# Servizi Onion

## Dalla teoria all'implementazione

**Leonardo Migliorelli**[1]
leonardo.migliorelli@studenti.unicam.it

# Computer Science @ Unicam

Tesi Unicam - Servizi Onion, dalla teoria dall'implementazione

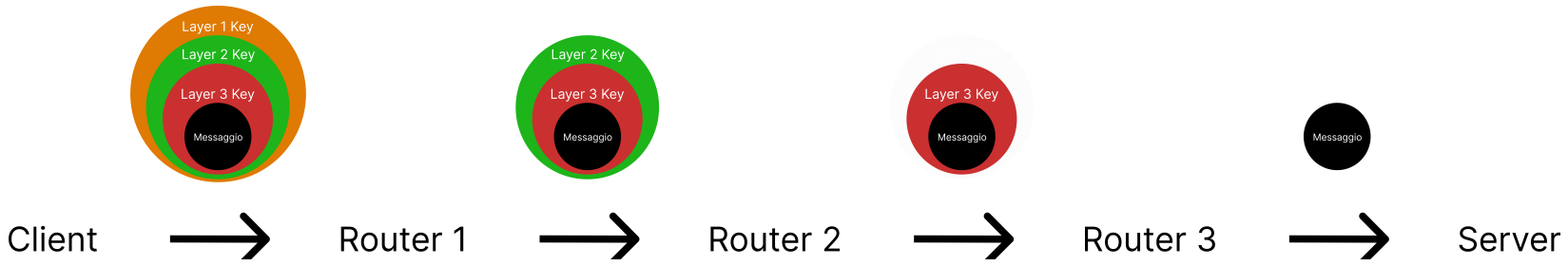
- Relatore: Marcantoni Fausto
  https://computerscience.unicam.it/marcantoni/
- Studente: Leonardo Migliorelli Mat.113920
  leonardo.migliorelli@studenti.unicam.it

# Indice

# Introduzione

In questa tesi spiegheremo il funzionamento delle reti **Onion** e parleremo in particolare della rete **Tor**, in fine mostreremo com'è possibile implementare un **servizio Onion/Tor**.
Le reti Onion sono state create per risolvere le due più grandi vulnerabilità di Internet che gravano sulla **privacy** e sull'**anonimato**, ovvero l'**analisi del traffico** e le **intercettazioni**.
Una rete di questo tipo nasconde infatti gli indirizzi e il contenuto di ogni richiesta, consentendo all'utente di navigare in rete senza essere tracciato.

# Onion Routing



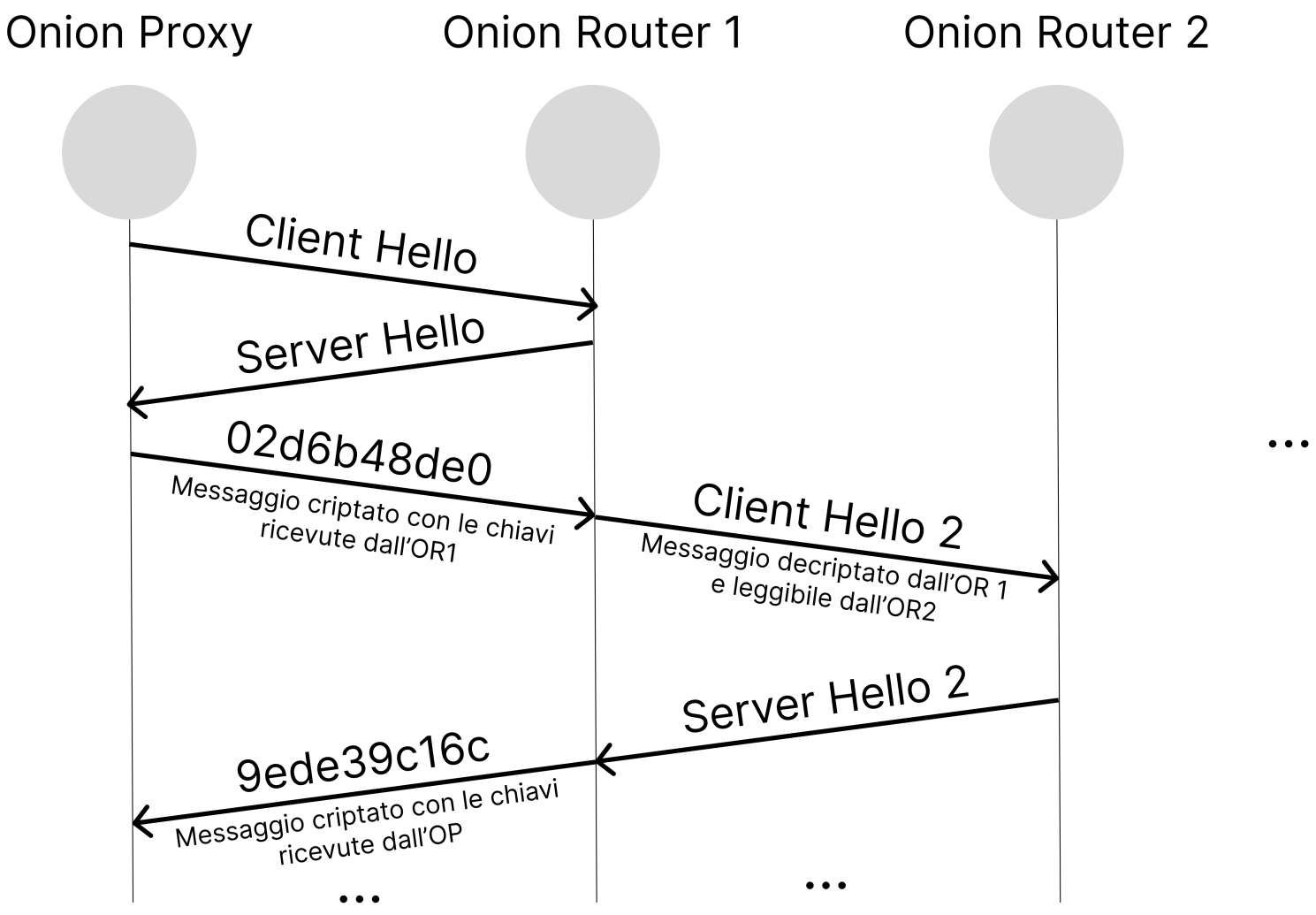

La rete Onion è una rete distribuita composta da nodi chiamati **Onion Router**, collegati tra loro tramite i circuiti creati dagli **Onion Proxy**.
Ogni pacchetto che passa nel circuito viene decriptato in maniera sequenziale dai relativi nodi fino ad arrivare all'exit node che si occupa di instradare il pacchetto nella rete Internet.
Grazie a questo meccanismo nessun nodo conosce contemporaneamente l'indirizzo del mittente e del destinatario.

Il proxy onion è composto da tre componenti:

- Un proxy che genera e gestisce le connessioni, ha necessità di conoscere la topologia della rete Onion.
- **Application Specific Proxy**, si occupa di convertire le richieste delle applicazioni in pacchetti Onion.
- **Application Specific Privacy Filter**, un proxy opzionale che sanifica lo stream di dati rimuovendo informazioni che potrebbero identificare il client.

# Creazione del circuito

La generazione del circuito è un processo **iterativo** e **progressivo** in cui il proxy server sceglie i nodi del circuito. A partire dal primo nodo viene instaurata una connessione **TLS** e vengono scambiate le **chiavi simmetriche** tramite un processo **asimmetrico** (in maniera simile allo scambio di chiavi di HTTPS), successivamente si usano queste chiavi per cifrare il messaggio che verrà inviato al primo nodo che lo decripta e inoltra al secondo nodo.
Questo processo continua fino all'exit node, a questo punto il circuito è completo e può iniziare a trasmettere i pacchetti.


Onion Proxy
Onion Router 1
Onion Router 2
Client Hello
Server Hello
02d6b48de0
Messaggio criptato con le chiavi ricevute dall'OR1
Client Hello 2
Messaggio decriptato dall'OR 1 e leggibile dall'OR2
Server Hello 2
9ede39c16c
Messaggio criptato con le chiavi ricevute dall'OP
...
...
...

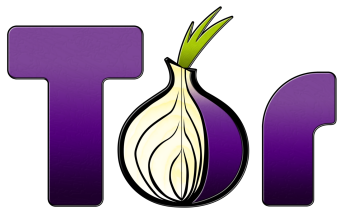
# Chaum Mix

La rete Onion è basata sullo studio di David Chaum che propose un sistema di comunicazione anonima basato sulla crittografia. Nella sua conclusione una rete di questo tipo doveva avere le seguenti caratteristiche:

- **Sealing**, una tecnica con cui il messaggio viene annesso ad una stringa casuale prima di essere criptato per aumentarne la sicurezza.
- **Indirizzo non tracciabile**, un indirizzo generato dal client criptando quello reale, rendendone possibile la decrittazione solo dal primo nodo. Viene usato come indirizzo di risposta dal destinatario.

# Tor

La rete Tor è la più famosa implementazione di Onion, le principali migliorie sono state:



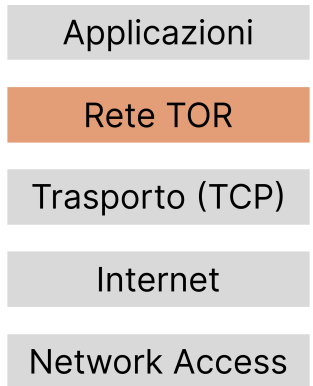
- Circuiti telescopici
- Utilizzo del protocollo SOCKS
- Controllo di congestione
- Directory server
- Controllo d'integrità end-to-end

L'obiettivo principale della rete TOR è quello di garantire l'anonimato e scoraggiare eventuali attaccanti, per questo la rete doveva essere semplice da usare, cosi da incrementare il numero di nodi.

# Network Design

La rete Tor è
una rete che esiste al di sopra delle reti esistenti,
per questo viene chiamata **overlay network**.
I pacchetti che viaggiano
nella rete Tor sono chiamati **celle**, hanno una
dimensione fissa 512 bytes, e sono di due tipi:

- **Control**, gestisce il circuito
- **Relay**, trasporta stream dati



# Controllo di congestione

Non potendo in una rete Onion cambiare un router quando non è più in grado di gestire il carico, è stato necessario implementare un controllo di congestione che blocca i circuiti o gli stream dati quando questi inviano troppi dati usando due finestre:

- Packaging window, tiene traccia del numero di celle che un OR può inviare all'OP
- Delivery window, tiene traccia del numero di celle che un OR può inviare fuori dalla rete

L'unica differenza tra il controllo a livello di circuito o di stream è la dimensione delle finestre.

# Tor Relay

La rete TOR si basa su un insieme di nodi gestiti da volontari chiamati TOR Relay, possono essere di tre tipi:

- **Non-exit Relay**, i nodi interni della rete che a loro volta si dividono in:
  - **Guard Relay**, il primo nodo del circuito
  - **Middle Relay**, i nodi intermedi del circuito
- **Exit Relay**, i nodi di uscita della rete Tor, instradano il traffico nella rete comune. Essendo gli unici IP visibili all'esterno sono i più esposti a rischi legali.
- **Bridge Relay**, nodi non pubblici che non possono quindi essere bloccati

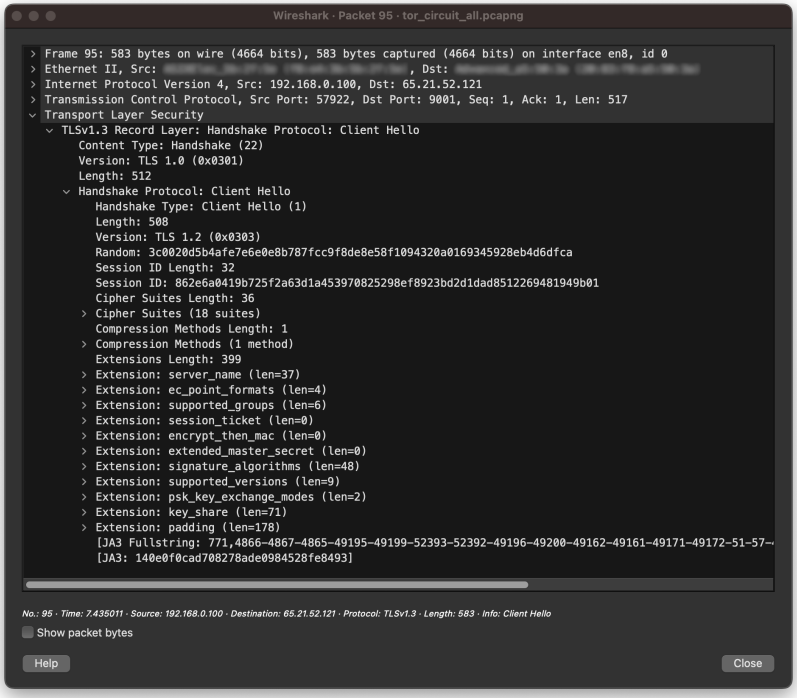
Tutte le informazioni sui Relay esistenti sono visitabili al seguente indirizzo:
`https://metrics.torproject.org/rs.html`

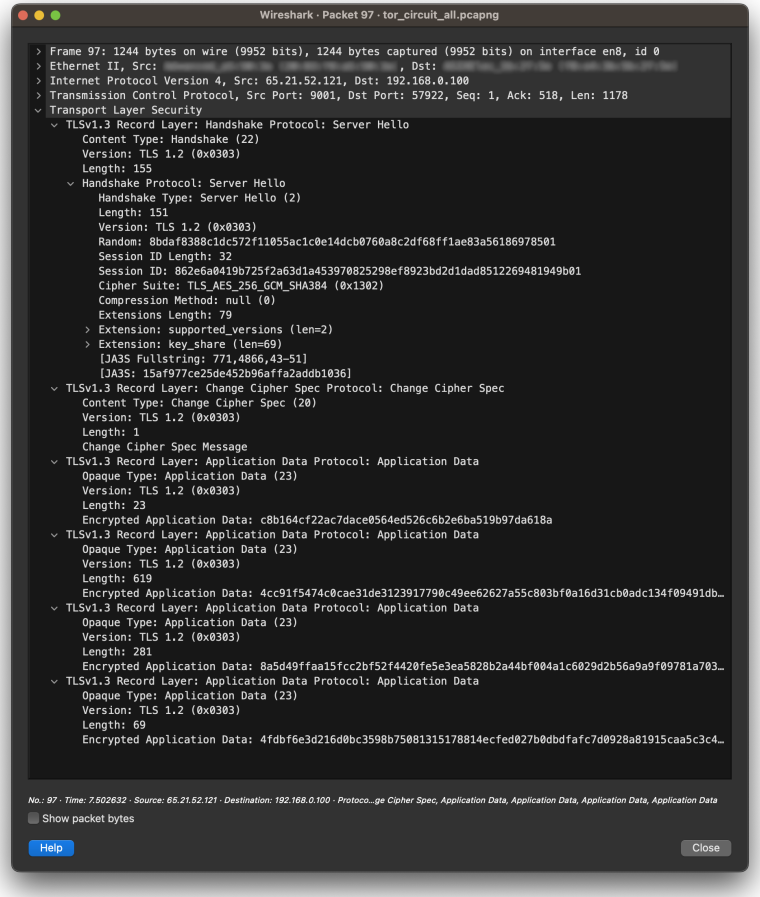
# Directory Servers

I Directory Server sono un sottogruppo di onion router che tracciano i cambiamenti nella **topologia di rete** e agiscono da **DNS server** per i servizi onion. Mantengono infatti i **descriptor**, pacchetti generati dai servizi onion criptati con la propria chiave privata, che contengono gli introduction points e la chiave pubblica.
Il sistema di generazione di indirizzi onion fornisce un **meccanismo di sicurezza** per evitare che un malintenzionato possa alterare i descriptor e reindirizzare gli utenti ai propri introduction points, infatti se cosi fosse la chiave pubblica nascosta nell'indirizzo onion non sarebbe in grado di decifrare il descriptor.

# Dimostrazione Wireshark

Wireshark ci permette di vedere come viene generata la connessione **TLS**, in cui il client invia alcune informazioni tra cui la versione TLS, la lista dei Cipher Suite supportati e la chiave pubblica. Il proxy riceve poi il Server Hello, ovvero la risposta del server con le relative informazioni di criptografia. Successivamente Tor usa le informazioni ottenute per stabilire una connessione TLS tra client e server, impedendoci di vedere il traffico dati.


Wireshark · Packet 95 · tor_circuit_all.pcapng
Frame 95: 583 bytes on wire (4664 bits), 583 bytes captured (4664 bits) on interface en8, id 0
Ethernet II, Src: [redacted], Dst: [redacted]
Internet Protocol Version 4, Src: 192.168.0.100, Dst: 65.21.52.121
Transmission Control Protocol, Src Port: 57922, Dst Port: 9001, Seq: 1, Ack: 1, Len: 517
Transport Layer Security
TLSv1.3 Record Layer: Handshake Protocol: Client Hello
Content Type: Handshake (22)
Version: TLS 1.0 (0x0301)
Length: 512
Handshake Protocol: Client Hello
Handshake Type: Client Hello (1)
Length: 508
Version: TLS 1.2 (0x0303)
Random: 3c0020d5b4afe7e6e0e8b787fcc9f8de8e58f1094320a0169345928eb4d6dfca
Session ID Length: 32
Session ID: 862e6a0419b725f2a63d1a453970825298ef8923bd2d1dad8512269481949b01
Cipher Suites Length: 36
Cipher Suites (18 suites)
Compression Methods Length: 1
Compression Methods (1 method)
Extensions Length: 399
Extension: server_name (len=37)
Extension: ec_point_formats (len=4)
Extension: supported_groups (len=6)
Extension: session_ticket (len=0)
Extension: encrypt_then_mac (len=0)
Extension: extended_master_secret (len=0)
Extension: signature_algorithms (len=48)
Extension: supported_versions (len=9)
Extension: psk_key_exchange_modes (len=2)
Extension: key_share (len=71)
Extension: padding (len=178)
[JA3 Fullstring: 771,4866-4867-4865-49195-49199-52393-52392-49196-49200-49162-49161-49171-49172-51-57-
[JA3: 140e0f0cad708278ade0984528fe8493]
No.: 95 · Time: 7.435011 · Source: 192.168.0.100 · Destination: 65.21.52.121 · Protocol: TLSv1.3 · Length: 583 · Info: Client Hello
Show packet bytes
Help
Close


Wireshark · Packet 97 · tor_circuit_all.pcapng
Frame 97: 1244 bytes on wire (9952 bits), 1244 bytes captured (9952 bits) on interface en8, id 0
Ethernet II, Src: [redacted], Dst: [redacted]
Internet Protocol Version 4, Src: 65.21.52.121, Dst: 192.168.0.100
Transmission Control Protocol, Src Port: 9001, Dst Port: 57922, Seq: 1, Ack: 518, Len: 1178
Transport Layer Security
TLSv1.3 Record Layer: Handshake Protocol: Server Hello
Content Type: Handshake (22)
Version: TLS 1.2 (0x0303)
Length: 155
Handshake Protocol: Server Hello
Handshake Type: Server Hello (2)
Length: 151
Version: TLS 1.2 (0x0303)
Random: 8bdaf8388c1dc572f11055ac1c0e14dcb0760a8c2df68ff1ae83a56186978501
Session ID Length: 32
Session ID: 862e6a0419b725f2a63d1a453970825298ef8923bd2d1dad8512269481949b01
Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
Compression Method: null (0)
Extensions Length: 79
Extension: supported_versions (len=2)
Extension: key_share (len=69)
[JA3S Fullstring: 771,4866,43-51]
[JA3S: 15af977ce25de452b96affa2addb1036]
TLSv1.3 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
Content Type: Change Cipher Spec (20)
Version: TLS 1.2 (0x0303)
Length: 1
Change Cipher Spec Message
TLSv1.3 Record Layer: Application Data Protocol: Application Data
Opaque Type: Application Data (23)
Version: TLS 1.2 (0x0303)
Length: 23
Encrypted Application Data: c8b164cf22ac7dace0564ed526c6b2e6ba519b97da618a
TLSv1.3 Record Layer: Application Data Protocol: Application Data
Opaque Type: Application Data (23)
Version: TLS 1.2 (0x0303)
Length: 619
Encrypted Application Data: 4cc91f5474c0cae31de3123917790c49ee62627a55c803bf0a16d31cb0adc134f09491db…
TLSv1.3 Record Layer: Application Data Protocol: Application Data
Opaque Type: Application Data (23)
Version: TLS 1.2 (0x0303)
Length: 281
Encrypted Application Data: 8a5d49ffaa15fcc2bf52f4420fe5e3ea5828b2a44bf004a1c6029d2b56a9a9f09781a703…
TLSv1.3 Record Layer: Application Data Protocol: Application Data
Opaque Type: Application Data (23)
Version: TLS 1.2 (0x0303)
Length: 69
Encrypted Application Data: 4fdbf6e3d216d0bc3598b75081315178814ecfed027b0dbdfafc7d0928a81915caa5c3c4…
No.: 97 · Time: 7.502632 · Source: 65.21.52.121 · Destination: 192.168.0.100 · Protoco…ge Cipher Spec, Application Data, Application Data, Application Data, Application Data
Show packet bytes
Help
Close

# Implementazione

Per implementare un servizio onion è necessario utilizzare il **Proxy Onion** per inoltrare le richieste dalla rete Tor al server web, il proxy gestisce la generazione delle chiavi, dell'indirizzo e la definizione e connessione con gli **introduction points**, oltre alla generazione del descriptor e la sua pubblicazione nei Directory Servers.
In particolare la nostra implementazione userà **Onion V3** (dato che Onion V2 è stato deprecato) con alcune migliorie tra cui la maggior sicurezza degli indirizzi.
Useremo un server **Linux EC2** di AWS per ospitare il proxy onion e un **server nginx**.

Nel sistema è necessario installare il pacchetto nginx (web server) e dopo una serie di configurazioni dei repository possiamo installare tor.
Configuriamo il file di configurazione torrc con HiddenServiceDir /var/lib/tor/hidden_service (per indicare la directory dove verranno salvate le chiavi) e HiddenServicePort 80 unix:/var/run/website.sock (per indicare il tipo e il modo di connessione con il web server).