SimpleX Chat: Guida definitiva a SMP e XFTP self-hosted (e come far digerire a Nginx le connessioni senza SNI)

SimpleX Chat: Guida definitiva a SMP e XFTP self-hosted (e come far digerire a Nginx le connessioni senza SNI)

Inutile girarci intorno: la maggior parte delle app di messaggistica "sicure" (sì, parliamo anche di Signal o Session) hanno comunque un tallone d'Achille strutturale. Che sia il numero di telefono, un ID utente centrale o un server di directory, c'è sempre qualcosa che riconduce le comunicazioni a una specifica identità.

Poi è arrivata **SimpleX Chat**.

SimpleX è radicalmente diverso: non ha ID utente, non ha account, non ha metadati centralizzati. Ma come funziona davvero sotto il cofano? E soprattutto, come si configura un'infrastruttura self-hosted seria con server **SMP** (Simple Message Protocol) e **XFTP** (File Transfer Protocol) senza impazzire dietro alle peculiarità del protocollo e di Nginx?

In questo articolo facciamo un viaggio a tutto tondo nell'architettura di SimpleX, spiegando come gira il fumo a livello teorico e come metterlo in piedi praticamente con Docker Compose e Nginx.

Anatomia e Filosofia di SimpleX: Come funziona la rete?

Per capire perché la configurazione di SimpleX ha delle regole tutte sue, dobbiamo capire la sua filosofia di base: **l'eliminazione totale delle identità a livello di rete**.

Le code unidirezionali (Simplex Queues)

A differenza di XMPP, Matrix o Signal, dove tu hai un indirizzo o una chiave pubblica fissa a cui gli altri inviano i messaggi, in SimpleX non esiste nulla del genere. La comunicazione avviene tramite **code unidirezionali temporanee** (da qui il nome *SimpleX*, che in ingegneria delle telecomunicazioni indica la trasmissione a senso unico).

Quando avvii una chat 1:1 con un contatto, si creano due code separate su server diversi:

1. **La tua coda di ricezione**: Viene creata sul **tuo** server SMP (es. `smp.example.org`). Tu dici al tuo amico: "*Se mi vuoi scrivere, deposita i messaggi nella coda X su smp.example.org*". Il tuo client si collegherà al tuo server per scaricare i messaggi.
2. **La sua coda di ricezione**: Viene creata sul **suo** server SMP. Quando tu scrivi a lui, il tuo client si connette al suo server e deposita il messaggio lì.

Questo significa che per parlare con una persona, **usi il tuo server per ricevere e il suo server per inviare**. Questa asimmetria strutturale rende impossibile per un singolo server SMP ricostruire il grafo delle relazioni (chi parla con chi).

Quando viene usato il tuo server?

Avendo capito questo schema, è facile individuare quando il tuo ferro (VPS/Server) viene effettivamente chiamato in causa:

- **Chat 1:1 (Ricezione)**: Ricevi messaggi dai tuoi contatti su `smp.example.org`.
- **Gruppi e Canali che crei tu**: Quando crei un gruppo o un canale, le code centrali di quel gruppo vengono ospitate sul tuo server. Tutti i partecipanti invieranno e scaricheranno i messaggi da lì.
- **Scambio File (XFTP)**: Se invii immagini o file usando il tuo server XFTP (`xftp.example.org`), i file cifrati vengono caricati lì temporaneamente, e i tuoi contatti li scaricheranno da lì.

Se invece entri in un gruppo creato da altri o rispondi in una chat 1:1, il tuo client scriverà sui server di terze parti.

La gestione dei Canali Pubblici e il problema della batteria (Chat Relay)

Nei gruppi classici senza intermediari, se invii un messaggio a un gruppo di 100 persone, il tuo telefono deve cifrare il messaggio 100 volte e inviarlo a 100 code diverse su server potenzialmente diversi. Questo processo (*client-side fan-out*), sommato al fatto che SimpleX usa pacchetti a dimensione fissa di **16 KiB** per nascondere la lunghezza dei testi, distruggerebbe la batteria e la banda del tuo smartphone (dovresti fare l'upload di 1.6 MB per un semplice "*Ciao*").

Per i canali pubblici e i grandi gruppi, SimpleX ha introdotto i **Chat Relay** (o *channel relays*):

- Il tuo client invia il post **una sola volta** (16 KiB) al Chat Relay.
- Il Chat Relay (che gira su un server con banda e CPU illimitate) si occupa di fare il *server-side fan-out*, distribuendo il messaggio a tutti i membri.
- *Nota sulla crittografia*: Essendo i canali spazi pubblici, il contenuto non è cifrato end-to-end in modo classico. Il relay può leggere i messaggi che propaga (anche se sono firmati digitalmente dall'autore per impedirne la manomissione), ma non conosce comunque l'identità (IP o account) né di chi pubblica né dei subscriber.

Le combinazioni Clearnet / Tor (Darknet) nelle chat 1:1 e di gruppo

Qui casca spesso l'asino, generando molta confusione su cosa debba girare su Tor e cosa in chiaro. Visto che in SimpleX non esiste la federazione tra i server SMP (ogni server è un'isola che non parla con gli altri), l'interazione tra utenti Clearnet e Tor segue regole logiche ben precise.

Ecco come funzionano le varie combinazioni:

Caso 1: Chat 1:1 tra utente Onion e utente Clearnet

Ipotizziamo che tu riceva su `mio_server.onion` e il tuo amico riceva su `suo_server.com` (in chiaro).

- **Tu scrivi a lui**: Il tuo client (che ha Tor attivo) si connette a `suo_server.com` passando per gli exit node di Tor. Funziona senza problemi.
- **Lui scrive a te**: Il suo client deve depositare i messaggi su `mio_server.onion`. Per farlo, **il tuo amico deve attivare il proxy Tor (es. Orbot o SOCKS5) nelle impostazioni della sua app SimpleX**. Non deve cambiare il suo server di ricezione (che rimane in chiaro), ma deve dare al suo client la capacità di navigare nella rete Tor per scrivere sulla tua coda. Se non lo fa, non potrà inviarti messaggi.

Caso 2: Entrare in un gruppo Onion

Se vedi un link di invito a un gruppo che punta a un server onion (es: `smp://...@xyz.onion/...`):

- Tutte le code del gruppo risiedono su quel server Tor.
- Per partecipare, **ti basta attivare Tor (il proxy SOCKS) nelle impostazioni della tua app**. Il tuo client userà Tor per parlare con il server del gruppo.
- **Il tuo server SMP personale non c'entra nulla**: non deve essere onion, e non riceverà alcun traffico da questo gruppo.

Caso 3: Posso usare il mio server clearnet per scaricare messaggi da un gruppo onion?

- **No.** Poiché i server SMP non si parlano tra di loro (non c'è federazione), non puoi chiedere al tuo server `smp.example.org` di fare da ponte per scaricare i messaggi dal server onion. Il tuo client deve connettersi direttamente al server del gruppo tramite Tor.

Il vero problema tecnico: SNI e Multiplexing su porta 443

Capiamo come gira il fumo a livello di rete prima di toccare il codice.

1. **SMP Server**: Funziona su TLS nativo. Usa la porta **5223** per le connessioni dirette dei client SimpleX (messaggi). Questa porta va esposta direttamente sul mondo, senza Nginx in mezzo. Inoltre, SMP serve una pagina web statica di cortesia sulla porta **443** (che noi mapperemo sulla porta locale **8444**). Per questa pagina serve un certificato Let's Encrypt valido.
2. **XFTP Server**: Funziona anch'esso su TLS nativo sulla porta **443** (che noi mapperemo sulla porta locale **5443**). XFTP non usa certificati Let's Encrypt commerciali; genera una coppia di chiavi autoprodotta al primo avvio, e la sicurezza è garantita dal client che verifica il fingerprint del server incluso nell'indirizzo di connessione.
3. **Nginx**: Se abbiamo un solo IP pubblico e vogliamo servire altri siti HTTPS sulla porta **443**, oltre alla pagina di SMP e al server XFTP, non possiamo usare un classico reverse proxy HTTP (`proxy_pass` nel blocco `http`). Dobbiamo usare il modulo **stream** di Nginx per fare un multiplexing TCP basato su SNI (*Server Name Indication*).

Ed ecco che scatta la trappola.

Per massimizzare la privacy ed evitare che chiunque analizzi il traffico di rete (es. censori di stato o ISP ficcanaso) sappia che stiamo usando SimpleX, **i client SimpleX non inviano l'SNI quando si collegano a XFTP**.

Quando Nginx riceve la connessione TLS senza SNI, la variabile `$ssl_preread_server_name` rimane vuota (`""`). Se la mappa di Nginx non gestisce esplicitamente il caso vuoto, la connessione cade nel backend di default (che di solito punta al blocco HTTP standard su `8443`). Il server web HTTP riceve traffico TLS grezzo non previsto, si incazza, sputa un errore TLS e butta giù la connessione. Nei log di Nginx non vedrete arrivare nulla, e voi impazzirete a capire perché i file non partono.

La soluzione? Mappare la stringa vuota `""` direttamente sul server XFTP.

Passo 1: Ottenere il certificato RSA 4096 (ECDSA fa schifo a SMP)

Un altro scazzo non documentato: il server SMP di SimpleX **non supporta i certificati ECDSA** (che oggi sono lo standard generato di default da Certbot). Se gli passate un certificato ECDSA, il container crasha silenziosamente o rifiuta le connessioni. Pretende tassativamente chiavi **RSA a 4096 bit**.

Forziamo Certbot a generare il certificato corretto per il nostro dominio SMP (es. `smp.example.org`):

```
sudo certbot certonly --nginx -d smp.example.org --key-type rsa --rsa-key-size 4096
```

Verificate la bontà del certificato generato con questo comando:

```
openssl x509 -in /etc/letsencrypt/live/smp.example.org/fullchain.pem -text -noout | grep "Public Key Algorithm"
```

Deve rispondere con `RSA-Encryption` e una dimensione della chiave di `4096 bit`.

Passo 2: Configurazione Docker Compose

Spostiamoci nella nostra directory di deployment (es. `/opt/smp/`) e creiamo la struttura delle cartelle:

```
mkdir -p data/smp/config data/smp/logs data/xftp/config data/xftp/logs data/xftp/files
```

Ora creiamo il file `docker-compose.yml`:

```
services:
  smp-server:
    image: simplexchat/smp-server:latest
    container_name: smp-server
    restart: unless-stopped
    env_file: [.env.smp]
    ports:
```

```
      - "5223:5223"
      - "127.0.0.1:8444:443"
    volumes:
      - ./data/smp/config:/etc/opt/simplex:z
      - ./data/smp/logs:/var/opt/simplex:z
      - /etc/letsencrypt/live/smp.example.org/fullchain.pem:/certificates/
smp.example.org.crt:ro
      - /etc/letsencrypt/live/smp.example.org/privkey.pem:/certificates/
smp.example.org.key:ro

  xftp-server:
    image: simplexchat/xftp-server:latest
    container_name: xftp-server
    restart: unless-stopped
    env_file: [.env.xftp]
    ports:
      - "127.0.0.1:5443:443"
    volumes:
      - ./data/xftp/config:/etc/opt/simplex-xftp:z
      - ./data/xftp/logs:/var/opt/simplex-xftp:z
      - ./data/xftp/files:/srv/xftp:z
```

Creiamo i relativi file d'ambiente per passare le configurazioni ai container.

Per **`.env.smp`**:

```
ADDR_PORT=smp.example.org:443
# Potete aggiungere altre variabili opzionali per limitare le connessioni o
abilitare i log
```

Per **`.env.xftp`**:

```
ADDR_PORT=xftp.example.org:443
```

Passo 3: Configurare Nginx col trucco dell'SNI vuoto

Su distribuzioni Debian/Ubuntu, il modulo `stream` di Nginx potrebbe non essere abilitato di default. Per prima cosa creiamo le link simbolico per caricarlo all'avvio di Nginx:

```
sudo ln -s /usr/share/nginx/modules-available/mod-stream.conf /etc/nginx/modules-enabled/50-mod-stream.conf
```

Ora apriamo il file di configurazione principale di Nginx (`/etc/nginx/nginx.conf`) e aggiungiamo il blocco `stream {}` **fuori** dal blocco `http {}` esistente:

```
stream {
    log_format stream_log '$remote_addr [$time_local] "$ssl_preread_server_name" -> $upstream_addr';
    access_log /var/log/nginx/stream.log stream_log;

    map $ssl_preread_server_name $backend {
        smp.example.org   127.0.0.1:8444;
        xftp.example.org  127.0.0.1:5443;
        ""                127.0.0.1:5443;   # <--- LA MAGIA: I client SimpleX senza SNI finiscono su XFTP
        default           127.0.0.1:8443;   # Tutti gli altri siti web normali del server
    }

    server {
        listen      443;
        listen      [::]:443;
        ssl_preread on;
        proxy_pass  $backend;
        proxy_connect_timeout 10s;
        proxy_timeout         600s;
    }
}
```

> ***Importante**: Avendo bindato Nginx TCP Stream sulla porta pubblica `443` , **dovete modificare tutti i vostri file di configurazione HTTP esistenti** (quelli in `/etc/nginx/sites-enabled/`). Sostituite qualsiasi occorrenza di `listen 443 ssl;` con `listen 8443 ssl;` . In questo modo, quando un browser richiede un vostro sito web normale inviando l'SNI corretto, Nginx Stream leggerà il nome host, non troverà corrispondenze per SimpleX, applicherà la regola `default` ed inoltrerà la connessione a `127.0.0.1:8443` , dove il blocco HTTP standard si occuperà di servire la pagina web normalmente.*

Per la gestione della porta 80 e del rinnovo dei certificati Certbot, create un file di configurazione minimale per SMP in `/etc/nginx/sites-enabled/smp.example.org.conf`:

```
server {
    listen 80;
    listen [::]:80;
    server_name smp.example.org;

    location /.well-known/acme-challenge/ {
        root /var/www/certbot;
    }

    location / {
        return 301 https://$host$request_uri;
    }
}
```

Riavviate Nginx per applicare le modifiche:

```
sudo nginx -t && sudo systemctl restart nginx
```

Passo 4: Hook di rinnovo per Let's Encrypt

Dato che il server SMP monta i certificati Let's Encrypt direttamente dal filesystem ed esegue l'handshake TLS internamente al container, non si accorgerà del rinnovo del certificato finché non viene riavviato.

Evitiamo che le connessioni smettano di funzionare dopo 90 giorni creando un deploy hook in Certbot che riavvia il container in automatico solo quando il certificato viene effettivamente rinnovato:

```
sudo cat > /etc/letsencrypt/renewal-hooks/deploy/restart-smp.sh << 'EOF'
#!/bin/bash
docker restart smp-server
EOF

sudo chmod +x /etc/letsencrypt/renewal-hooks/deploy/restart-smp.sh
```

Passo 5: Avvio ed estrazione degli indirizzi SimpleX

Avviamo i servizi:

```
docker compose up -d
```

Ora dobbiamo estrarre le stringhe di connessione da passare ai client.

Indirizzo SMP

Leggiamo i log di SMP per recuperare il fingerprint della chiave pubblica:

```
docker compose logs smp-server | grep Fingerprint
```

Dovreste vedere un output del tipo:

```
smp-server   | Fingerprint: _Kv-q8JOE_Hfx5Nf_fymYBGqzF-3ToKJ2ac0KZLz-RA=
```

Componiamo l'indirizzo SMP per il client: `smp://_Kv-q8JOE_Hfx5Nf_fymYBGqzF-3ToKJ2ac0KZLz-RA=@smp.example.org`

Indirizzo XFTP

Facciamo lo certo con i log di XFTP:

```
docker compose logs xftp-server | grep -E "Fingerprint|address"
```

Cercate la chiave pubblica generata automaticamente. L'indirizzo finale da inserire su SimpleX Chat per il server dei file sarà del tipo: `xftp://sUbQhuyLhyVcsMTgst0pkY8h75WcIIkP7ebuWO9IOsE=@xftp.example.org`

Verifica e test di connessione

Possiamo testare che Nginx stia facendo lo smistamento corretto direttamente da una macchina esterna usando `openssl`.

Per testare se il routing con SNI (smp) risponde correttamente:

```
openssl s_client -connect smp.example.org:443 -servername smp.example.org
```

Questo deve mostrare la catena del vostro certificato Let's Encrypt.

Per testare la connessione senza SNI (che simula il client SimpleX verso XFTP):

```
openssl s_client -connect xftp.example.org:443
```

Questo deve mostrare la catena del certificato autofirmato generato internamente da XFTP (es. `CN = xftp.example.org`, firmato da `SMP server CA` o simile).

Se entrambi i test rispondono correttamente, complimenti: avete un'infrastruttura SimpleX Chat federata, autonoma ed estremamente sicura, perfettamente integrata col vostro reverse proxy web.

Fluxer Icon

Canale di discussione e supporto

Canale di discussione e supporto. Per confrontarti su configurazioni o altro entra su fluxer.dreadful.work (https://fluxer.dreadful.work/invite/dreadful).