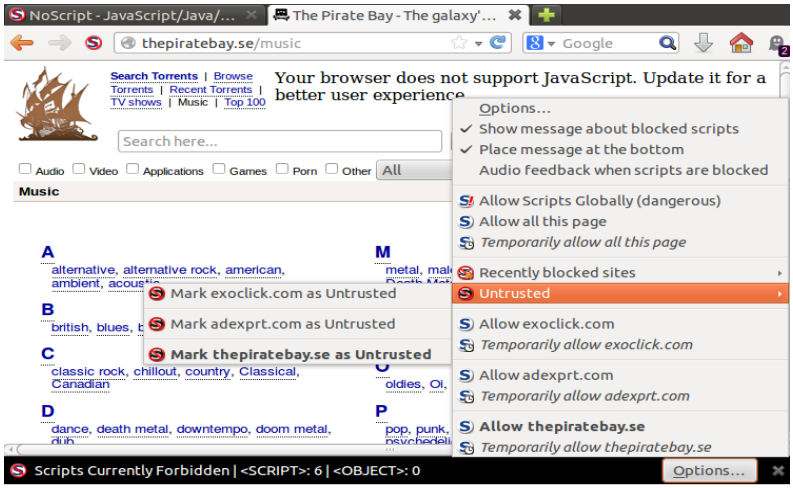

Figura 42. Extensión NoScript activada en navegador Mozilla Firefox

Como podemos ver en este ejemplo, NoScript encontró 6 script que intentaban ejecutarse pero al tener una dirección de origen señalada como "untrusted" fueron bloqueados automáticamente. Al permitir la ejecución de

estas instrucciones y cargar de nuevo la página vemos que aparecen los objetos que fueron bloqueados como la propaganda en la siguiente imagen.

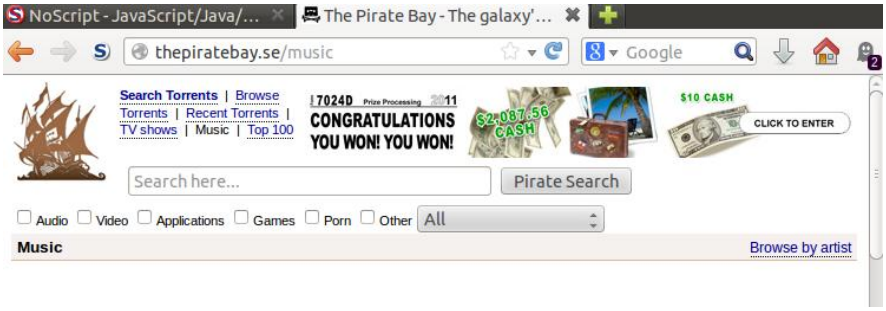

Figura 43. Extensión NoScript desactivada en navegador Mozilla Firefox

9.6 Ghostery

Es una extensión gratuita disponible para los navegadores Internet Explorer, Mozilla Firefox, Google Chrome, Opera y Safari. Ghostery permite monitorear todas las solicitudes a servidores externos embebidas en la página que estamos visitando, que no pertenecen al mismo sitio. Posteriormente brinda una lista de estos sitios comúnmente llamados "rastreadores"[10], identificados en su base de datos como publicitarios o de código malicioso que se encargan de recolectar información de nuestro comportamiento en la red sin nuestro consentimiento. Por último podemos controlar (permitir o bloquear) la ejecución de estos rastreadores.

Ghostery posee la mayor base de datos de rastreadores de toda la red. De manera muy meticulosa seleccionaron, elaboraron perfiles y filtraron más de 1,900 rastreadores[11] y 2,300 patrones de rastreo. **[16]**

[10] Empresas que recolectan información de la navegación de un usuario por medio de cookies o avisos publicitarios.

Luego de su instalación nos presenta un tutorial con los diferentes tipos de bloqueo que tiene frente a una página que ha ejecutado eventos invisibles al usuario.

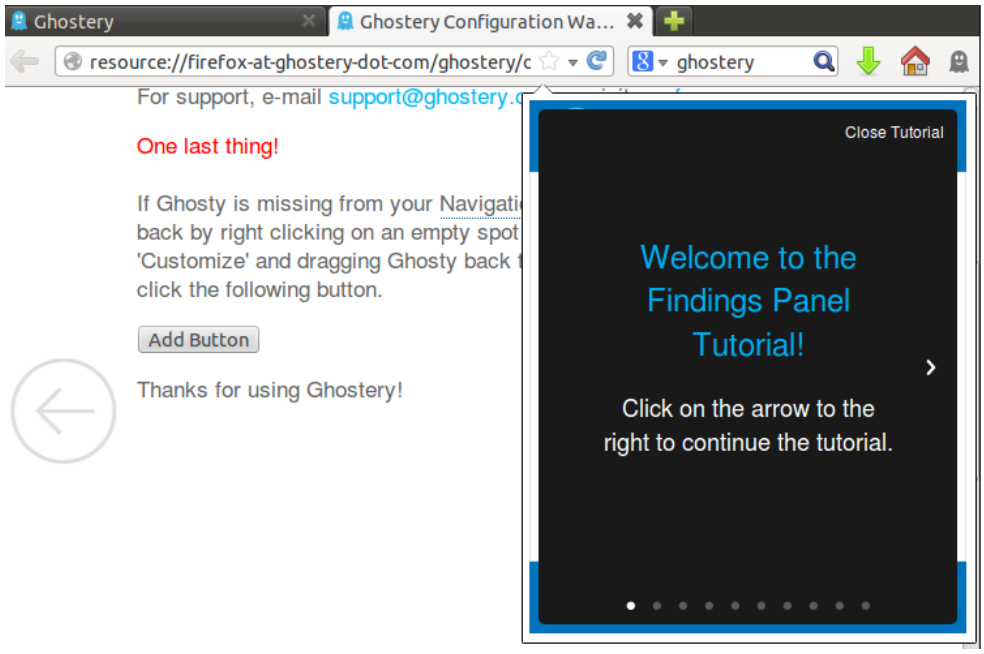

Figura 44. Extensión Ghostery en navegador Mozilla Firefox

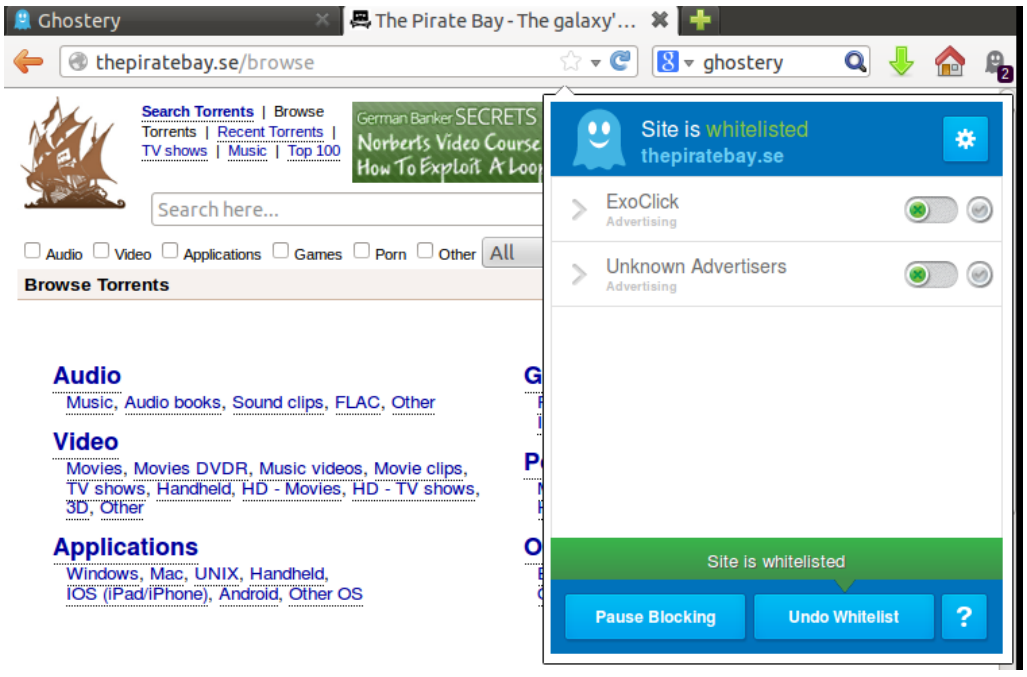

Figura 45. Funcionamiento de la extensión Ghostery en navegador Mozilla Firefox

Luego de ingresar a la página “thepiratebay.se” vemos que se ha detectado dos páginas publicitarias que previamente se encontraban en la lista de filtro de web publicitarias, dándonos varias acciones a tomar:

- Botón **rojo** bloquea el seguimiento.
- Botón **azul** permite el seguimiento.
- Botón **verde** permite siempre el seguimiento de este sitio.
- Botón **Naranja** pausa el bloqueo temporalmente.

Por último en la venta de la aplicación tenemos una vista de las estadísticas del sitio en cuanto a mensajes de publicidad bloqueados.

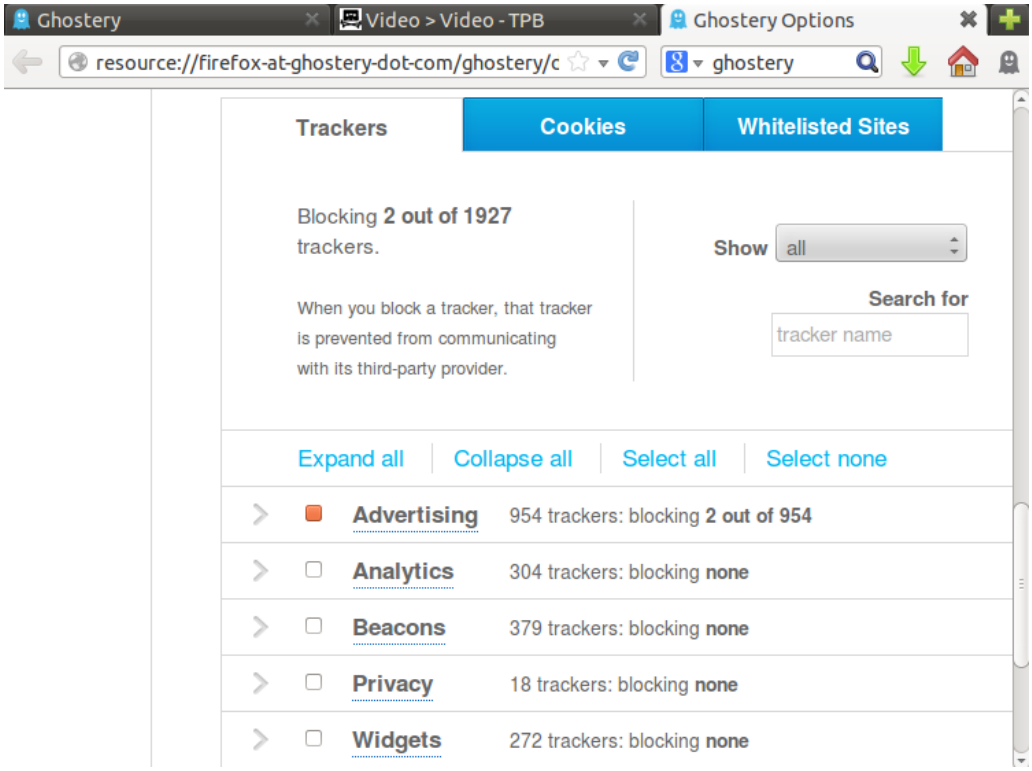

Figura 46. Bloqueo de anuncios por extensión Ghostery en navegador Mozilla Firefox

10. BIBLIOGRAFIA

[1] Stallings, W, Cryptography and Network Security Principles and Practice, 5th Edition. Prentice Hall,2011

[2] Curve25519: new Diffie-Hellman speed records, https://www.iacr.org/archive/pkc2006/39580209/39580209.pdf (Consultada 08/12/2014)

[3] HMAC-based Extract-and-Expand Key Derivation Function (HKDF), http://tools.ietf.org/html/rfc5869 (Consultada 04/02/2015)

[4] Version 2 Tor connection protocol, https://gitweb.torproject.org/torspec.git/tree/proposals/130-v2-conn-protocol.txt (Consultada 05/01/ 2015)

[5] Blocking resistant TLS certificate usage, https://gitweb.torproject.org/torspec.git/tree/proposals/124-tls-certificates.txt (Consultada 06/01/2015)

[6] Tor Protocol Specification, https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt (Consultada 13/09/ 2014)

[7] The Transport Layer Security (TLS) Protocol, https://tools.ietf.org/html/rfc5246 (Consultada 05/01/2015)

[8] Renegotiating TLS, https://kryptera.se/Renegotiating%20TLS.pdf (Consultada 20/01/2015)

[9] Eliminate TLS renegotiation for the Tor connection handshake, https://gitweb.torproject.org/torspec.git/tree/proposals/169-eliminating-renegotiation.txt (Consultada 02/12/2014)

[10] NSA chief says Snowden leaked up to 200,000 secret documents, http://www.reuters.com/article/2013/11/14/us-usa-security-nsa-idUSBRE9AD19B20131114 (Consultada 20/11/2014)

[11] Edward Snowden: the whistleblower behind the NSA surveillance revelations, http://www.theguardian.com/world/2013/jun/09/edward-snowden-nsa-whistleblower-surveillance (Consultada 19/11/2014)

[12] Tor Metrics, https://metrics.torproject.org/ (Consultada 20/11/2014)

[13] Russia offers 3.9m roubles for 'research to identify users of Tor', http://www.theguardian.com/world/2014/jul/25/russia-research-identify-users-tor (Consultada 22/11/2014)

[14] HTML <script> Tag, http://www.w3schools.com/TAgs/tag_script.asp (Consultada 18/11/2014)

[15] HTML <object> Tag, http://www.w3schools.com/TAgs/tag_object.asp (Consultada 18/11/ 2014)

[16] Ghostery, https://www.ghostery.com/es/features (Consultada 18/11/2014)

[17] More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE), https://www.ietf.org/rfc/rfc3526.txt (Consultada 22/07/2015)

[18] Noscript, http://noscript.net/ (Consultada 17/11/2014)

[19] Security.tls.version.*, http://kb.mozillazine.org/Security.tls.version.* (Consultada 10/01/2015)

[20] https-everywhere, https://www.eff.org/https-everywhere (Consultada 17/11/ 2014)

[21]Ayuda de google, https://support.google.com/ (Consultada 13/11/2014)

[22] Tor:Bridges, https://www.torproject.org/docs/bridges#BridgeIntroduction (Consultada 23/07/2015)

[23] Tor: Pluggable Transports, https://www.torproject.org/docs/pluggable-transports.html.en (Consultada 23/07/2015)

[24] obfs3 (The Threebfuscator), https://gitweb.torproject.org/pluggable-transports/obfsproxy.git/tree/doc/obfs3/obfs3-protocol-spec.txt (Consultada 25/07/2015)

[25] Obfsproxy: the next step in the censorship arms race, https://blog.torproject.org/blog/obfsproxy-next-step-censorship-arms-race (Consultada 25/07/2015)