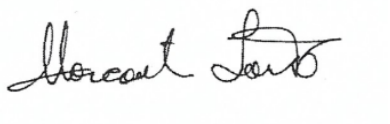
Universià degli Studi di Camerino

FACOLTÀ DI SCIENZE E TECNOLOGIE
Corso di Laurea Triennale in Informatica

Tor,
da frontend Android a backend

Studente:
Samuele Plescia
Matricola 101438

Relatore:
Fausto Marcantoni



Anno Accademico 2020-2021

"Io non sono veramente libero se non quando tutti gli esseri umani che mi circondano, uomini e donne, sono liberi allo stesso modo..., di modo che, quanto più numerosi sono gli uomini liberi che mi stanno intorno e quanto più profonda e più vasta è la loro libertà, e quanto è più estesa, tanto più profonda e più grande è la mia libertà"

M. Bakunin

Indice

Sommario

Sviluppo di un'applicazione Android configurata per navigare con Tor e quindi tramite protocollo Onion per lo scambio di dati con un HiddenService hostato su Raspberry Pi 4. Inizierò con la compilazione della libreria tor con Android NDK da usare per l'applicazione Android Java, passando per il tunnelling creato da Tor e dal sistema di routing della rete Onion fino ad attraversare la porta virtuale dell'HiddenService per raggiungere il backend creato con NodeJS e sqlite.

1 Motivazione

L'ispirazione per questa tesi è il principio di un'idea che avrei comunque portato avanti nel tempo libero. Il contenuto di questa tesi è quindi una piccola parte, l'inizio, di questo mio progetto. Un'altra delle motivazioni principali è l'onnicomprensività di questo argomento sia dal lato tecnico: Android Java, NodeJS, Sqlite, Tor, TLS, OpenSSL, protocolli, moduli, compilazione, Linux... Sia dal lato umano: influenza, anonimato, libertà, etica, comunicazione, informazione, persone. Tralasciando la sempreverde questione dell'anonimato in rete, l'unica cosa su cui spero tutti possiamo essere d'accordo è che l'anonimato è un potente strumento.

In ogni caso, come scritto sul sito ufficiale, i criminali hanno strumenti migliori per rimanere anonimi. [1]

2 Introduzione

Ho incontrato alcuni informatici e/o sviluppatori che non sapevano cosa fosse Tor, questo significa che viviamo ancora nella parte fortunata del pianeta.

Tor è stato e viene utilizzato nelle situazioni più varie:

- Durante la primavera araba non solo proteggeva l'identità delle persone online, ma consentiva loro anche di accedere a risorse critiche, social media e siti web che erano stati bloccati. [2] [3]
- Dopo le rivelazioni di Edward Snowden [4] (che hanno dimostrato che nel 2013 Tor non poteva essere violato nemmeno dalla NSA) l'uso di Tor è aumentato.
- Esiste un hidden service per inviare dati, o informazioni in anonimo tramite Tor a Wired [5]
- Per combattere le restrizioni imposte da Cina (The Great Firewall) ed Iran
- Dai 50 ai 60 mila Iraniani lo usano giornalmente [6]
- Molti altri usi tra cui Facebook, la repository di Debian, SecureDrop, ProPublica, GlobaLeaks. . . [7]

In generale per proteggere la comunicazione senza dover essere esperti del settore: i giornalisti, soprattutto di alcuni stati, lavorano giornalmente con questo software.

3 Persone e Progetti

3.1 The Tor Project

Il Tor Project è un'organizzazione no profit di ricerca-educazione con sede a Seattle, fondata nel 2006 dagli informatici Roger Dingledine [8], Nick Mathewson, Paul Syverson ed altre 4 persone.
Il Tor Project si occupa principalmente della manutenzione del software per la rete di anonimato Tor.

Figura 2: Roger Dingledine

source: `https://it.wikipedia.org/wiki/Roger_Dingledine`

3.2 Tor: The Onion Router

Tor è un software gratuito ed open source (licenza BSD 3 clausole), serve a navigare in modo anonimo su Internet (ma non a proteggere la confidenzialità dei dati). Dirige il traffico Internet attraverso una rete overlay gratuita, mondiale e volontaria, composta da più di seimila relay e quasi duemila bridge (stimati, in quanto ad esempio in Cina cambiano in continuazione). [9] E' conosciuto anche come "la più grande implementazione di onion routing ".

Figura 3: Tor logo

source: `https://www.torproject.org/static/images/tor-project-logo-onions.png`

3.3 Onion Routing

Sviluppato a metà degli anni '90 presso il Laboratorio di Ricerca Navale degli Stati Uniti, l'Onion routing è una tecnica (protocollo) di anonimizzazione delle comunicazioni in una rete di telecomunicazioni.

3.4 Tor Browser

Su desktop si tratta di un'applicazione basata su Firefox ma con delle modifiche applicate [10], ad esempio:

- Patch per la protezione contro il cross-site tracking (mediante una misura nota come isolamento di prima parte)
- Resistenza al fingerprinting ovvero si mira a rendere tutti gli utenti uguali: tutti gli utenti appaiono con le stesse informazioni del browser e del dispositivo utilizzato
- Prevenzione della memorizzazione di dati privati su disco
- Una connessione alla rete Tor

Su Android è anch'essa basata su Firefox [11] ma non ha le stesse caratteristiche della versione desktop nel momento in cui scrivo. Fino al 2019, per collegarsi a Tor tramite Browser c'era bisogno di Orbot/Orfox. [12]

3.5 Orbot

E' un'applicazione che fornisce un proxy HTTP locale e le interfacce proxy standard SOCKS4A/SOCKS5 per entrare nella rete Tor.

Quindi permette agli utenti smartphone di utilizzare qualsiasi applicazione, che consenta la configurazione di un proxy, sulla rete Tor.

La prima release è stata pubblicata nel 2008 (stessa licenza di Tor) dal gruppo The Guardian Project.

3.6 The Guardian Project

E' un collettivo globale di sviluppatori software, designer, sostenitori, attivisti e formatori che sviluppano software di sicurezza per smartphone liberi ed open source.

4 Motivazioni tecniche

Quello che mi appresto a fare è creare un'applicazione Android Java che implementi la libreria Tor compilata da codice sorgente.

A questo punto sorge una domanda: perché non usare semplicemente Tor Browser, od Orbot?

4.1 Javascript

Con la creazione di un frontend nativo e un backend REST si porta a zero lo scambio di codice Javascript tra client e server che è sempre fonte di dubbio.

4.2 Grafica

Il sistema di anonimato di Onion è pesante, sarebbe un enorme spreco di risorse far scaricare la grafica.

4.3 Facilità di utilizzo

Questi è uno dei motivi più importanti.

Usando un'app dedicata non c'è bisogno che l'utente finale si ricordi il domain name dell'hidden service, non ha bisogno di sapere la differenza tra Orbot, Orfox, o Tor Browser.

Banalmente l'utente finale avrà la sensazione di star usando un'app come tutte le altre, in caso contrario se la persona dovesse cominciare ad informarsi verrebbe influenzata da descrizioni esagerate e da articoli di giornale spaventosi e crederebbe di star facendo qualcosa di illegale.

4.4 Curiosità

Il semplice voler rispondere alla domanda: ma come funziona?

4.5 Scopi didattici

"L'obiettivo di questo documento è da ritenersi a titolo didattico, informativo, illustrativo, d'intrattenimento"

Cit.

5 Strumenti

5.1 Debian

Inizialmente avevo provato con ArchLinux ma dopo due giorni di prove, dopo l'ennesimo errore (una lista di errori è inserita nell'appendice A), decisi di cambiare sistema operativo ed usare Debian con quest'ultimo riuscii a fare la compilazione in una mattinata: questo perché lo script che aiuta la compilazione di torlib è stato scritto avendo in mente l'utilizzo su Debian.

5.2 Android NDK

E' un set di strumenti che ti consente di implementare parti della tua app in codice nativo utilizzando linguaggi come C e C++.

Questo da la possibilità di utilizzare le caratteristiche di Linux su Android al massimo. Android NDK risulterà necessario per la compilazione delle varie librerie tutte scritte appunto in C/C++.

5.3 NodeJS ed Sqlite

Per il backend ho semplicemente cercato delle opzioni che mi dessero la possibilità di implementare facilmente un servizio.

L'agilità e la facilità di implementazione di questi due strumenti è impressionante.

5.4 Raspberry Pi 4 Model B

La versione da 4GB di ram con sistema operativo Raspian.

Sarà il server che hosterà l'hidden service.

6 Preparazione e Compilazione

6.1 Preparazione alla compilazione

L'installazione di Debian non sarà affrontata in questo documento in quanto prenderebbe diverso spazio.

La prima cosa da fare è aggiornare il sistema:

```
sudo apt-get update
sudo apt-get dist-upgrade
```

dist-upgrade è diverso dal semplice upgrade in quanto rimuove i pacchetti deprecati.

Scaricare quindi la repository contenente sia il codice sorgente di Tor per Android sia un esempio di implementazione in una applicazione:

```
sudo apt install git
git clone https://github.com/guardianproject/tor-android.git
```

Si scaricano gli strumenti per la compilazione:

```
sudo apt-get install build-essential autotools-dev automake
   autogen autoconf libtool gettext-base autopoint
sudo apt-get install pkg-config
sudo apt-get install openjdk-11-jdk
```

Per scaricare Android SDK e NDK si consiglia l'uso dell'interfaccia grafica scaricando Android Studio.

Ho provato ad usare sdkmanager ma per l'utilizzo che se ne fa non conviene sprecare tempo ad imparare questo strumento.

```
tar -zxvf android-studio-ide-linux.tar.gz
# comando di avvio di Android Studio
./android-studio/bin/studio.sh
```

6.1.1 Installazione delle SDK e di Android NDK

Dopo aver avviato Android Studio si procede all'installazione di Android NDK: navigare su Tools e poi SDK Manager come mostrato in figura 4.

1. SDK Tools
2. Spuntare "Show Package Details"
3. Selezionare NDK (side by side) la versione 21.4.7075529 come mostrato in figura 5
4. Apply & Accept
5. Installare

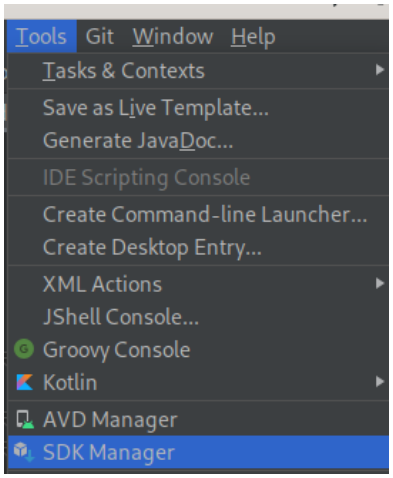


Figura 4: Tools > SDKManager

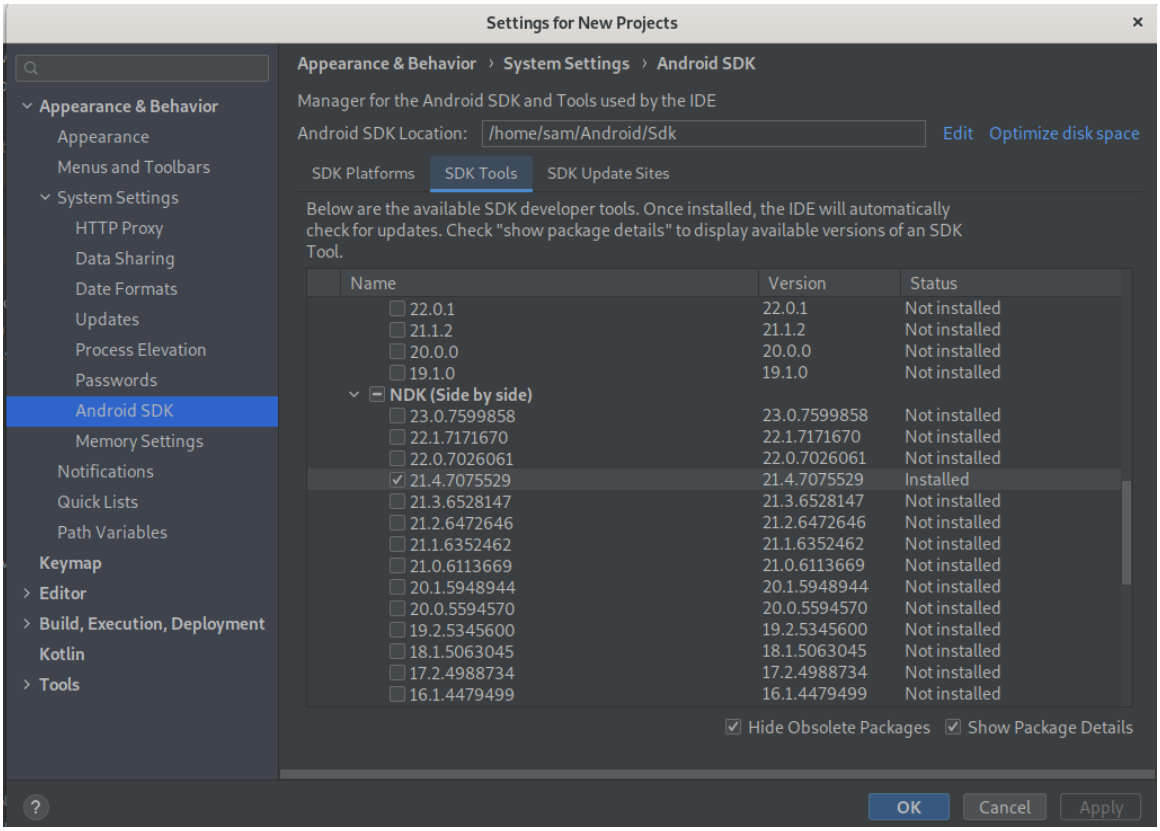


Figura 5: Screenshot della finestra di installazione di Android NDK

Settare le variabili d'ambiente:

```
cd ~
nano ./.bashrc
export ANDROID_HOME=~/Android/Sdk
export ANDROID_NDK_HOME=${ANDROID_HOME}/ndk/21.4.7075529
export PATH='${ANDROID_HOME}/tools/:${ANDROID_HOME}/platform-
    tools/:${PATH}'

# NB. non ci sara' bisogno di settare JAVA_HOME
# in quanto Debian installa le openjdk
# nella cartella /usr/bin presente nel PATH
# sin da subito
```

6.2 Compilazione

Tor per android è compilato andando ad utilizzare diversi altri moduli tra cui:

- tor

  In questo caso si tratta di una fork da parte di The Guardian Project della repository che contiene il source code di Tor creato da Tor Project. Le modifiche fatte al source code permettono la compatibilità con Android.

- openssl (compilato staticamente e patchato per Android)

  E' un'implementazione open source dei protocolli SSL e TLS (in ogni caso viene configurato per non usare i protocolli SSL).

  "compilato staticamente" significa che nella configurazione il path è indicato esplicitamente e l'opzione è abilitata:

  ```
  ./configure --with-openssl-dir=path/to/lib --enable-static-
      openssl
  ```

- libevent (compilato staticamente e patchato per Android)

  E' una libreria software che fornisce la notifica degli eventi asincrona. Serve ad implementare il paradigma nonblocking tramite i sockets. [13]

  Generalmente ci sono 3 modi per implementare questo paradigma:

  - tramite la creazioni di nuovi processi (fork)
  - tramite la creazione di threads (multithreading)

  - tramite i sockets (impostandoli a nonblocking)

    La chiamata per impostare i socket a nonblocking in C su Unix è:

    ```
    // fd e' il file descriptor per il socket
    fcntl(fd, F_SETFL, O_NONBLOCK);
    ```

- xz e zstd

  sono programmi di compressione dati

Per scaricare tutti questi sottomoduli automaticamente si può usare il comando fetch dello script fornito nella repository:

```
cd path/to/tor-android
./tor-droid-make.sh fetch
```

Per iniziare la build usare il comando:

```
./tor-droid-make.sh build
```

6.3 Esportazione su Window

Finita la compilazione ci troveremo con questi file

```
tor-android
├── external
│   └── bin
│       ├── include
│       └── lib
│           ├── arm64-v8a
│           ├── armeabi-v7a
│           ├── x86
│           └── x86-64
├── gradle
├── sampletorapp
└── tor-android-binary
    └── src
        ├── androidTest
        └── main
            ├── java
            ├── jniLibs (symlink alla cartella ./libs)
            ├── libs
            │   ├── arm64-v8a (symlink alla cartella ./tor-android/external/lib/arm64-v8a)
            │   ├── armeabi-v7a (symlink alla cartella ./tor-android/external/lib/armeabi-v7a)
            │   ├── x86 (symlink alla cartella ./tor-android/external/lib/x86)
            │   └── x86-64 (symlink alla cartella ./tor-android/external/lib/x86_64)
            └── AndroidManifest.xml
```

Per l'esportazione su Window basterà:

1. Eliminare tutti i symlink
2. Spostare la cartella ./tor-android/external/lib dentro ./tor-android/tor-android-binary/src/main
3. Rinominare la cartella ./tor-android/tor-android-binary/src/main/lib in jniLibs

6.4 Creazione di un progetto aggiornato

Molti file risultano inutili dopo aver compilato la libreria e il progetto nella repository ne contiene molti.
Per creare quindi un progetto pulito ed aggiornato bisogna eseguire alcuni passaggi.
Si crea quindi un nuovo progetto su Android Studio e si copiano all'interno tutti i file che riguardano Tor:

- La cartella "external"

  Di questa cartella ci interessa solo la sottocartella "lib"

- Il sottoprogetto "tor-android-binary"

Il sottoprogetto, per funzionare, deve essere aggiunto alle opzioni di compilazione. Nel caso di Gradle si aggiunge il nome della cartella del sottoprogetto:

- nel file **./project_name/settings.gradle**

```
include ':tor-android-binary', ':your_app_name'
```

- e nel file **./project_name/your_app_name/build.gradle**

```
dependencies {
 implementation project(':tor-android-binary')
}
```

Nel file **./project_name/tor-android-binary/build.gradle** possono essere eliminati tutti i tasks e si possono configurare le versioni di Gradle e Android più recenti. A questo punto il progetto risulta facilmente aggiornabile.

6.5 Perché tenere tutto aggiornato

Più un sistema è esteso e più componenti diversi sono presenti nella sua configurazione e più risulta aperto a possibili attacchi.
Ogni punto è attaccabile:

1. Network (la comunicazione)

   Soprattutto quando si parla di Tor buona parte degli attacchi sono eseguiti agendo sul sistema di comunicazione. Su internet sono documentati molti tipi di attacco al protocollo Onion.

2. Applicazioni (la parte software)

   I programmi su cui si basa Tor, oppure i webserver hostati sugli hidden service, o anche i programmi usati dagli utenti per navigare su Tor.

3. Dispositivi (la parte hardware)

Quindi tutto è da tenere aggiornato.

Un articolo del 9 Luglio del 2021 discuteva del fatto che circa il 75% degli utenti Tor usava una versione vecchia (la 2.3) e che si presume sia indifesa alle possibilità dell'NSA. [14]

In pratica si sta ipotizzando che grazie ai finanziamenti ricevuti [15] e al fatto che i servizi online tendono a riutilizzare gli stessi numeri primi (o gruppi di numeri primi) a 1024 bit per la crittografia, come si legge in questo dettagliato paper [16], allora l'NSA risulterebbe tranquillamente capace di decriptare moltissime comunicazioni con chiavi RSA/DH a 1024 bit.

Alcune citazioni:

"Breaking a single, common 1024-bit prime would allow NSA to passively decrypt connections to two-thirds of VPNs and a quarter of all SSH servers globally. Breaking a second 1024-bit prime would allow passive eavesdropping on connections to nearly 20% of the top million HTTPS websites. In other words, a one-time investment in massive computation would make it possible to eavesdrop on trillions of encrypted connections."

https://weakdh.org/

"The Logjam attack allows a man-in-the-middle attacker to downgrade vulnerable TLS connections to 512-bit."

https://weakdh.org/

In sostanza Tor v2 che era a rischio, ora è deprecato.

Tor v3 invece ha invece sostituito i vecchi algoritmi di criptazione SHA1/DH/RSA1024 con SHA3/ed25519/curve25519 nel 9 gennaio 2018. [17] [18] [19] [20, linea 132]

In pratica se fino al 2013 era andato tutto bene, da quell'anno al 2018 Tor era (probabilmente) compromesso.

6.6 Osservazioni

Nonostante nelle istruzioni ci sia scritto di scaricare e configurare Apache Ant, in realtà non ce ne sarà bisogno in quanto non viene utilizzato.

Il metodo getVersionName() nel build.gradle funziona solo se la cartella .git è rimasta nel progetto.

7 Onion Routing

Prima di arrivare a parlare dell'integrazione di Tor con Android, risulta sicuramente utile ed interessante sapere come funziona e perché vengono usati quei programmi nella compilazione del programma e quindi contestualizzare il suo utilizzo.

7.1 Da Tor client a clearnet

Adesso prendiamo l'esempio di un client cinese che vuole accedere a YouTube (che in Cina è bloccato).

Uno dei Tor client più utilizzati è Tor Browser 6. Appena avviato, il programma, lancerà in background Tor che comincerà a costruire i circuiti.

All'utente invece apparirà solo la schermata della home con la casella di testo di DuckDuckGo. Per raggiungere YouTube basterà fare una ricerca; questa sarà reindirizzata al server proxy locale creato dal programma Tor.

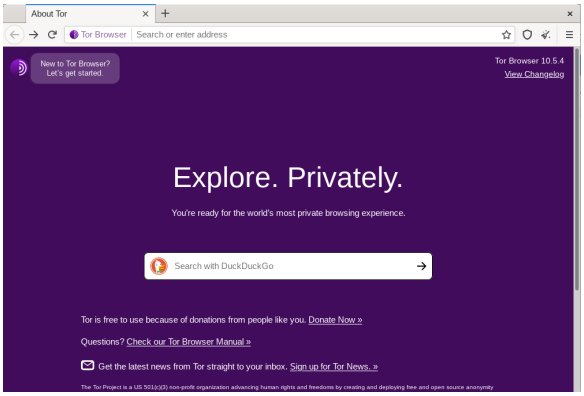


Figura 6: Screenshot della finestra di Tor Browser

7.1.1 Primo avvio di Tor

Al primo avvio di Tor il software ha necessità di sapere quali sono i nodi della rete attivi in quel momento. Per farlo invia una richiesta alle **directory authority**.

Una directory authority è un relay che si occupa di mantenere un elenco di relay attualmente in esecuzione e pubblica periodicamente un consenso insieme alle altre directory authority.

Il consenso, o **consensus**, è un documento compilato e votato dalle directory authority una volta ogni ora e assicura che tutti i client abbiano le stesse informazioni sui relè che compongono la rete Tor.

Tor viene fornito con un elenco integrato (hardcoded) di directory authority.

Nel file **/tor/src/app/config/auth_dirs.inc** è possibile visualizzare la lista di tutte le directory authority disponibili.

Le prime righe del file auth_dirs.inc:

```
// directory authority 1
"moria1 orport=9101 " // nome e porta
    "v3ident=D586D18309DED4CD6D57C18FDB97EFA96D330566 "   //
       fingerprint
    "128.31.0.39:9131 9695 DFC3 5FFE B861 329B 9F1A B04C 4639
       7020 CE31", // IP
// directory authority 2
"tor26 orport=443 " // nome e porta
    "v3ident=14C131DFC5C6F93646BE72FA1401C02A8DF2E8B4 "
    "ipv6=[2001:858:2:2:aabb:0:563b:1526]:443 "
     "86.59.21.38:80 847B 1F85 0344 D787 6491 A548 92F9 0493 4E4E
        B85D",
// directory authority 3
"dizum orport=443 "
  ...
```

E' possibile trovare una lista di tutte le directory authority anche nel sito ufficiale di Tor [21].

Quindi l'unico modo per indurre gli utenti ad utilizzare una falsa rete Tor è fornire loro una versione appositamente modificata del software, per questo diventa importante controllare che il pacchetto scaricato non sia stato modificato. Ad esempio, per eseguire questo controllo, si possono usare le chiavi GPG [22].

Una lista di tutti i relay attualmente disponibili può essere trovata qui a questo link [23].

7.2 Costruzione del primo circuito

Il protocollo base prevede la scelta di 3 relay in modo semi-casuale dalla lista di nodi ricevuti. Vengono scelti:

- Un Guard node (o un Bridge node)
- Un Middle node
- Un Exit node

Questi sono chiamati anche Onion Routers (OR), o **relay**. Con ognuno di essi viene creata una sessione sicura (es. TLS, quindi a chiave simmetrica) lungo la catena attraverso una serie di passaggi.

Il nostro client che in figura 7 appare con il nome di "Alice", in gergo tecnico è anche detto **Onion Proxy (OP)**. Esso costruisce il circuito in modo incrementale, negoziando una chiave simmetrica con ciascun OR sul circuito, un nodo alla volta [24].

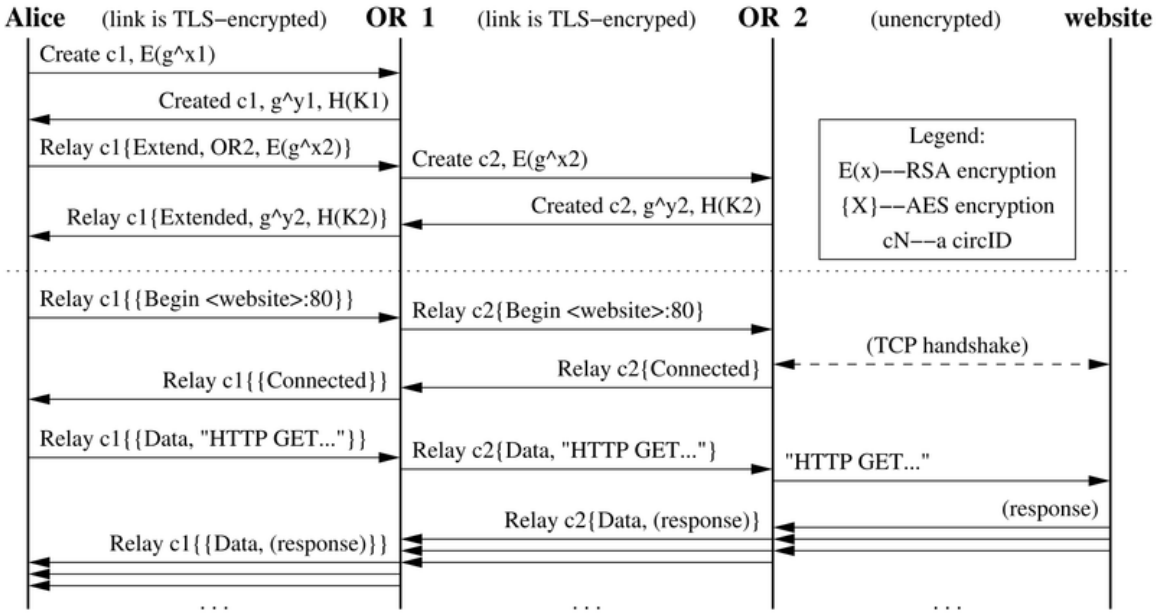


Figura 7: Immagine esempio che mostra la creazione di due sessioni sicure e lo scambio di dati con un sito web. Fonte: Researchgate [25]

Gli Onion Proxy e gli Onion Router sono anche detti **nodi**, essi comunicano tramite le celle (che ricordano un po' la logica di funzionamento dei pacchetti nelle telecomunicazioni) che possono essere di due tipi:

- Celle di controllo
  Queste celle si occupano della creazione dei circuiti e delle sessioni.
- Celle di Relay
  Queste celle si occupano dello scambio di dati e degli errori di connessione.

Un esempio della composizione dei tipi di cella in figura 8.
Tutti le celle sono di grandezza fissa a 512 byte, questo serve ad evitare un attacco che discrimina in base alla lunghezza di ogni cella.

1. Alice (l'OP) invia una cella di controllo con comando CREATE al primo nodo (OR1) del percorso scelto.
   - Il comando CREATE va nella zona dell'header della cella chiamata CMD.
   - Viene scelto un nuovo circID non attualmente utilizzato sulla connessione da Alice a OR1.
   - Il payload (DATA) della cella di creazione contiene la prima metà dello scambio Diffie-Hellman (g^{x1}), crittografata con la chiave RSA pubblica (detta Onion key) dell'OR1.
2. OR1 risponde con una cella creazione contenente la seconda metà dello scambio Diffie-Hellman g^{y1} insieme ad un hash (H) della chiave negoziata/segreta/di sessione (K1), ovvero:

$$HK1 = g^{x1y1}$$

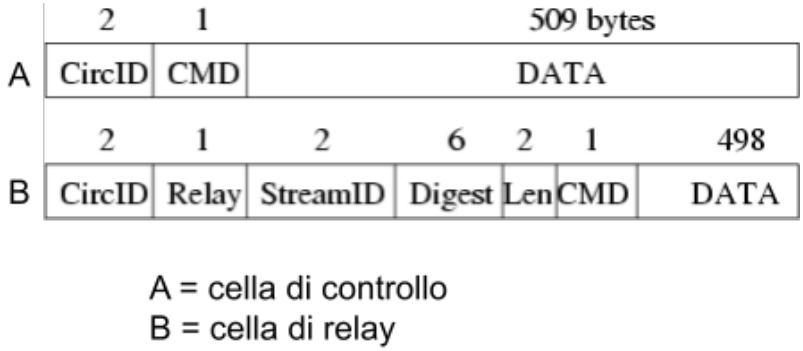


Figura 8: I due tipi di celle. Fonte: Researchgate [26]

Una volta che il circuito è stato stabilito, Alice e OR1 possono scambiarsi celle di relay crittografate con la chiave negoziata (K1).

In realtà, la chiave negoziata viene utilizzata per derivare due chiavi simmetriche: una per ogni direzione.

La comunicazione tra due relay Tor, o tra un client e un relay, usa dei protocolli TLS o SSLv3.

Nello specifico sono queste 3 [27, 198-230] le configurazioni di base:

- TLS_DHE_RSA_WITH_AES_256_CBC_SHA
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA
- SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA

Possono essere supportati metodi di cifratura migliori, se disponibili.

Per estendere ulteriormente il circuito:

1. Alice invia una cella relay di estensione (RELAY EXTEND) a OR1, specificando l'indirizzo di OR2 e la prima parte dello scambio DH (g^{x2}).
    - la prima parte dello scambio DH g^{x2} è criptata con l'RSA pubblica di OR2. Questo impedisce a OR1 di leggerne il contenuto.
    - La cella di estensione è criptata con la chiave di sessione K1 ottenuta nella fase di creazione con OR1.
2. OR1 copia g^{x2} criptata in una cella di creazione e la invia a OR2 per estendere il circuito.
    - OR1 sceglie un nuovo circID non attualmente utilizzato sulla connessione tra lui e OR2.
    - Alice non ha mai bisogno di conoscere questo circID.
        - Solo OR1 associa il circID con Alice sulla sua connessione con Alice al circID con OR2 sulla sua connessione con OR2 (simile ad un port forwarding)
3. OR2 risponde con una cella di creazione contenente la seconda parte dello scambio DH (g^{y2}) e l'hash della chiave K2. OR1 incapsula tutto su una cella di estensione (criptata con la chiave K1) e la invia ad Alice.

4. Ora Alice e OR2 condividono una chiave segreta, o di sessione:

$$K2 = g^{x2y2}$$

   - Alice può controllare la riuscita della creazione della chiave confrontando l'hash inviato da OR2.

$$HK2$$

7.3 Qualche parola sul funzionamento del circuito

7.3.1 Apertura di una connessione TCP con la clearnet

Quando Alice vuole aprire un TCP stream:

1. Alice apre uno stream inviando una cella RELAY BEGIN all'exit relay.
   - Viene usato uno streamID random.
   - Ogni cella viene criptata:
     (a) Prima con la chiave (di sessione) dell'Exit node
     (b) Poi con la chiave (di sessione) del Middle node
     (c) Infine con la chiave (di sessione) del Guard node
2. L'exit relay che riceve il pacchetto provvede ad effettuare l'handshake TCP con l'host destinatario (YouTube) alla porta richiesta.
3. Successivamente l'exit node risponde con una cella RELAY CONNECTED facendole percorrere il percorso inverso nel circuito.
   - La risposta viene criptata, un nodo alla volta, seguendo la catena nel verso opposto.
4. A questo punto Alice potrà inviare dati dall'applicazione all'exit node attraverso delle celle RELAY DATA. [28, pagina 22]

La chiusura di uno stream TCP viene effettuata in maniera analoga con la sola differenza che al posto di una cella RELAY CONNECT, o RELAY CONNECTED viene inviata una cella RELAY END, o una RELAY ENDED.

Uno stream TCP si potrebbe chiudere anche in maniera inaspettata.

L'exit node in tal caso invierà a ritroso nel circuito una cella RELAY TEARDOWN che segnalerà ad Alice la chiusura inaspettata dello stream TCP. [28, pagina 23]

7.3.2 Ogni quanto il circuito viene creato?

In linea generale ogni minuto viene calcolato in background un nuovo circuito.

"**4.2 Circuits and streams** Onion Routing originally built one circuit for each TCP stream. Because building a circuit can take several tenths of a second (due to public-key cryptography and network latency), this design imposed high costs on applications like web browsing that open many TCP streams. In Tor, each

circuit can be shared by many TCP streams. To avoid delays, users construct circuits preemptively. To limit linkability among their streams, users' OPs build a new circuit periodically if the previous ones have been used, and expire old used circuits that no longer have any open streams. OPs consider rotating to a new circuit once a minute: thus even heavy users spend negligible time building circuits, but a limited number of requests can be linked to each other through a given exit node. Also, because circuits are built in the background, OPs can recover from failed circuit creation without harming user experience."

Tor: The Second-Generation Onion Router [24, pagina 5]

7.3.3 Ma alla Cina non basta bloccare quindi l'accesso alle directory authority?

La risposta corta è che per risolvere questo problema sono stati inventati i bridge che non vengono segnalati dalle directory authority ma vanno configurati manualmente. [29]

Nel momento in cui le directory authority vengono bloccate, o se ne perde la fiducia, vengono utilizzati i bridge.

Nel 2009 in Cina, dopo l'installazione del Great Firewall, le richieste alle directory authority sono state bloccate e sono aumentati gli accessi tramite bridges [30]:

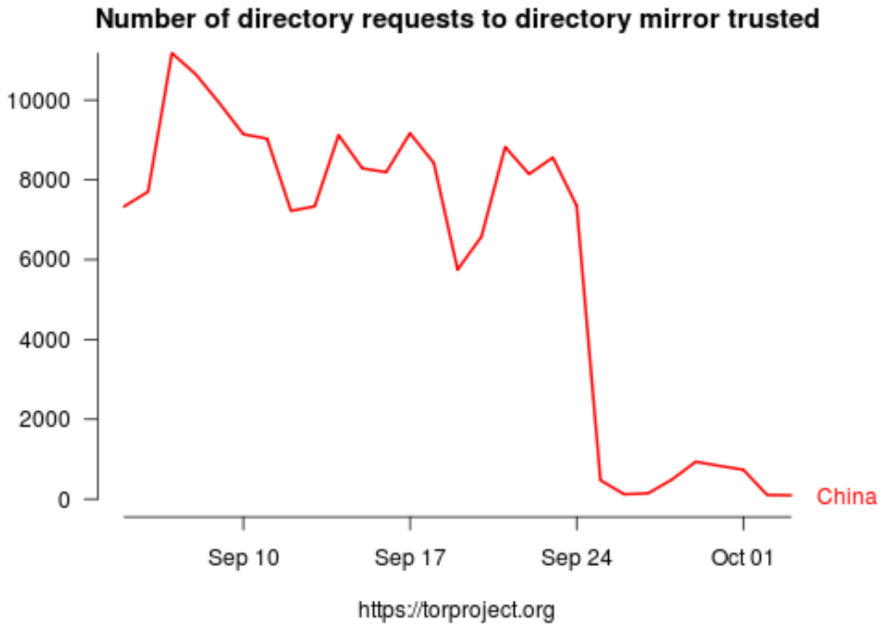


Figura 9: Un grafico che mostra il numero di richieste fatte alle directory authority nel mese di Settembre

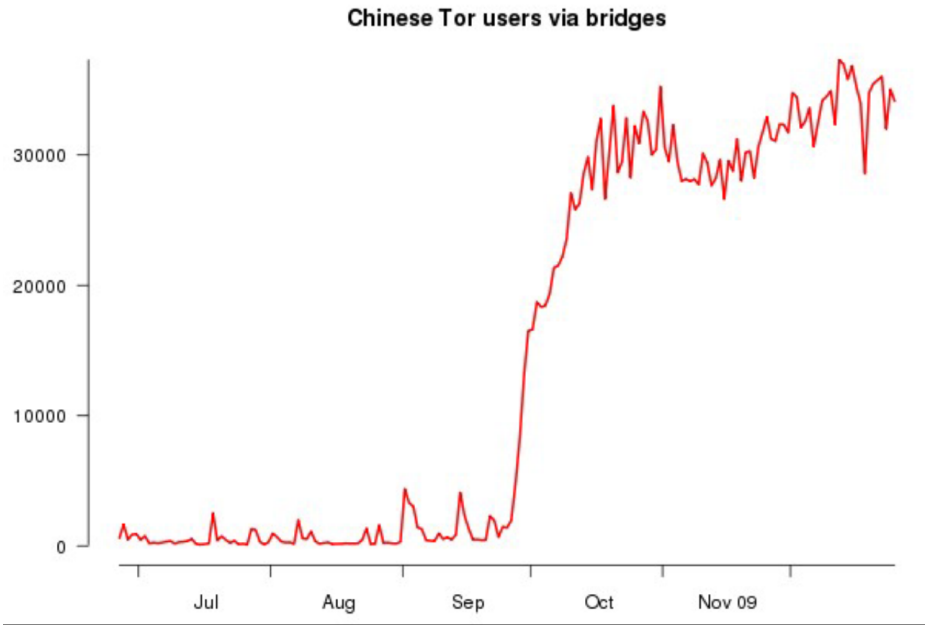


Figura 10: Un grafico che mostra il numero di collegamenti alla rete Tor tramite bridges a fine 2009

Dal 2010, in Cina, quasi tutti i bridge presenti su TorProject.org sono stati bloccati. Sono ancora funzionanti i bridge pubblicati in modo indipendente che usano i protocolli Obfs4 e meek. [31]

7.4 Da Tor client ad un Onion Service

Tor consente ad ogni client e relay di offrire **Onion Services**. I servizi Onion, precedentemente noti come **Hidden Services** (HS), sono servizi (come un server web, un server SSH, ecc...) accessibili solo tramite la rete Tor.

Ovvero, Tor permette di offrire un servizio hostato su un server senza rivelare l'indirizzo IP pubblico del server agli utenti.

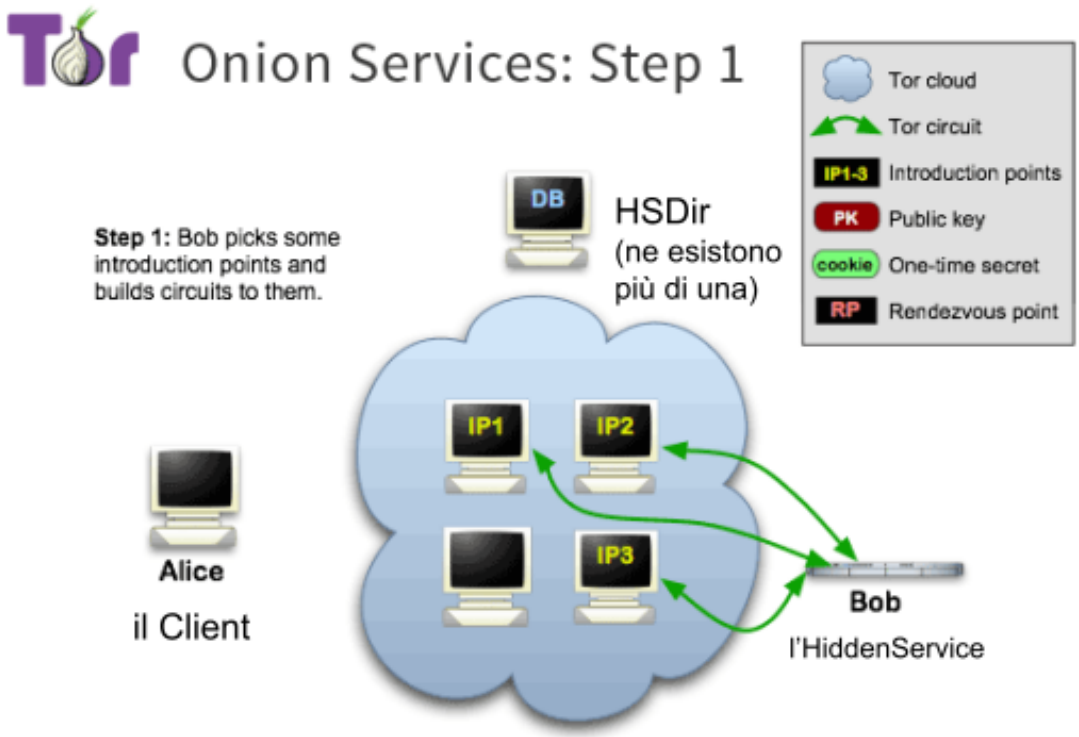
Per funzionare, Tor deve essere installato da entrambe le parti.

7.4.1 Step 1: Inizializzazione degli Introduction Points

Prima che qualsiasi OP possa collegarsi ad un HS, questo deve far sapere alla rete Tor della sua esistenza.

Per farlo deve richiedere degli Introduction Points.

Figura 11: Inizializzazione degli Introduction Point



https://2019.www.torproject.org/docs/onion-services.html.en

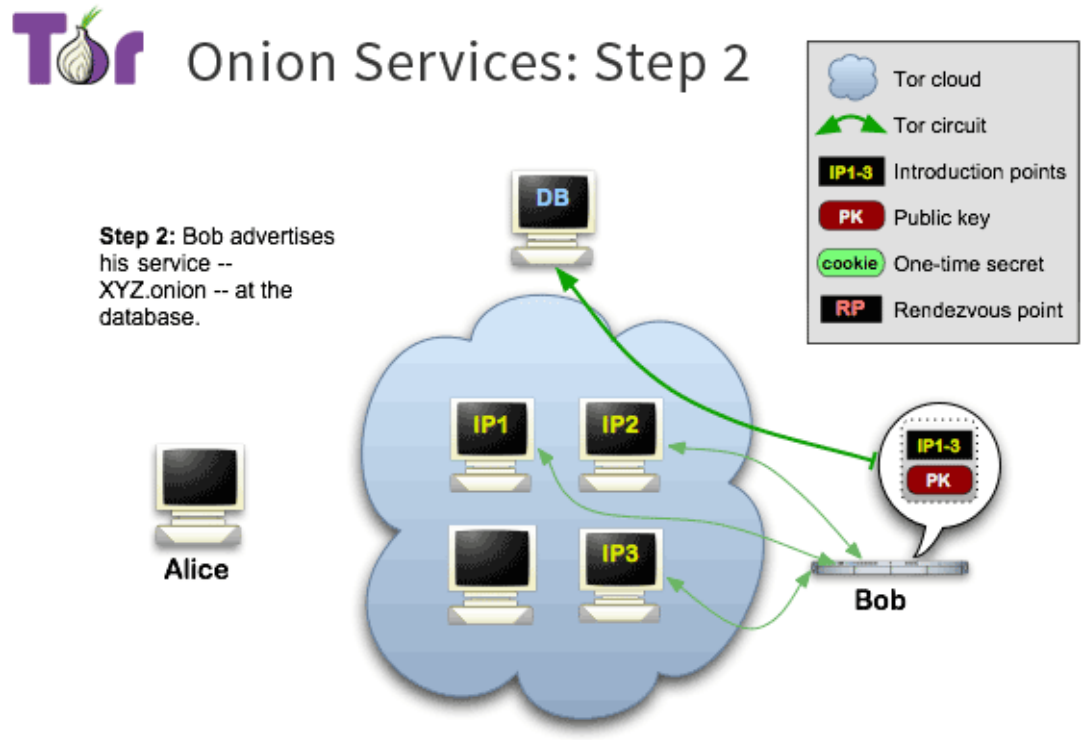
L'HS in figura 11, chiamato Bob, genera una coppia di chiavi a lungo termine per identificare il suo servizio (la chiave pubblica sarà il nome del suo dominio .onion).

Un esempio di dominio Onion: http://www.hdnwikikj6nampkg.onion/

HS sceglie 3 Introduction Points e costruisce un circuito Tor verso ognuno di quei nodi.

Gli Introduction Points sanno che nel momento in cui qualche OP richiede l'accesso all'HS dovranno indirizzare le richieste a quest'ultimo utilizzando quei circuiti. [20, linee 388-391]

7.4.2 Step 2: Creazione, invio e posizionamento dei descrittori



https://2019.www.torproject.org/docs/onion-services.html.en

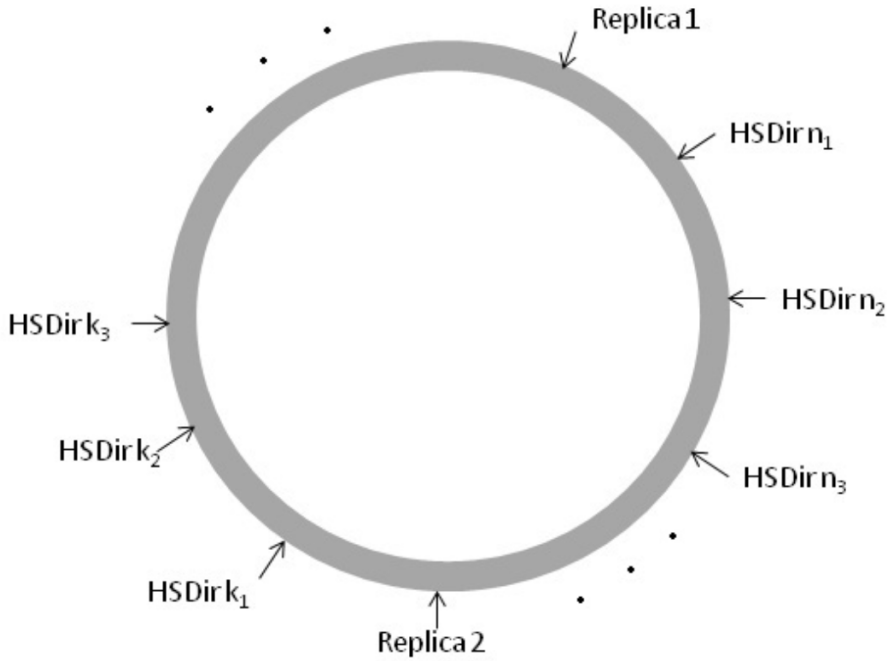
Figura 12: Caricamento dei descriptors sugli Hidden Service Directory

Una volta selezionati i punti di introduzione, l'HS crea uno, o una serie di documenti (dipende se l'HS offre uno, o più servizi) chiamati **Hidden Service Descriptors** (descrittori di servizio nascosto) o più semplicemente **descriptors** che firma con la sua chiave e che carica su un insieme di nodi chiamati **Hidden Service Directory**, o HSDir (sull'immagine 12 questo gruppo di nodi è identificato come DB).

Questi documenti elencano gli attuali punti di introduzione all'HS e descrivono come entrare in contatto con lui (es. quale configurazione di cifratura usare per la comunicazione). [20, linee 393-397]

Periodicamente, i descriptors vengono archiviati in diversi HSDir per impedire ad un HSDir, oppure ad un piccolo insieme di HSDir, di diventare un buon obiettivo per attacchi DoS con lo scopo di bloccare l'HS. [20, linee 480]

Per calcolare su quali HSDir un descriptor deve essere caricato viene usato un time period e un **Shared Random Values** (SRV).



Conceptual view of the hash ring.

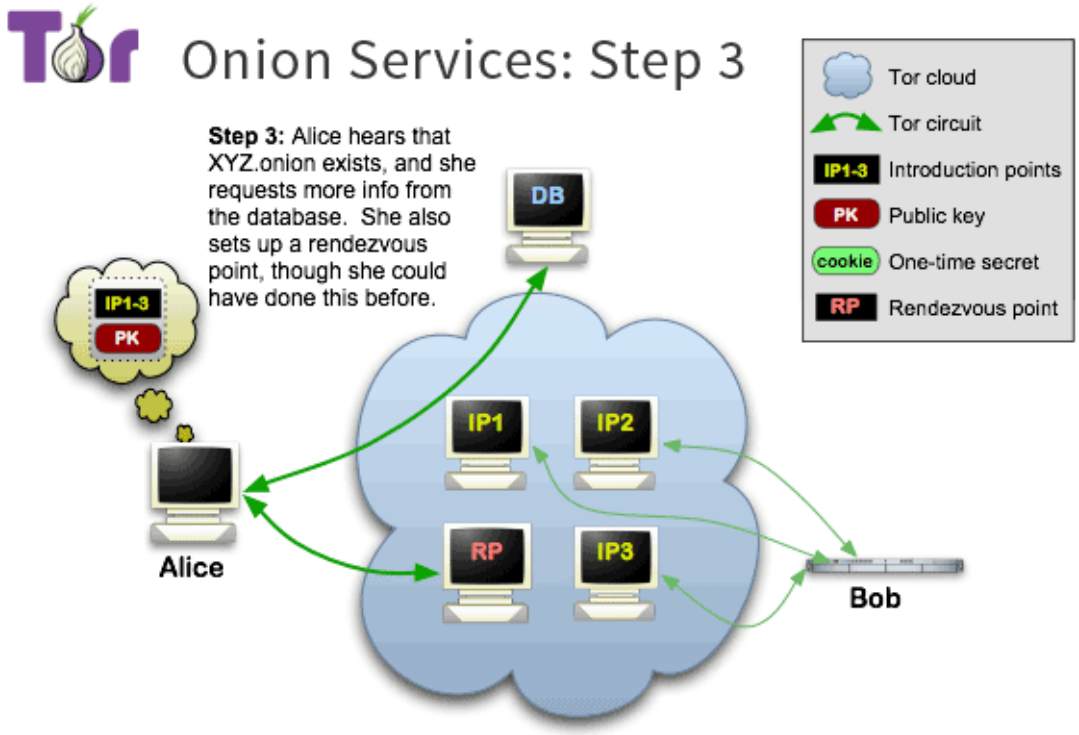
https://www.benthamsgaze.org/wp-content/uploads/2015/11/sucu-torscaling.pdf

Figura 13: Rappresentazione concettuale dell'anello rappresentante la posizione degli HSDir

Ogni 24 ore le Directory Authority generano in modo collaborativo un SRV. Questo valore, combinato con la chiave pubblicha di ogni HSDir, determina la posizione di ciascun HSDir nell'hash ring per la distribuzione dei descriptors realizzati in quel periodo, mentre, la posizione dei descriptors è basata sulla chiave che è stata utilizzata per la firma.

7.4.3 Step 3: Inizializzazione del punto di ritrovo e scaricamento del descrittore

Quando Alice vuole connettersi ad un HS prima sceglie un nodo Tor a caso per essere il suo **Rendezvous Point** e costruisce un circuito con quel relay. Poi se il client non ha i descriptors aggiornati per il servizio, contatta un appropriato HSDir e lo richiede (descriptor lookup).



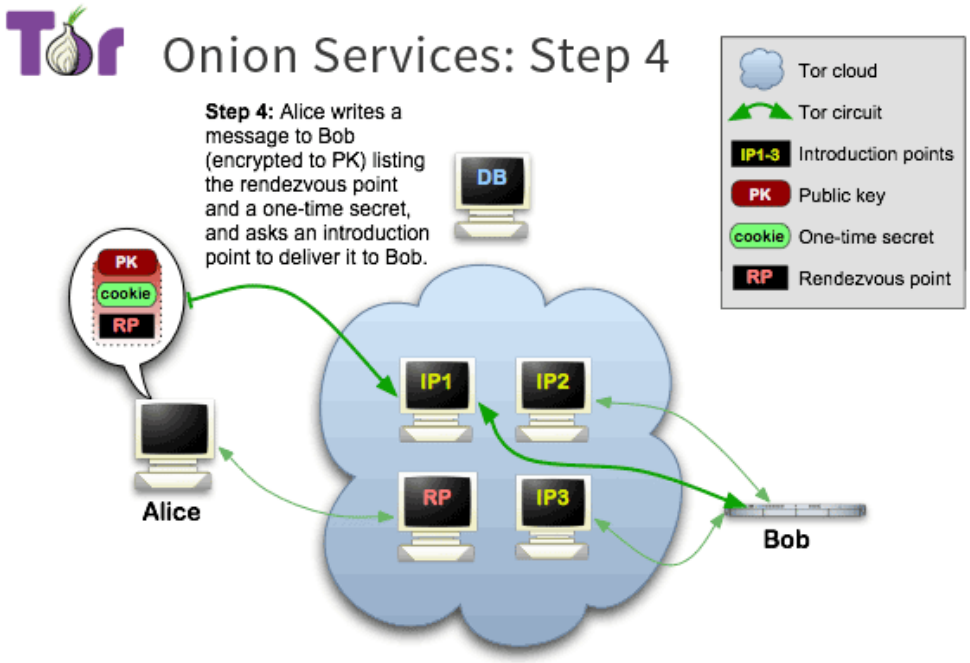
https://2019.www.torproject.org/docs/onion-services.html.en

Figura 14: Rappresentazione della creazione del circuito con il Rendezvous Point e richiesta dei descriptors aggiornati

7.4.4 Step 4: Connessione al punto di introduzione e richiesta di introduzione

Alice costruisce un circuito anonimo ad uno degli introduction points elencati nel descrittore del servizio e gli invia una **Introduction Request** da passare all'HS.

Questa richiesta di introduzione include il rendezvous point ed una prima parte dell'handshake per criptare la connessione (**oltre al circuito criptato creato con il rendezvous point**). Serve ad impedire al rendezvous point di leggere il contenuto della connessione (risulta comunque uno degli anelli deboli della catena).

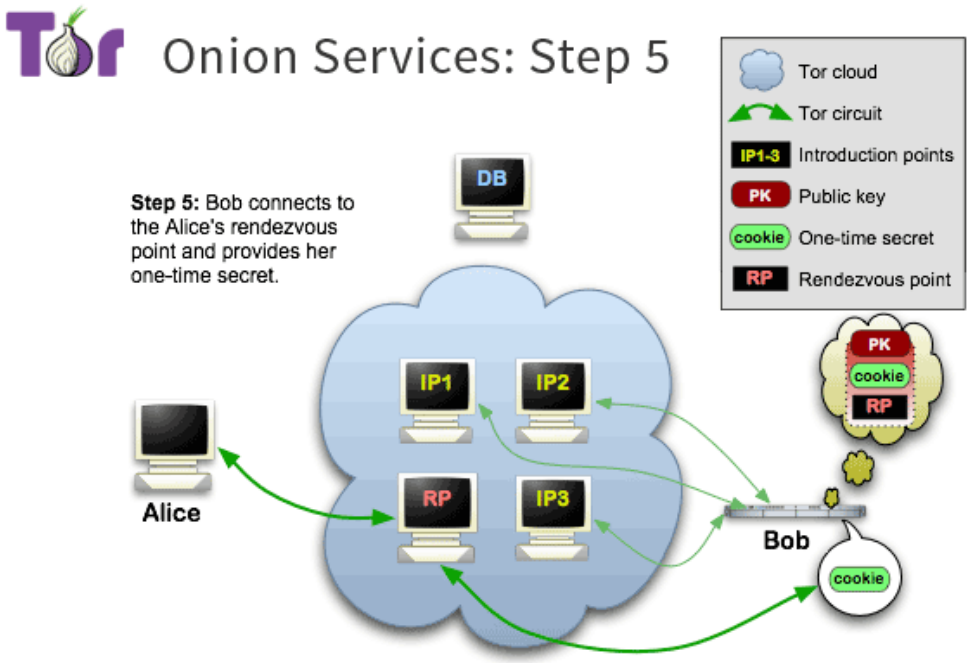


https://2019.www.torproject.org/docs/onion-services.html.en

Figura 15: Alice invia i dati per la connessione all'introduction point

7.4.5 Step 5: Completamento della connessione

Dopo aver ricevuto la introduction request, l'HS può decidere se accettarla, se lo fa costruisce un circuito anonimo fino al rendezvous point e completa la seconda parte per criptare la connessione.



https://2019.www.torproject.org/docs/onion-services.html.en

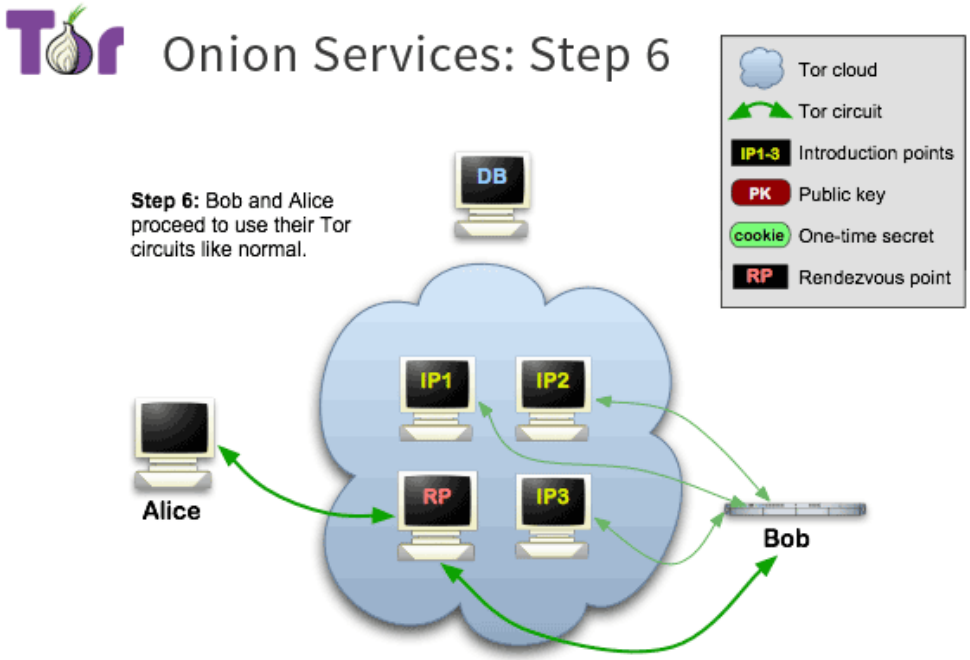
Figura 16: Connessione al rendezvous point

7.4.6 Step 6: Connessione completata e osservazioni

La finalizzazione della connessione criptata fornisce ad Alice e a Bob un insieme condiviso di chiavi che vengono usate per scambio ed autenticazione del traffico come nel normale protocollo di crittografia di Tor.

Questo sistema ha diverse caratteristiche e diverse funzionalità:

- L'HS può decidere quali connessioni accettare e quali no (riceve le richieste agli introduction point ma risponde sul rendezvous point)
- L'HS è difeso da attacchi DoS
- L'HS non è legato ad un singolo OR, in questo modo può mantenere la sua identità ma cambiare gli introduction points
- In linea teorica non è possibile per un attaccante far finta di essere un altro HiddenService



https://2019.www.torproject.org/docs/onion-services.html.en

Figura 17: Connessione completata tra HS e OP

8 Funzionamento dell'applicazione esempio

Ricapitolando, il progetto ha 2 sottoprogetti:

- /tor-android/sampletorapp/
- /tor-android/tor-android-binary/

sampletorapp contiene il codice che dimostra come si potrebbe iniziare a costruire l'applicazione.
Questo progetto esempio fa capire molte cose sia di alcune funzioni di Android, sia di come è stato progettato il software di Tor.

sampletorapp ha due classi:

- GenericWebViewClient.java
- MainActivity.java

8.1 MainActivity

Da questa classe, dopo le prime righe, subito dopo la configurazione della grafica, si viene a scoprire che la libreria Tor comunica tramite messaggi di broadcast. Il codice relativo è questo:

```
registerReceiver(new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        // il Toast e' un semplice popup a schermo
        Toast.makeText(context,
                intent.getStringExtra(TorService.EXTRA_STATUS), //
                    setta il contenuto del popup
                Toast.LENGTH_SHORT // setta la durata del popup
        ).show();
    }
}, new IntentFilter(TorService.ACTION_STATUS));
```

Questo codice crea un **BroacastReceiver** e lo setta in ascolto di una singola azione (filtro) definita da **TorService.ACTION_STATUS**.

8.1.1 Messaggi di Broadcast

I broadcast sono messaggi utilizzati quando si desidera inviare o ricevere dati tra applicazioni diverse. Mentre se la comunicazione è limitata alla singola applicazione è buona pratica utilizzare altri meccanismi di trasmissione.

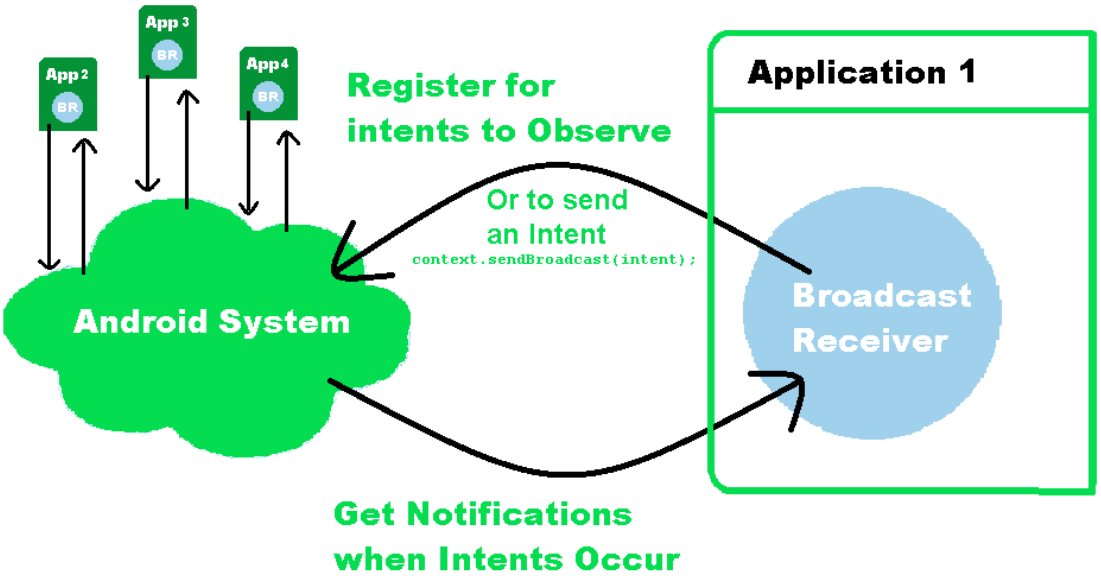


Figura 18: Schema del funzionamento dei Broadcast su Android

- Il BroadcastReceiver descrive un'azione da eseguire nel caso il filtro (IntentFilter) venga attivato.
- Un BroadcastReceiver può essere registrato su più filtri.

Si possono comunque creare azioni personalizzate. Una lista completa di tutte le possibili azioni (IntentsFilter) può essere trovata in un file di testo scaricato con l'installazione di Android Studio a questo percorso:

```
C:\Users\[username]\AppData\Local\Android\Sdk\platforms\[android-
    version]\data\broadcast_actions.txt
```

Per ovvie ragioni, i broadcast non devono mai contenere informazioni sensibili, infatti l'utilizzo che ne fa la libreria Tor è quello di comunicare l'avvenuta connessione con la rete Onion.

Tuttavia, si possono trasmettere informazioni anche localmente utilizzando la classe **LocalBroadcastManager**, che fa parte della libreria di supporto di Android (com.android.support:support-v4:23.4.0).

Di seguito sono riportati alcuni dei vantaggi del LocalBroadcastManager:

- I dati di trasmissione rimarranno locali e quindi non lasceranno la tua app, quindi è possibile inviare dati privati.
- Non è possibile per altre applicazioni inviare questi messaggi alla tua app, quindi non c'è pericolo di avere buchi di sicurezza sfruttabili.
- E' più efficiente dell'invio di un broadcast globale attraverso il sistema.
- Nessun sovraccarico della trasmissione a livello di sistema.

8.1.2 Intents

I Broadcast usano gli Intents, questi, oltre a contenere dati extra per essere inviati tramite sendBroadcast, possono essere usati anche per:

- Far partire delle Activity (il metodo usato per far partire un'applicazione Android).
- **Avviare un servizio, o associare un servizio per la comunicazione con un servizio in background**.

Proprio la seconda funzionalità degli Intents sarà utilizzata per avviare TorService:

```
context.startService(new Intent(this, TorService.class));
```

8.1.3 Services

Appena aperto il codice di TorService la prima cosa che deve saltare all'occhio è l'estensione alla classe astratta **Service**.

```
public class TorService extends Service { ... }
```

Un servizio è un componente dell'applicazione in grado di eseguire operazioni a lunga durata in background.

Una volta avviato, un servizio potrebbe continuare a funzionare anche dopo che l'utente è passato ad un'altra applicazione. Inoltre, un altro componente può associarsi ad un servizio già esistente per interagire con esso ed eseguire delle comunicazioni interprocesso (IPC).

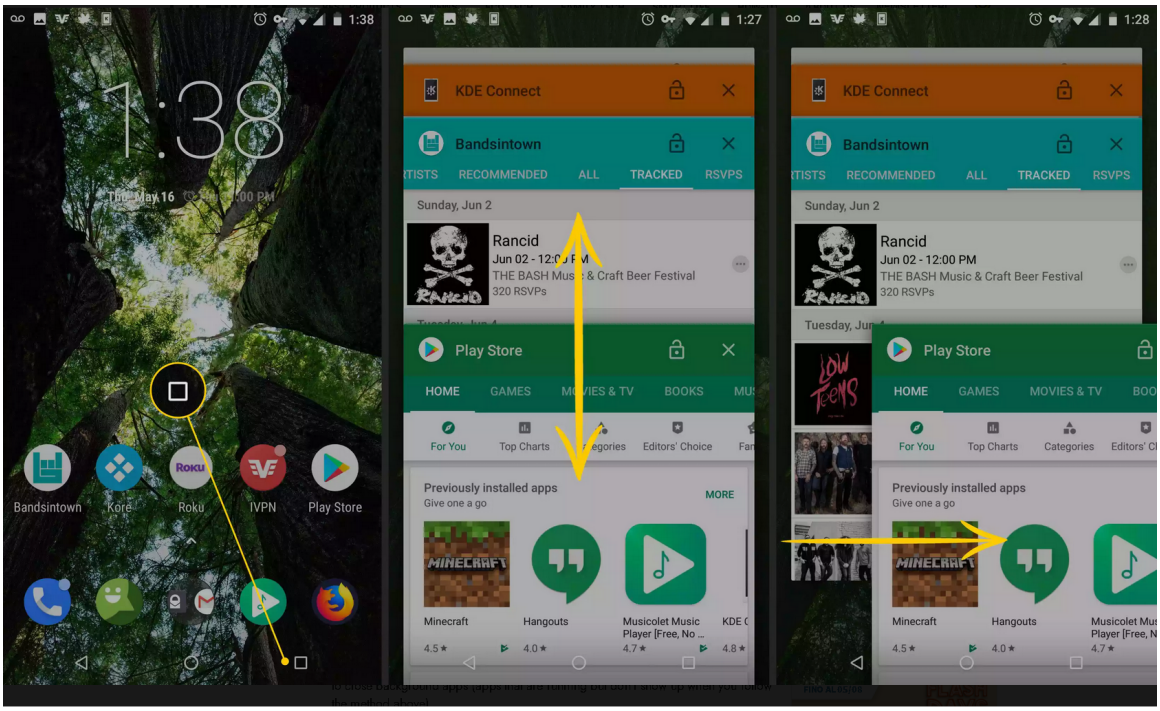
In senso lato si può intendere come meccanismo di comunicazione interprocesso anche la semplice clipboard che consente ad un utente di copiare ed incollare informazioni da una finestra ad un'altra, o l'uso di file, che un processo scrive ed un altro legge; tuttavia, si parla di IPC in senso stretto solo per quei meccanismi che possono essere usati dal software senza intervento manuale umano e che non memorizzano i dati su memorie di massa.

Ad esempio, un servizio può gestire transazioni di rete, riprodurre musica, eseguire I/O di file o interagire con un provider di contenuti, il tutto in background.

8.1.4 Differenza tra servizi e thread e la loro combinazione

Un servizio è un componente che può essere eseguito in background, ma che agisce nel main thread e ha la particolarità di funzionare anche quando l'utente non sta interagendo con l'applicazione che ha lanciato il servizio. Se invece si vogliono eseguire operazioni al di fuori del thread principale, ma solo mentre l'utente sta interagendo con l'applicazione, si crea invece un nuovo thread nel contesto dell'applicazione.

Questi due componenti possono essere usati insieme per creare un processo che può lavorare quando l'utente non sta interagendo con l'applicazione (startService()) ma anche staccato dal thread principale (new Thread). L'unico modo per fermare questo thread è chiudere il servizio, oppure terminare l'applicazione tramite il task manager.



https://www.lifewire.com/close-apps-on-android-4164116

Figura 19: Esempio di chiusura di un'applicazione tramite il taskmanager su Android

8.1.5 TorService

Questa classe si trova nel sottoprogetto:

/tor-android/tor-android-binary/

e si occupa di avviare il software Tor. Andiamo a guardare subito i 2 metodi più interessanti: l'override di onCreate() e il metodo startTorServiceThread().

OnCreate() va a comunicare tramite il metodo broadcastStatus usando un LocalBroadcastManager che il servizio sta partendo:

```
@Override
public void onCreate() {
    super.onCreate();
    broadcastStatus(this, STATUS_STARTING);
    startTorServiceThread();
}
```

startTorServiceThread è invece il metodo centrale dove viene creato il comando settando i vari parametri:

- Vengono configurate le porte del proxy e del tunnel HTTP

- Vengono trovati e passati i percorsi di alcuni file
- Vengono settate alcune flags

Ad esempio si può creare, o settare un file chiamato torrc che è un file di testo che contiene le istruzioni di configurazione su come dovrebbe comportarsi Tor. La configurazione predefinita dovrebbe funzionare bene per la maggior parte degli utenti Tor.

```
// start Tor in a Thread with the minimum required config
private void startTorServiceThread() {
    final Context context = this.getApplicationContext();
    new Thread("tor") {
        @Override
        public void run() {
```

Il metodo cha fa partire Tor crea un nuovo thread dentro il servizio TorService.

```
            try {
                String socksPort = "auto";
                if (isPortAvailable(9050))
                    socksPort = Integer.toString(9050);
                String httpTunnelPort = "auto";
                if (isPortAvailable(8118))
                    httpTunnelPort = Integer.toString(8118);
```

Vengono definite le variabili delle porte che verranno passate a Tor per creare il proxy locale.

```
    createTorConfiguration();
    ArrayList<String> lines = new ArrayList<>(Arrays.asList("tor",
        "--verify-config", // comando che dice a Tor di controllare questa
            stringa di modo che in caso di errore il problema possa venire
            loggato
        "--RunAsDaemon", "0",
        "-f", getTorrc(context).getAbsolutePath(), // setta il path del file
            torrc
        "--defaults-torrc", getDefaultsTorrc(context).getAbsolutePath(),
        "--ignore-missing-torrc",
        "--SyslogIdentityTag", TAG,
        "--CacheDirectory", new File(getCacheDir(), TAG).getAbsolutePath(),
        "--DataDirectory", getAppTorServiceDataDir(context).getAbsolutePath()
            ,
        "--ControlSocket", getControlSocket(context).getAbsolutePath(),
        "--CookieAuthentication", "0",
        "--SOCKSPort", socksPort, // setta la porta per il proxy
        "--HTTPTunnelPort", httpTunnelPort, // setta la porta da usare su cui
             Tor permettera' l'utilizzo del protocollo HTTP

        // can be moved to ControlPort messages
        "--LogMessageDomains", "1",
        "--TruncateLogFile", "1"
    ));
```

L'array "lines" contiene le stringhe che descrivono il comando per avviare Tor e i suoi parametri.

```
    String[] verifyLines = lines.toArray(new String[0]);
    if (!mainConfigurationSetCommandLine(verifyLines)) // verifyLines
        throw new IllegalArgumentException("Setting command line failed: " +
            Arrays.toString(verifyLines));
    int result = runMain();  // run verify
    if (result != 0)
        throw new IllegalArgumentException("Bad command flags: " + Arrays.
            toString(verifyLines));
```

Viene creata una copia di "lines" chiamato "verifyLines" che verrà passato ad un **metodo nativo** per settare il comando da lanciare a terminale (il terminale di Android è visibile tramite adb).

mainConfigurationSetCommandLine(verifyLines) e runMain() compiono una serie di controlli sulla configurazione del comando ma non è qui che Tor viene avviato.

```
        controlPortThreadStarted = new CountDownLatch(1);
        controlPortThread.start();
        controlPortThreadStarted.await();

        String[] runLines = new String[lines.size() - 1];
        runLines[0] = "tor";
        for (int i = 2; i < lines.size(); i++) // parte dall'indice 2 per
            rimuovere la '--verify-config' flag
            runLines[i - 1] = lines.get(i);
        if (!mainConfigurationSetCommandLine(runLines))
            throw new IllegalArgumentException("Setting command line failed:
                " + Arrays.toString(runLines));
        if (!mainConfigurationSetupControlSocket())
            throw new IllegalStateException("Setting up ControlPort failed!")
                ;
        if (runMain() != 0) // con la flag rimossa Tor e' abilitato a partire
            throw new IllegalStateException("Tor could not start!");

    } catch (IllegalStateException | IllegalArgumentException |
        InterruptedException e) {
        e.printStackTrace();
        broadcastError(context, e);
        broadcastStatus(context, STATUS_STOPPING);
        TorService.this.stopSelf();
    }
   }
  }.start();
 }
```

Con la '–verify-config' flag rimossa il metodo nativo runMain() avvierà Tor.

8.1.6 Moduli e metodi nativi

Nella classe TorService sono presenti diversi metodi nativi:

- private native String apiGetProviderVersion();
- private native boolean createTorConfiguration();
- private native void mainConfigurationFree();
- private native static FileDescriptor prepareFileDescriptor(String path);
- private native boolean mainConfigurationSetCommandLine(String[] args);
- private native boolean mainConfigurationSetupControlSocket();
- private native int runMain();

Sono riconoscibili grazie alla parola chiave **native** e la loro implementazione risiede in dei moduli scritti in codice nativo, ad esempio, per quanto riguarda Android si tratta di C e C++.

Per essere costruiti questi moduli usano un framework chiamato JNI (Java Native Interface).

La libreria condivisa viene caricata in un blocco statico. Ciò garantisce che sarà pronta quando ce ne sarà bisogno. Il caricamento di questi moduli avviene attraverso questa chiamata:

```
static {
    System.loadLibrary("tor");
}
```

Questa funzione andrà a caricare in memoria il contenuto del file "libtor.so" la cui compilazione è descritta al punto 7.

Per includere nella build questi file invece è necessario aggiungere una impostazione sul build.gradle del sottoprogetto:

```
dependencies {
 implementation fileTree(dir: 'libs', include: ['*.so'])
 ...
}
```

8.2 GenericWebViewClient

La seconda classe è un esempio di come implementare un WebViewClient, questo serve insieme ad una view chiama WebView per mostrare pagine web su un'applicazione android.

La particolarità di questo WebViewClient è che le richieste vengono intercettate e vengono reindirizzate verso Tor.

9 Frontend

Il problema dell'applicazione esempio è che non presenta un esempio di come inviare e ricevere dati da delle RestAPI in un hidden service.

La prima cosa da decidere è quale tipo di porta proxy far utilizzare a Tor.

9.1 Il proxy di Tor

Quando viene avviato Tor sul proprio computer, questo creerà un servizio di proxy locale in cui l'utente potrà indirizzare tutte le applicazioni che supportano la possibilità di collegarsi a questo.

Il proxy di Tor è accessibile, nella configurazione di default, solo dalla macchina dell'utente, è hostato sulla macchina dell'utente, è gratis e per gli utenti più esperti è personalizzabile.

A differenza di Tor, un tipico fornitore proxy:

- Imposta un server proxy su internet e ti consente di utilizzarlo per inoltrare il tuo traffico. Quindi **non è locale**.
- Molti utenti diversi possono collegarsi a questo proxy.
- Il fornitore può addebitare un costo per il suo utilizzo o finanziare i propri costi tramite pubblicità sul server.
- Nella configurazione più semplice, non è necessario installare nulla.

Semplici fornitori proxy offrono semplici soluzioni per proteggere privacy e anonimato, mentre fornitori di proxy a pagamento offrono soluzioni migliori.

9.2 Differenze tra socket HTTP e SOCKS5 nei proxy

"A differenza dei proxy HTTP, che possono lavorare solo con pagine Web HTTP e HTTPS, i proxy Socks5 possono funzionare con qualsiasi traffico. I proxy HTTP sono proxy di alto livello solitamente progettati per un protocollo specifico. Questo significa velocità di connessione migliori, ma non sono flessibili e sicuri come i Socks5.. Quest'ultimi sono proxy a basso livello che possono gestire qualsiasi programma o protocollo e qualsiasi traffico senza limitazioni."

[32]

Non cercherò di più ma sembrano esserci diversi buoni motivi per cui scegliere i proxy Socks5 tra cui il fatto che sono più aggiornati e che sembrano essere più sicuri.

9.3 Splashscreen

Oltre alla piacevole sensazione alla vista di una splashscreen ben fatta, anche e soprattutto un espediente per caricare in background il servizio Tor e tutto quello che potrebbe servire per far funzionare bene l'app.

Quindi si va a modificare l'AndroidManifest.xml per indicare l'attività da avviare al momento dell'avvio dell'app:

```
<activity android:name=".activities.SplashScreenActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

<activity android:name=".activities.MainActivity" />
```

Dentro la classe SplashScreenActivity si registra quindi un BroadcastReceiver che cambia attività tramite un controllo: nel momento in cui TorService manda un Broadcast che contiene la stringa che dice che Tor risulta acceso e dopo aver inizializzato il proxy allora si chiuderà la splashscreen e si passerà all'attività principale.

```
registerReceiver(new BroadcastReceiver() {
  @Override
  public void onReceive(Context context, Intent intent) {
     String status = intent.getStringExtra(TorService.EXTRA_STATUS);
     Toast.makeText(context, status, Toast.LENGTH_SHORT).show();

     if(status.compareTo(TorService.STATUS_ON) == 0) { // controlla che
         TorService confermi l'avvio di Tor
        new Thread(() -> {
           TorProxy.getInstance(); // inizializza il Proxy tramite un
               singleton
           runOnUiThread(() -> { // termina SplashScreenActivity e passa a
               MainActivity
              Intent mainIntent = new Intent(SplashScreenActivity.this,
                  MainActivity.class);
              SplashScreenActivity.this.startActivity(mainIntent);
              SplashScreenActivity.this.finish();
           });
        }).start();
     }
  }
}, new IntentFilter(TorService.ACTION_STATUS)); // il Broadcast e'
    registrato su questa azione

startService(new Intent(this, TorService.class)); // fa partire TorService
```

TorService manda un broadcast globale quindi tutte le applicazioni possono sapere se Tor è stato avviato.

9.4 TorProxy

Per l'implementazione della classe TorProxy ho deciso di usare un Singleton. Per questo proxy è stata configurata solo la porta per il socket di tipo SOCKS5.

```
public class TorProxy {
  private static volatile Proxy INSTANCE;
  public static Proxy getInstance() {
     if (INSTANCE == null) {
        synchronized (TorProxy.class) {
           INSTANCE = new Proxy(Proxy.Type.SOCKS, new InetSocketAddress("
               localhost", 9050));
        }
     }
     return INSTANCE;
  }
}
```

Essendo l'istanza del proxy dichiarata con la parola chiave "volatile", tutte le modifiche fatte alle configurazioni del proxy saranno visibili da tutti i threads che andranno ad utilizzare l'istanza.

Questo implica che se si vuole creare una configurazione personalizzata sarà necessario creare un clone dell'istanza, o un altro proxy.

9.5 Retrofit

Per controllare che Tor stia funzionando correttamente eseguirò 2 chiamate tramite Retrofit per andare ad ottenere dei json contenenti gli IP: uno ottenuto tramite il proxy collegato a Tor, mentre l'altro ottenuto tramite la rete normale, se sono differenti allora Tor sta probabilmente funzionando correttamente.

Prima di fare questo è necessario parlare appunto di Retrofit. Normalmente per creare delle richieste si usano le classi OkHttp e HttpURLConnection ma mentre queste sono implementazioni a basso livello, Retrofit fornisce una libreria costruita sopra OkHttp che facilita l'implementazione delle chiamate ai servizi che si andranno ad utilizzare.

Per lavorare con Retrofit sono fondamentalmente necessarie le seguenti classi:

- Una classe modello utilizzata come modello JSON
- Delle interfacce che definiscono le possibili operazioni HTTP

Retrofit da sola però non è in grado di gestire la deserializzazione del body della risposta HTTP, infatti si integra con molte librerie di conversione da JSON ad oggetti Java, anche se si possono deserializzare anche altri formati.

9.6 Test del funzionamento di Tor

Creo quindi un'interfaccia chiamata Ip che contiene le annotazioni di Retrofit per definire la richiesta:

```
import okhttp3.ResponseBody;
import retrofit2.Call;
import retrofit2.http.GET;

public interface Ip {
  @GET("https://api.ipify.org?format=json") // ci sono diversi modi per
      configurare gli endpoints
  Call<ResponseBody> getIp(); // ritorna un JSON contenente l'IP in formato
      stringa
}
```

Senza un convertitore allora bisognerà ritornare un oggetto ResponseBody in quanto OkHttp è in grado di gestire solo quello.

Per la conversione da ed in JSON si possono utilizzare i seguenti framework, o librerie:

- Gson
- Jackson
- Moshi

Per convertire da e verso dei protocolli per buffer:

- Protobuf
- Wire

Per convertire da e verso XML:

- Simple XML

9.7 MainActivity

Creo quindi 2 configurazioni Retrofit, un per Tor ed una per la rete normale.

Configurazione senza proxy:

```
Retrofit.Builder builder2 = new Retrofit.Builder();
Retrofit noProxyRetrofit = builder2.baseUrl("http://localhost/").build();
Ip noProxyService = noProxyRetrofit.create(Ip.class);
Call<ResponseBody> call2 = noProxyService.getIp();
call2.enqueue(new Callback<ResponseBody>() {
    @Override
    public void onResponse(Call<ResponseBody> call, Response<ResponseBody>
        response) {
        try {
            String resp = response.body().string();
            Log.i("MainAct - no proxy", resp);
        } catch (IOException e) { e.printStackTrace(); }
    }

    @Override
    public void onFailure(Call<ResponseBody> call, Throwable t) { }
});
```

Configurazione con proxy:

```
OkHttpClient client = new OkHttpClient.Builder()
        .proxy(TorProxy.getInstance()) // qui viene settato il proxy
        .build();
Retrofit.Builder builder = new Retrofit.Builder().client(client);
Retrofit proxyRetrofit = builder.baseUrl("http://localhost/").build();
Ip proxyService = proxyRetrofit.create(Ip.class);
Call<ResponseBody> call = proxyService.getIp();
call.enqueue(new Callback<ResponseBody>() {
    @Override
    public void onResponse(Call<ResponseBody> call, Response<ResponseBody>
        response) {
        try {
            String resp = response.body().string();
            Log.i("MainAct - proxy", resp);
        } catch (IOException e) { e.printStackTrace(); }
    }

    @Override
    public void onFailure(Call<ResponseBody> call, Throwable t) { }
});
```

Questo è il risultato visualizzato su logcat:

```
I/MainAct - no proxy: {"ip":"37.16X.XXX.XXX"}
I/MainAct - proxy: {"ip":"185.56.80.65"}
```

9.8 Possibili miglioramenti all'app

Potrebbe essere utile identificare tramite pseudonimo gli utilizzatori dell'app.

Una possibile soluzione potrebbe essere quella di usare il codice identificativo dello smartphone (IMEI su GSM, MEID su CDMA).

Il problema in questo caso è che bisogna chiedere all'utente il permesso di leggere lo stato del telefono:

```
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

Risulta immediatamente ovvio il problema di questo metodo: **un'applicazione che usa Tor e che dovrebbe assicurarti l'anonimato sta chiedendo i permessi per leggere lo stato del telefono...**

Un altro metodo sarebbe quello di distribuire un ID automatico, o costringere l'utente a scegliere un nome al primo avvio dell'app.

Un miglioramento di quest'ultimo metodo è la creazione di un token casuale (la data in formato epoch con salt più una funzione di hashing) sempre al primo avvio dell'app.

Un altro possibile miglioramento consiste nel trovare il modo di ottenere la lista di circuiti che Tor ha costruito e visualizzarla da qualche parte nell'app.

10 Backend

Ora che il frontend sembra funzionare correttamente bisogna creare un hidden service che hosta delle API.

10.1 API, Web API, REST API

API è un termine generale per indicare un pezzo di codice che parla con un altro. Nello sviluppo web ci si riferisce spesso alle (web) API come un modo per recuperare le informazioni da un servizio online. Le Web API sono delle API che possono essere RESTful oppure no. REST è un insieme di regole, standard, linee guida su come creare un'interfaccia API.

10.2 Tor Hidden Service

Come detto in precedenza l'HS sarà hostato su un Raspberry, nello specifico installerò Tor su un sistema operativo basato su Debian chiamato Raspian (Raspberry Pi OS) e poi eseguirò tutte le configurazioni necessarie alla configurazione.

Si procede quindi all'installazione di Tor:

```
$ sudo apt-get install tor -y
```

Dopo l'installazione si modifica il file di configurazione di Tor:

```
sudo nano /etc/tor/torrc
```

Aperto il file bisogna andare sulla sezione degli HS ed inserire le configurazioni necessarie per impostare il percorso, le porte e l'IP dell'host che fornirà il servizio.

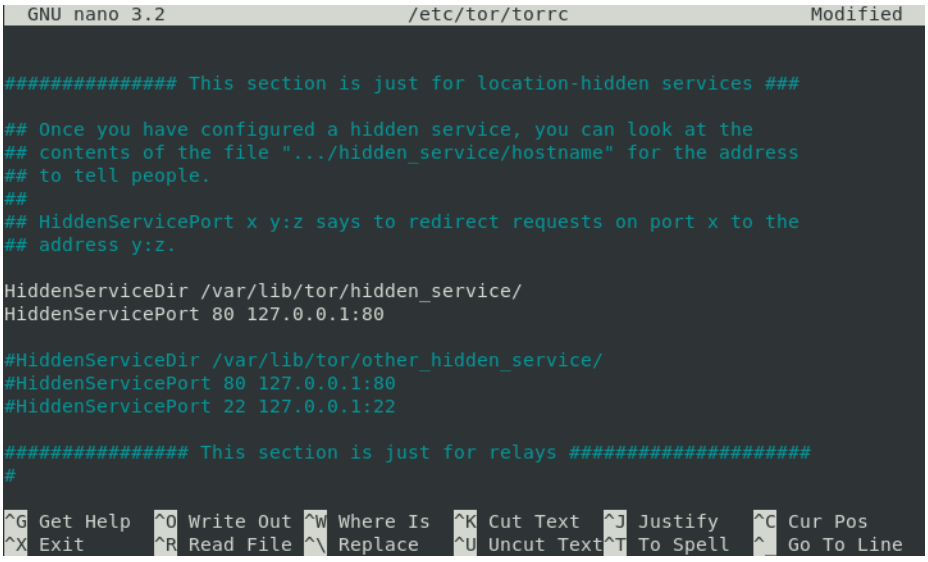


Figura 20: Impostazioni di configurazione del file torrc

Quindi salvare e riavviare tor.service:

```
sudo systemctl stop tor
sudo systemctl start tor
```

10.3 Onion domain names

Dentro il file

```
/var/lib/tor/hidden_service/hostname
```

è possibile trovare il nome pubblico del servizio che può essere comunicato alle persone, ovvero l'indirizzo che verrà usato per accedere alle API dal frontend.

10.4 NodeJS

Tor è settato, adesso bisogna creare un servizio, o un web server che si metta in ascolto sulla porta virtuale settata per l'HS. Con NodeJS risulta molto semplice creare un web server, soprattutto grazie alle librerie, o ai framework forniti, tipo express.

10.5 Express

Express è un framework per applicazioni web per Node.js, open source sotto Licenza MIT. È stato progettato per creare web application e API ed ormai è definito come il server framework standard de facto per Node.js.

10.6 Creazione dell'applicazione web

Si installano quindi i pacchetti necessari e si creano i file da utilizzare:

```
mkdir ~/tor_hs_server
cd ~/tor_hs_server
npm init
npm install express --save
```

Creo un classico server con NodeJS:

```
const PATH = require('path')
const express = require('express')
let app = express()

app.use(express.static("public"))
app.get('/', (req, res) => {
   res.sendFile(PATH.join(__dirname, "./index.html"))
})

let server = app.listen(3333, () => {
   let host = server.address().address
   let port = server.address().port
   console.log("Service listening at http://%s:%s", host, port)
})

process.on('uncaughtException', err => {
   console.error('There was an uncaught error', err)
   process.exit(1) //mandatory (as per the Node.js docs)
})
```

Quindi anche il file html da servire in modo statico:

```
<!DOCTYPE html>
<html> <body> Hello World! </body> </html>
```

Cambio la configurazione dell'HS per dirgli di reindirizzare tutte le richieste provenienti dall'esterno sulla porta 80 alla porta 3333 del localhost e poi riavvio il servizio di Tor.

```
sudo nano /etc/tor/torrc
# HiddenServiceDir /var/lib/tor/hidden_service/
# HiddenServicePort 80 127.0.0.1:3333
sudo systemctl stop tor
sudo systemctl start tor
```

Per verificare il funzionamento dell'hidden service basterà provare a collegarsi tramite Tor Browser, od Orbot.

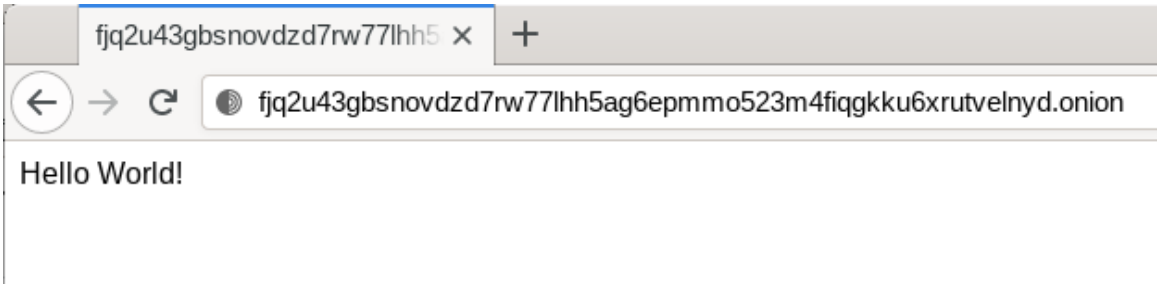


Figura 21: Accesso tramite TorBrowser all'HS

10.7 HTTP over TLS con un hidden service

Quando si accede ad un normale sito web (clearnet) tramite l'HTTP con Tor, la connessione tra il nodo di uscita e il sito web di destinazione non è crittografata. Ciò significa che il nodo di uscita può intercettare e manipolare l'intera connessione.

Questo non si applica ai servizi nascosti (.onion). In tal caso la crittografia è end-to-end ed aggiungere un altro livello HTTPS è ridondante (e controproducente perché uno degli scopi di Tor è quello di rendere anonimo il server).

Tuttavia, alcuni servizi nascosti utilizzano l'HTTPS. Facebook ne è un esempio. Viene usato il certificato SSL per fornire la prova che la connessione sta avvenendo veramente con dei server di Facebook e non a dei server falsi.

10.8 Curiosità ed osservazioni

Gli indirizzi v2 vengono deprecati nell'ottobre del 2021.

E' possibile creare un URL .onion parzialmente leggibile dall'uomo generando un numero enorme di coppie di chiavi fino a trovare un URL sufficientemente desiderabile, questa tecnica risulta più semplice sugli indirizzi v2.

L'architettura con cui funziona Tor fornisce diverse caratteristiche:

- Permette di avere un servizio web senza dover aprire le porte del router
- Hai il tuo domain name senza pagare
- Non si utilizza nessun servizio DNS

10.9 Possibili miglioramenti all'hidden service

10.9.1 Onionscan

Usare onionscan [33] per testare il proprio hidden service.

10.9.2 Esplicitare la versione dell'hidden service

E' possibile indicare esplicitamente la versione di Onion che si vuole utilizzare modificando le configurazioni sul file torrc in questo modo:

```
HiddenServiceDir /var/lib/tor/hidden_service/
HiddenServiceVersion 3 // versione esplicita
HiddenServicePort 80 127.0.0.1:3333
```

Dopo aver fatto questa modifica bisogna cancellare la cartella hidden_service e riavviare Tor, il nome del dominio pubblico cambierà.

10.9.3 Visualizzare i log di Tor a fini di monitoraggio

E' possibile utilizzare questo comando per ottenere i log inviati da Tor:

```
sudo cat /var/log/syslog | grep tor -i
```

10.9.4 Limitare l'ascolto dell'applicazione web unicamente al localhost

L'applicazione web dovrebbe essere settata per ascoltare solo il localhost.

In NodeJS può essere fatto esplicitando l'IP nel momento in cui si fa partire il server.

```
let server = app.listen(3333, '127.0.0.1',() => {
   let host = server.address().address
   let port = server.address().port
   console.log("Service listening at http://%s:%s", host, port)
})
```

10.9.5 Disabilitare la segnalazione degli errori

Esistono diversi metodi per farlo e per la maggior parte sono specifici al programma che si sta utilizzando.

Per quanto riguarda NodeJS il metodo più diretto è l'utilizzo di un blocco try/catch in ogni parte del codice che potrebbe risultare in un errore.

Express però ha la caratteristica di avere un gestore degli errori predefinito "intelligente".

In modalità sviluppo invia al browser l'intero stack trace, mentre in modalità di produzione invia solo "500 Internal Server Error".

Per configurare questo comportamento bisogna settare la variabile d'ambiente NODE_ENV, per farlo è possibile creare degli script personalizzati direttamente nel package.json.

```
"scripts": {
   "test": "echo \"Error: no test specified\" && exit 1",
   "start": "node index.js",
   "start-linux-prod1": "export NODE_ENV=production&& npm start",
   "start-linux-prod2": "export $(cat .env | xargs) && npm start"
 },
```

Nel primo caso (start-linux-prod1) la variabile d'ambiente viene settata direttamente, mentre nel secondo caso viene usato un file chiamato .env.

Questo file può essere usato per settare diverse variabili d'ambiente automaticamente.

xargs converte i dati dallo standard input (cat .env) in argomenti in modo che il seguente comando (export) possa eseguirsi usando quegli argomenti.

Questo perché alcuni comandi non possono ricevere input direttamente dallo standard input ma solo come argomenti (ad es. echo, rm).

E' possibile verificare la configurazione stampando il valore della variabile in questo modo:

```
console.log(process.env.NODE_ENV)
```

10.9.6 Divide et impera

Eseguire Tor e il servizio in macchine separate, o almeno in macchine virtuali separate.

Ad esempio se si sta utilizzando una singola macchina, è raccomandato avviare Tor sull'host e il servizio nella VM, dato che il rischio maggiore è sulla possibilità del servizio di venire violato e che tenti di disattivare il client Tor, piuttosto che viceversa.

10.9.7 Disabilitare il directory listing

Il "Directory listing", anche detto "directory indexing", riguarda la possibilità di lasciare ad un utente in visita sul sito di navigare e avere sott'occhio tutto l'albero dei file presenti sul server.

Express ha questa opzione disabilitata di default (ovvero restituisce una pagina di errore se l'utente prova a navigare tra le cartelle del server).

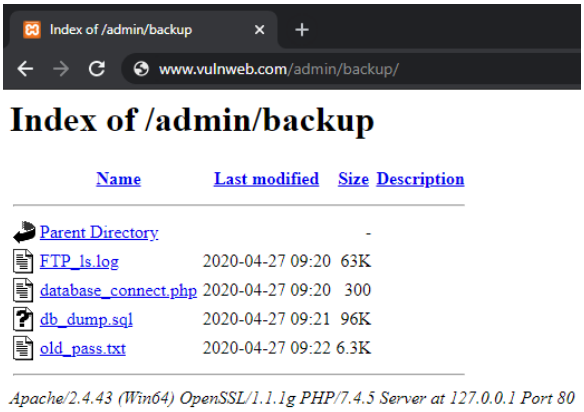


Figura 22: Esempio di directory listing

10.9.8 Rimozione della firma del server

Assicurarsi che la versione del server web, il nome del server del sistema operativo, i programmi utilizzati per creare il backend non vengano comunicati per sbaglio negli header, o nelle pagine di errore predefinite, o nelle pagine in generale.

Su Express è possibile rimuovere alcune di queste informazioni con questo comando:

```
let app = express()
app.disable('x-powered-by')
```

10.9.9 Protezione contro attacchi DoS

Se creo un'API nel mio hidden service come evito ad un utente di intasare il servizio con un numero spropositato di richieste? [34] [35]

Questo problema non sorge quando un essere umano fa 100 richieste al minuto, ma quando quest'ultimo crea un bot che ne fa 10mila al secondo.

Il modo per differenziare un essere umano da un bot è già normalmente un problema, attraverso Tor lo è ancora di più, tanto che sembra essere un problema aperto, ancora alla ricerca di una soluzione efficace.

L'approccio standard sarebbe quello di inviare un'email all'utente per confermare l'account e, se ciò non si verifica in un periodo di tempo predefinito, eliminare la registrazione. Questa soluzione banalmente non può essere implementata per natura stessa dell'applicazione che si sta creando, chiedere informazioni all'utente non è esattamente un buon metodo per costruire fiducia.

Un altro approccio potrebbe essere quello di creare una registrazione basilare con ID, password e richiedere il completamenti di CAPTCHA per le API più pesanti.

Questo potrebbe scoraggiare gli aggressori meno determinati, ma ci sono diversi modi per aggirarli, come dei servizi di completamento CAPTCHA automatici che sono relativamente economici.

Il movimento del mouse e il riconoscimento delle immagini sono due esempi di come Google gestisce il suo reCAPTCHA. L'utilizzo di un servizio terzo diventa un fonte di dubbi da non sottovalutare.

Esistono diverse possibilità che comunque non andrò ad implementare e descrivere in questa ricerca:

- SafetyNet reCAPTCHA
- SimpleCaptcha
- JCaptcha
- Android Easy Captcha [36]
  Questa libreria in particolare non si basa su un servizio terzo.
- hCaptcha

10.9.10 Altre possibilità

- Usare thttpd, perché ha "un'area" di attacco più piccola, meno possibilità di attacco dall'esterno
- Whonix
- Altre best practices [37]

11 Collegamento frontend-backend

E' arrivato il momento di creare un collegamento tra i 2 pezzi di codice.
Vado a creare un semplice database con SQLite e lo vado a popolare con una lista di link di youtube.

11.1 Backend

Creo un file per contenere delle variabili che rimarranno costanti, come ad esempio il nome del database:

```
/tor_hs_server
└── src
    └── constants.js
```

```
const PATH = require('path')

module.exports = {
   DBPATH: './db/db.sqlite',
   ROOT: PATH.join(__dirname, "../")
}
```

Creo il database e un file che contiene una sola operazione per inserire una stringa:

```
/tor_hs_server
├── db
│   ├── db_utils.js
│   └── db.js
└── src
    └── constants.js
```

```
/**
 * db.js
 */
const fs = require('fs')
const sqlite3 = require('sqlite3').verbose()
const dbpath = require('../src/constants').DBPATH

// serve a controllare l'esistenza o meno del db
fs.stat(dbpath, (err, stats) => {
   if (err == null) return console.log('il db esiste')

   // crea il db se non esiste
   let db = new sqlite3.Database(dbpath, (err) => {
       if (err) return console.error(err.message)

       // crea una tabella chiama links che ha un campo di tipo TEXT
           chiamato url
       db.run('CREATE TABLE links(url TEXT)')
   })
})
```

```
/**
 * db_utils.js
 */
const sqlite3 = require('sqlite3').verbose()
const dbpath = require('../src/constants').DBPATH

function insert_link(db, link) {
    db.run(`INSERT INTO links(url) VALUES(?)`, [link],
          // se viene usata la lambda l'oggetto 'this' diventa inutilizzabile
          function (err) { // DO NOT USE LAMBDA HERE !!
              if (err) return console.error(err.message)
              console.log(this)
          })
}

function get_link(db, callback) {
    db.all(`SELECT * from links`,
          function (err, rows) {
              if (err) return console.error(err.message)
              callback(rows)
          })
}

module.exports = {
    insert_link,
    get_link
}
```

Creo la route che contiene il codice da far partire quando l'utente naviga su un certo percorso:

```
/tor_hs_server
├── db
│   ├── db_utils.js
│   └── db.js
└── src
    ├── constants.js
    └── routes
        └── links.js
```

```
var express = require('express')
var router = express.Router()

const sqlite3 = require('sqlite3').verbose()
const dbpath = require('../constants').DBPATH
const db_utils = require('../../db/db_utils')

router.get('/', (req, res) => {
    // apre la connessione al database
    let db = new sqlite3.Database(dbpath, (err) => {
        if (err) return console.error(err.message)
        db_utils.get_link(db, (rows) => {
            console.log('richiesta GET ricevuta')
            db.close()
            res.setHeader('Content-Type', 'application/json')
            res.end(JSON.stringify(rows))
        })
    })
})
module.exports = router
```

Infine, sul file principale del server si chiama il db e si setta la route:

```
/tor_hs_server
├── db
│   ├── db_utils.js
│   └── db.js
├── src
│   ├── constants.js
│   └── routes
│       └── links.js
└── index.js
```

```
const DB = require('./db/db.js')
const links_router = require('./src/routes/links')
app.use('/api/insert_link', links_router)
```

11.1.1 Verifica tramite Tor Browser

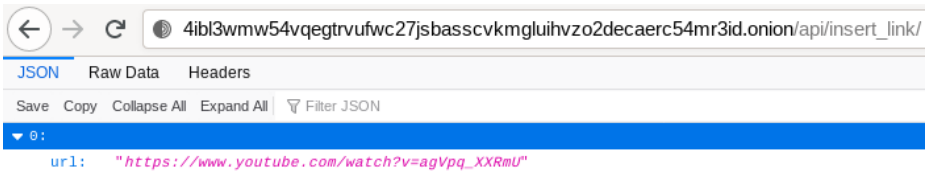


Figura 23: Accesso tramite Tor Browser all'API

11.2 Frontend

Si passa quindi all'implementazione lato frontend. Creo l'interfaccia per le richieste HTTP:

```
import okhttp3.ResponseBody;
import retrofit2.Call;
import retrofit2.http.GET;
import retrofit2.http.POST;

public interface Links {
 @GET("http://4ibl3wmw54vqegtrvufwc27jsbasscvkmgluihvzo2decaerc54mr3id.onion
     /api/insert_link/")
 Call<ResponseBody> getYTLink();
}
```

Uso poi una istanza della classe Proxy per configurare il client:

```
OkHttpClient client = new OkHttpClient.Builder()
        .proxy(TorProxy.getInstance())
        .build();
```

Con l'istanza del cliente viene configurato il proxy di Retrofit:

```
Retrofit.Builder builder = new Retrofit.Builder().client(client);
Retrofit proxyRetrofit = builder.baseUrl("http://localhost/").build();
```

Creo un bottone che quando cliccato fa partire la richiesta HTTP, utilizzando il proxy, verso l'HS:

```
Button insertReq = findViewById(R.id.buttonInsert);
insertReq.setOnClickListener(view -> {

    Links proxyLinkService = proxyRetrofit.create(Links.class);
    // chiamata GET all'hidden service
    Call<ResponseBody> insertCall = proxyLinkService.getYTLink();
    insertCall.enqueue(new Callback<ResponseBody>() {
        @Override
        public void onResponse(Call<ResponseBody> call1, Response<
            ResponseBody> response) {
            try {
                String resp = response.body().string();
                Log.i("MainAct - proxy", resp);
            } catch (IOException e) { e.printStackTrace(); }
        }

        @Override
        public void onFailure(Call<ResponseBody> call1, Throwable t) {
            t.printStackTrace();
        }
    });
});
```

Nel log su Android studio si dovrebbe poter visualizzare questo messaggio che indica che i dati sono stati inviati dall'HS all'applicazione Android passando per Tor.

```
I/MainAct - proxy: [{"url":"https://www.youtube.com/watch?v=agVpq_XXRmU"}]
```

12 Distribuzione dell'applicazione

Una volta programmato il backend e costruito l'applicazione si arriva all'ultimo problema da risolvere che è quello della distribuzione dell'applicazione Android.

12.1 Distribuzione non anonima

Se l'applicazione non contiene nulla di illegale si potrebbe pensare ad una distribuzione facile e conosciuta come il caricamento dell'app sul play store. Questo permetterebbe di costruire un po', o la quasi completa, fiducia con gli utenti Android abituati a scaricare le app in questo modo.

12.2 Distribuzione anonima

Per la distribuzione anonima è possibile utilizzare Tor per accedere ad internet e poi:

- Creare un account Github tramite Tor ed usare una semplice repository.
- Creare tramite Tor un account MEGA e usare il pacchetto Megatools che offre la possibilità di caricare sul cloud da linea di comando.
- Affittare una VPS tramite bitcoin, eseguire all'interno di essa Nextcloud.
- Registrarsi ad un forum tramite Tor e presentare la propria applicazione.

Il problema di questo metodo è che l'utente deve fidarsi di ciò che sta scaricando, un modo per risultare più affidabili potrebbe essere quello di presentare un'analisi dell'applicazione fatta da virustotal.

13 Conclusione

“nani sulle spalle dei giganti”.

Per questa tesi sono stati utilizzati diversi, grandi e complicati progetti: Tor, OpenSSL, Android NDK, etc...

Dopo la loro presentazione è stata mostrata la compilazione di Tor, poi la spiegazione del funzionamento dell’onion routing, è stata integrata la libreria Tor dentro un’applicazione Android Java ed è stata collegata ad un hidden service tramite delle RestAPI. Sono state accennate diverse migliorie e possibili problemi sia frontend che, e soprattutto, backend.

E’ stato raggiunto con successo, infine, un collegamento tramite Tor da un’applicazione Android ad un hidden service.

13.1 Sviluppi futuri

13.1.1 QSOR: Quantum-safe Onion Routing

In questo paper [38] vengono mostrate diverse soluzioni al fine di rendere Tor sicuro in un mondo quantistico. Sarebbe interessante valutarne la difficoltà di implementazione.

13.1.2 Studi per aumentare la conoscenza degli attacchi documentati

- **Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice** è un paper [16] parecchio citato il cui studio è altamente consigliato a chi è interessato all’argomento. E’ importante in quanto anche in una connessione da Tor a clearnet l’exit node risulta un anello debole del sistema.
- **Researcher Finds Tor Exit Node Adding Malware to Binaries** è un articolo [39] che ipotizza un attacco da parte di exit node maleintenzionati. Sarebbe interessante capire se il problema è stato risolto, o esistono fix documentati.
- Un video [40] che mostra diversi metodi di attacco agli HS.

13.1.3 Tor + OpenVPN

Un possibile metodo per rendere più ostico ad exit node malintenzionato di arrecare danno. In più si potrebbe riuscire a risolvere il problema degli IP degli exit node bloccati da alcuni servizi sulla clearnet. L’interesse consiste nel valutare la difficoltà implementativa e i limiti della configurazione. In questo articolo [41] un tutorial per l’installazione.

13.1.4 Confronto tra Jap, Tor e Ghostsurf

Studiare e confrontare i diversi sistemi per capire quale meglio corrisponde a quali esigenze.

13.1.5 SQLCipher: Encrypted Database

The Guardian Project ci fornisce una soluzione [42] per implementare l’amato SQLite con una difesa in più.

13.1.6 Onion v3 vanity address generation

Riuscire a generare un onion url che riporta un nome parziale nella nuova versione dei Onion. [43]

13.1.7 The Trouble with Tor

Uno dei problemi principali di Tor, che sembra non essere stato risolto, è che almeno nel 2016 il 94% del traffico sembrava essere generato da richieste automatiche di bot. Risulta importante uno studio sull'evoluzione o sulle conseguenze di questo fatto.

Elenco degli acronimi

DA Directory Authority, un relay speciale che mantiene un elenco di tutti i relay attualmente in esecuzione e che pubblica periodicamente un consenso insieme alle altre DA., page 20

HS Hidden Service, ormai chiamati Onion Service, sono dei servizi hostati in dei server raggiungibili solo tramite Tor., page 26

$HSDir$ Hidden Service Directory, sono dei nodi che hanno una certa configurazione e che hanno ricevuto una "flag" (HSDir flag). Sono nodi che vengono considerati affidabili per svolgere il compito di ricevere e distribuire gli Hidden Service Descriptors., page 28

OP Onion Proxy, ovvero i client che si connettono tramite il Proxy locale creato dal programma Tor, page 21

OR Guardare la voce "relay", page 21

$relay$ Un nodo elencato pubblicamente nella rete Tor che inoltra il traffico per conto dei client e che si registra presso le directory authorities, page 21

Appendice

A Lista dei problemi riscontrati su ArchLinux

A.1

Execution failed for task ':tor-android-binary:mergeDebugJniLibFolders'

Questo errore indica che non sono state seguite correttamente le istruzioni nel file BUILD presente all'interno della repository.

A.2

Warning: File /.android/repositories.cfg could not be loaded

Un file è mancante, per risolvere questo errore basta crearlo:

```
touch ~/.android/repositories.cfg
```

A.3

Warning: An error occurred while preparing SDK package NDK: No space left on device

Android Studio e sdkmanager tentano di decomprimere dei file zippati dentro /tmp che di default ha 4GB di spazio su Arch che non sono abbastanza per gli SDK di Android. Si può' risolvere in diversi modi:

- Aumentare la grandezza della cartella /tmp (richiede un riavvio)
- Usare un'altra cartella al posto di /tmp

Android durante il processo di download ed estrazione crea una cartella chiama PackageOperation01, se si vuole usare un'altra cartella basta creare un symlink dentro /tmp che porta ad un'altra cartella con lo stesso nome:

```
# controllare che il nome della cartella creata da Android sia
    PackageOperation01
mkdir ~/tmp/PackageOperation01
cd /tmp/
ln -s ~/tmp/PackageOperation01
```

A.4

Makefile:36: WARNING: NDK 21.4.7075529 required for reproducible builds, 22.1.7171670 is installed

Una specifica versione di Android NDK è richiesta per compilare tor, quindi basta installarla:

```
sdkmanager --install 'ndk;21.4.7075529'
export ANDROID_NDK_HOME=$ANDROID_HOME/ndk/21.4.7075529/
```

A.5

src/core/mainloop/mainloop.c:129:10: fatal error: 'systemd/sd-daemon.h' file not found

Autoconf costruisce il file ./configure da file ./configure.ac.

Questo è l'errore che mi ha convinto a cambiare sistema operativo... Per qualche motivo, tra cui dei pacchetti che potrebbero non aver scaricato i file necessari oppure è il sistema operativo ad essere settato in modo diverso.

Il problema è che il compilatore non trova gli header per fornire a torlib la possibilità di essere eseguito come daemon. Dopo diversi tentativi mi sono arreso.

Riferimenti bibliografici

[1] Tor Project. Tor faq. https://2019.www.torproject.org/docs/faq-abuse#WhatAboutCriminals, .

[2] ioerror. Recent events in egypt. https://blog.torproject.org/recent-events-egypt, 2011.

[3] Tor Project. History. https://www.torproject.org/about/history/, .

[4] By EWEN MACASKILL, Produced by FEILDING CAGE GABRIEL DANCE, and GREG CHEN. "snowden, nsa, files surveillance revelations". https://www.theguardian.com/world/interactive/2013/nov/01/snowden-nsa-files-surveillance-revelations-decoded#section/1, 1 Novembre 2013.

[5] WIRED. Secure drop | wired. https://www.wired.com/securedrop/.

[6] Seth Rosenblatt. Nsa likely targets anybody who's 'tor-curious'. https://www.cnet.com/tech/services-and-software/nsa-likely-targets-anybody-whos-tor-curious/, 3 Luglio 2014.

[7] asn Roger, David Goulet. Tor onion services: more useful than you think. https://www.cnet.com/tech/services-and-software/nsa-likely-targets-anybody-whos-tor-curious/, 29 Dicembre 2015.

[8] Roger dingledine. https://en.wikipedia.org/wiki/Roger_Dingledine.

[9] Tor Project. Tor network size. https://metrics.torproject.org/networksize.html, .

[10] Security/tor uplift. https://wiki.mozilla.org/Security/Tor_Uplift, .

[11] Tor browser for android now based on updated firefox app. https://www.androidpolice.com/2020/11/02/tor-browser-for-android-now-based-on-updated-firefox-app/, .

[12] sysrqb. Orfox paved the way for tor browser on android. https://blog.torproject.org/orfox-paved-way-tor-browser-android, 3 Settembre 2019.

[13] Nick Mathewson. A tiny introduction to asynchronous io. http://www.wangafu.net/~nickm/libevent-book/01_intro.html, 2009-2012.

[14] Urian B. Tor encryption can allegedly be accessed by the nsa, says security expert. https://www.techtimes.com/articles/262645/20210709/tor-encryption-can-allegedly-be-accessed-by-the-nsa-says-security-expert.htm, 09 Luglio 2021.

[15] By Wilson Andrews and Todd Lindeman. The black budget. https://www.washingtonpost.com/wp-srv/special/national/black-budget/, 29 Agosto 2013.

[16] David Adrian e altri 14. Imperfect forward secrecy: How diffie-hellman fails in practice. https://dl.acm.org/doi/10.1145/2810103.2813707, 12 Ottobre 2015.

[17] Luca Ziesler. Tor in 2020, what changed? https://www.secjuice.com/tor-in-2020-what-changed/, 27 Settembre 2020.

[18] nickm. Tor 0.3.2.9 is released: We have a new stable series! https://blog.torproject.org/tor-0329-released-we-have-new-stable-series, 09 Gennaio 2018.

[19] Alexander Færøy. Nextgenonions. https://gitlab.torproject.org/legacy/trac/-/wikis/doc/NextGenOnions, 15 Giugno 2020.

[20] Tor rendezvous specification - version 3. https://gitweb.torproject.org/torspec.git/tree/rend-spec-v3.txt.

[21] Lista directory authority. https://metrics.torproject.org/rs.html#search/flag:authority.

[22] Tor faq - key management. https://2019.www.torproject.org/docs/faq#KeyManagement.

[23] Relay search. https://metrics.torproject.org/rs.html#search/running:true.

[24] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. https://svn-archive.torproject.org/svn/projects/design-paper/tor-design.pdf, 1 Gennaio 2004.

[25] Nick Mathewson. Tor: The second-generation onion router, Giugno 2004. URL https://www.researchgate.net/publication/2910678_Tor_The_Second-Generation_Onion_Router.

[26] Ferry Astika. Tor: The second-generation onion router, Giugno 2004. URL https://www.researchgate.net/figure/TOR-Control-Cell-and-Relay-Cell-Structure_fig2_313951935.

[27] Tor protocol specification. https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt.

[28] Davide Alessandrini. Analisi del protocollo tor e del simulatore shadow. A.A. 2012-2013.

[29] Nick Mathewson. Tor and circumvention: Lessons learned. https://www.usenix.org/legacy/event/leet11/tech/slides/mathewson.pdf, 2011. Invited talk at Crypto 2011.

[30] Great firewall evasion. https://it.wikipedia.org/wiki/Great_Firewall#Evasione.

[31] Arun Dunna and Ciar an O'Brien. Analyzing china's blocking of unpublished tor bridges. https://www.usenix.org/system/files/conference/foci18/foci18-paper-dunna.pdf, 28 Ottobre 2019.

[32] Emily Green. What are the benefits of socks5 proxy? https://nordvpn.com/blog/socks5-proxy/#head3, 02 Giugno 2021.

[33] s rah. onionscan. https://github.com/s-rah/onionscan, 25 Febbraio 2017.

[34] asn. How to stop the onion denial (of service). https://blog.torproject.org/stop-the-onion-denial, 18 Agosto 2020.

[35] Rob Jansen and Tavish Vaidya andMicah Sherr. Point break: A study of bandwidth denial-of-service attacks against tor. https://www.usenix.org/system/files/sec19-jansen.pdf, 16 Agosto 2019.

[36] floydfix. Android-easy-captcha. https://github.com/floydfix/Android-Easy-Captcha, 10 Settembre 2016.

[37] Best practices for hosting onion services. https://riseup.net/it/security/network-security/tor/onionservices-best-practices.

[38] Zsolt Tujner, Thomas Rooijakkers, Maran van Heesch, and Melek Onen. Qsor: Quantum-safe onion routing. https://arxiv.org/abs/2001.03418, 10 Gennaio 2020.

[39] Dennis Fisher. Researcher finds tor exit node adding malware to binaries. https://threatpost.com/researcher-finds-tor-exit-node-adding-malware-to-binaries/109008/, 24 Ottobre 2014.

[40] Gareth Owen. Tor: Hidden services and deanonymisation. https://www.youtube.com/watch?v=-oTEoLB-ses, 1 Gennaio 2015.

[41] Tor + openvpn. https://www.oilandfish.com/posts/tor-openvpn.html, 19 Giugno 2021.

[42] The Guardian Project. Sqlcipher: Encrypted database. https://guardianproject.info/code/sqlcipher/, .

[43] Tor onion v3 vanity address. https://www.jamieweb.net/blog/onionv3-vanity-address/, 6 Gennaio 2018.

Significato e Ringraziamenti

Potrei scrivere più qui che nella tesi. Grazie all'università, alla possibilità di aver convissuto con persone a me coetanee, grazie per l'esperienza del Campus Party, per l'Erasmus e soprattutto per il CyberChallenge (non c'è bisogno di fare l'università basterebbe fare 4 mesi di questa splendida competizione). Grazie al professor Marcantoni di cui ho apprezzato i metodi e l'attitudine. Grazie alla mia famiglia per avermi sostenuto finanziariamente per tutto questo tempo, ma soprattutto per aver capito la strada che stavo percorrendo fatta più che altro di dubbi, crisi esistenziali e domande troppo difficili per un semplice studente di informatica. Grazie alla provvidenza, o al mio inconscio, non lo so, perché da solo ho trovato le risposte che mi servivano, un metodo sostenibile per resistere, se non migliorare, senza mai perdere se stessi, l'amore, o il divertimento e quindi la motivazione; attraverso le sfide della vita, accettando anche i vuoti, le mancanze, i demoni delle altre persone. Grazie a me stesso, perché sono il mio miglior amico e il mio peggior nemico.