

# LABORATORIO DI INTERNET

## Tesina di gruppo

### Analisi di Signal v5.4.0

Gruppo 21

Giugno 2021



**Politecnico  
di Torino**

Diego Zanfardino s256536,  
Fabio Trovero s258574,  
Lorenzo Ferro s260878

prof. Mellia Marco

## 0 Descrizione

Per quest'ultima esperienza di laboratorio si è scelto di analizzare *Signal*, un'applicazione di messaggistica gratuita e open source, sviluppata e gestita da un'associazione no-profit (Signal Foundation), con possibilità di ricevere donazioni da parte degli utenti. L'applicazione è in sviluppo dal 2015, creata da Moxie Marlinspike, ex capo della sicurezza di twitter, e Brian Acton, uno dei fondatori di Whatsapp, prendendo vita dall'unione di RedPhone e TextSecure (sviluppati a partire dal 2010).

Signal può essere utilizzata per inviare e ricevere messaggi privati e di gruppo, condividere la propria posizione GPS, GIF animate, adesivi, allegati e messaggi multimediali ed effettuare videochiamate singole o di gruppo. Può altresì sostituirsi all'applicazione di sistema per lo scambio di SMS e MMS.

Il principale focus dell'applicazione è garantire la privacy degli utenti. Questa caratteristica è ottenuta grazie all'uso di diversi algoritmi di cifratura (Curve25519, AES-256, HMAC-SHA256) e crittografia end-to-end, in modo da non permettere neanche ai gestori dell'applicazione di poter intercettare e comprendere le conversazioni. Sono inoltre ridotti i metadati salvati, in particolare vengono conservati all'interno dei server solo numero di telefono, ora di ultima connessione e data di iscrizione. È tuttavia possibile impostare un PIN che permette all'utente, se lo desidera, di memorizzare sui server in forma crittografata il proprio profilo, i propri contatti e le impostazioni dell'app esclusivamente per funzioni di ripristino (per esempio, in caso di cambio telefono o reinstallazione).

Lo scopo del nostro report è quello di analizzare i meccanismi che regolano :

- apertura e chiusura app
- messaggio chat singola
- messaggio chat gruppo
- messaggi vocali
- invio file multimediali (foto, video, GIF)
- videochiamata

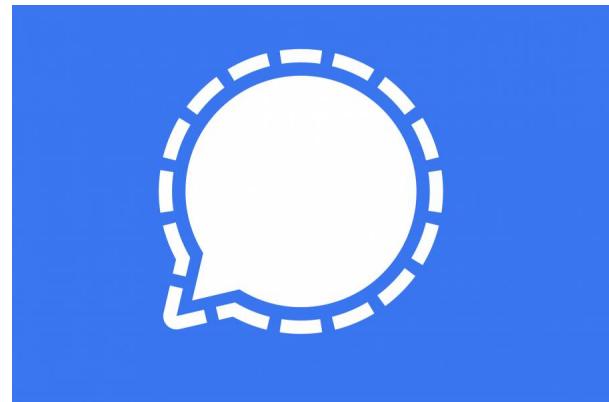
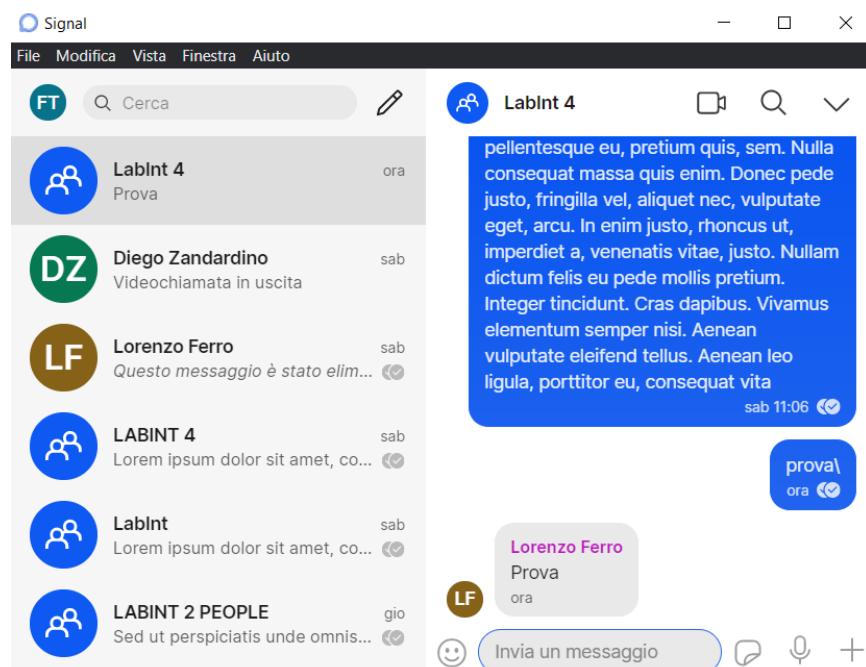
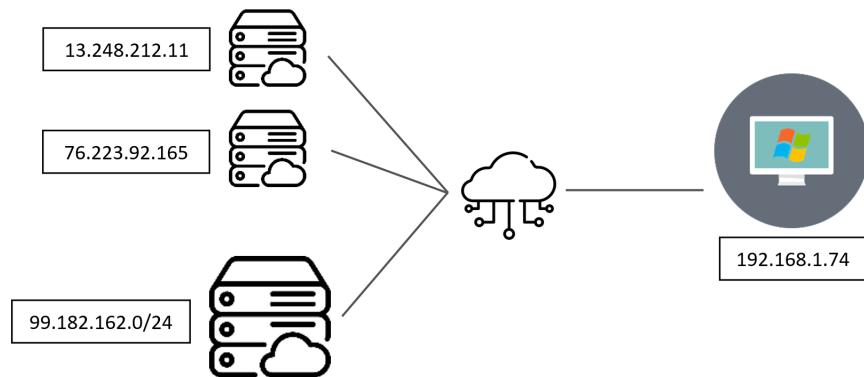


Figura 1: Logo Signal



# 1 Testbed



Per le analisi che abbiamo effettuato, si è utilizzato principalmente un computer Windows 10 (collegato tramite WiFi) con indirizzo ip 192.168.1.74, in altri scenari abbiamo utilizzato anche altri computer appartenenti a sottoreti diverse per confrontare i dati tra quelli che venivano inviati al server e quelli che arrivavano al destinatario.

Eseguendo il comando `host www.signal.org` otteniamo il seguente output:

```
www.signal.org has address 13.226.175.98
www.signal.org has address 13.226.175.88
www.signal.org has address 13.226.175.31
www.signal.org has address 13.226.175.56
```

Questi risultati fanno però riferimento al sito web di signal e non coincidono interamente con quelli utilizzati dall'applicazione. Si nota, attraverso le catture wireshark, che i server utilizzati dall'applicazione desktop a cui sono inviate le nostre richieste sono in realtà solo 76.223.92.165 e 13.248.212.111, quindi solo il secondo appartiene alla stessa sottorete degli indirizzi sopra elencati. Analizzando gli indirizzi ip con l'ausilio di internet, si scopre che entrambi i server sono di proprietà Amazon e hanno apparentemente sede a Seattle (US). In realtà effettuando un ping a questi indirizzi, entrambi rispondono con un *round-trip-time* di 20 ms, non compatibile con i tempi che realmente un pacchetto impiegerebbe per percorrere la tratta Torino/Seattle/Torino.

Durante le analisi e la stesura della relazione ci siamo accorti che vengono utilizzati altri server, sempre di proprietà di amazon (servizio AWS) con sottorete 99.182.162.0/24, che vengono interpellati quando la dimensione dei pacchetti inviati supera una certa dimensione. Nel grafico in Figura 2 vengono riportati in colore viola i Bytes ricevuti dai server 76.223.92.165 e 13.248.212.111, mentre in verde i Bytes/s ricevuti da un server nella sottorete 99.182.162.0/24. Inviando 4 messaggi di dimensione crescente di nota che i primi due cioè i più piccoli sono inviati dai primi due server, mentre i due più grandi sono inviati dall'altro server, nonostante i primi due continuino a mandare pacchetti probabilmente destinati al controllo della conversazione.

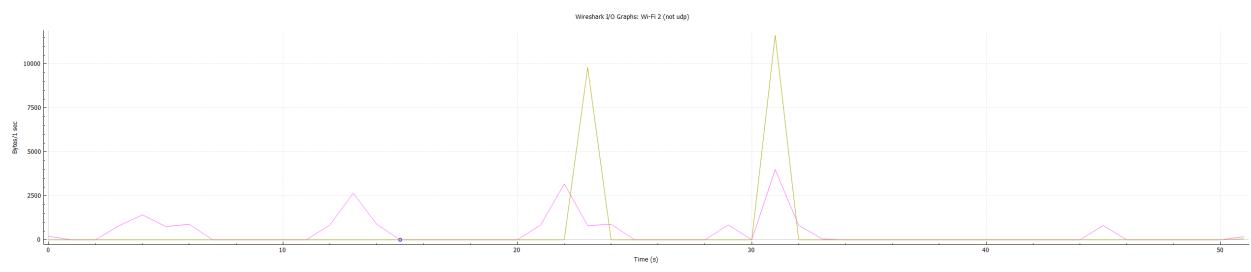
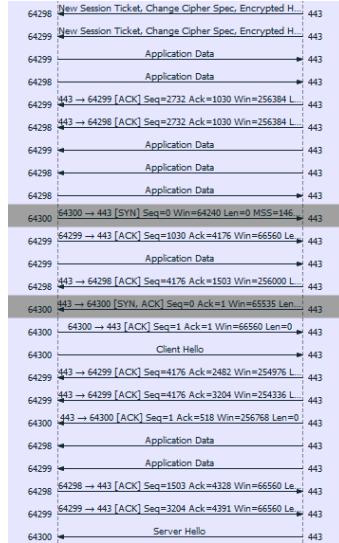
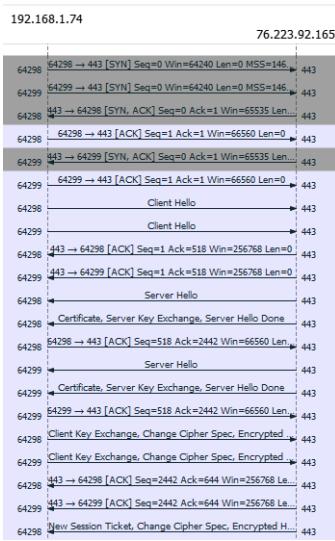


Figura 2: Cambiamento server

Questo fatto può essere parzialmente spiegato siccome il servizio AWS non assegna staticamente i propri indirizzi ma adegua gli indirizzi assegnati ai propri clienti a seconda della congestione, durante le analisi effettuate compaiono diversi indirizzi ip appartenenti alla sottorete sopraccitata.

Esegendo il comando traceroute con ciascuno dei seguenti indirizzi, non si hanno informazioni aggiuntive oltre al default gateway del router, tutti i pacchetti vanno in timeout senza ottenere risposta dai router intermedi.

## 2 Apertura e chiusura app



In primo luogo si analizza il comportamento dell'applicazione all'avvio della stessa. Analizzando il traffico generato con Wireshark e ripetendo il test più volte è possibile riconoscere un protocollo di apertura e chiusura come segue:

- TCP Three-Way-Handshake, ripetuta verso il server usando tre porte differenti
  - TCP SYN
  - TCP SYN, ACK
  - TCP ACK
- TLS v1.2 handshake, composta principalmente da:
  - *Client Hello*: messaggio di apertura connessione del client verso il server, in questo messaggio il client propone al server la versione di TLS da utilizzare, la suite di cifratura che include gli algoritmi di cifratura supportati e una stringa di bytes random (client-random)
  - *Server Hello*: in risposta al messaggio del client il server manda un messaggio contenente il certificato SSL, la suite di cifratura scelta dal server e un'ulteriore stringa di byte random (server-random)
  - Autenticazione e creazione della chiave di sessione attraverso lo scambio delle rispettive chiavi pubbliche (di client e server) e riscontro con un messaggio criptato utilizzando la rispettiva chiave privata.
    - ⇒ scambio di Application Data e keep alive (filtrati in Figura 3, 4 e 5 ) della connessione
  - Encrypted Alert: mandato dal protocollo TLS per chiudere la connessione cifrata quando non vi sono più dati da scambiare.
- Chiusura connessione TCP
  - FIN, ACK
  - ACK
  - RST, ACK

### 3 Messaggi

#### 3.1 Privati

Vediamo in seguito di analizzare il traffico generato dall'applicazione quando vengono scambiati messaggi in una chat privata. Per analizzare esaustivamente il comportamento dell'applicazione si mandano diversi messaggi di dimensione crescente. Si nota che il protocollo TCP esegue frammentazione per pacchetti di dimensione superiore ad una MTU (1452 Byte) ma questi vengono racchiusi in un unico pacchetto TLS che si occupa di riassumere il payload. Come accennato nell'introduzione prendendo in considerazione il grafico in Figura 2, abbiamo notato che per pacchetti di dimensione maggiore di 3015 Bytes vengono usati i server con indirizzi 99.182.162.0/24. Questi server sono utilizzati solo per mandare pacchetti contenenti i dati del messaggio, mentre gli altri due (76.223.92.165 e 13.248.212.111) mantengono la loro funzione scambiando messaggi di controllo.

The screenshot shows a network capture in Wireshark. It highlights several TCP segments and their corresponding TLS frames. Segment 2220 is selected, showing it contains three TLS frames: frame 2218 (payload: 0-1451), frame 2219 (payload: 1452-2903), and frame 2220 (payload: 2904-3014). The transport layer security section shows the application data protocol as http-over-tls.

No.	Time	Source	Destination	Protocol	Length	Info
2170	11:02:39.095	13.248.212.111	192.168.1.74	TCP	34	
2218	11:02:39.098	13.248.212.111	192.168.1.74	TCP	1506	1452
2219	11:02:39.142	13.248.212.111	192.168.1.74	TCP	1506	1452
2220	11:02:39.142	13.248.212.111	192.168.1.74	TLSv1.2	165	3015,111
2221	11:02:39.207	192.168.1.74	13.248.212.111	TCP	54	0
2222	11:02:39.254	192.168.1.74	13.248.212.111	TLSv1.2	110	56
2223	11:02:39.466	13.248.212.111	192.168.1.74	TCP	54	0

Figura 6: Frammentazione TLS

Durante la conversazione sono mandati diversi pacchetti per migliorare l'esperienza utente. Dopo diversi test si può ipotizzare che l'applicazione manda i seguenti pacchetti:

- un pacchetto (di dimensione che varia da mittente a destinario) per avvisare uno dei due host che l'altro sta digitando dei caratteri nella casella di testo. Se l'utente continua a digitare l'informazione è aggiornata mandando un pacchetto ogni 10 secondi, altrimenti è mandato un pacchetto quando l'host termina di digitare.
- un pacchetto (di dimensione che varia da mittente a destinario) per avvisare dell'avvenuta ricezione del messaggio da parte del destinatario.
- un pacchetto (di dimensione che varia da mittente a destinario) per avvisare della lettura del messaggio da parte del destinatario.

L'applicazione permette inoltre di disabilitare l'indicatore di scrittura e la conferma di lettura; entrambe queste modifiche sono visibili nella cattura come un numero ridotto di informazioni aggiuntive scambiate (questa impostazione ci ha permesso di identificare le probabili dimensioni dei pacchetti sopra elencati, che abbiamo riportato in appendice in tabella).

È importante notare che c'è una differenza nelle dimensioni dei pacchetti di segnalazione tra mittente e destinatario. Infatti quando il mittente sta digitando vengono mandati al server pacchetti di lunghezza 787 Bytes, mentre il destinatario riceve pacchetti di lunghezza 791 Bytes. In risposta a questi messaggi si è notato che il server manda pacchetti di dimensione rispettivamente 269 Bytes e 110 Bytes che presumibilmente sono dei messaggi di controllo a livello applicazione, con una funzione simile agli ACK TCP ma a un livello differente della pila protocollare.

The screenshot shows two separate windows of Wireshark capturing the same session. The top window is titled 'ip.src == 192.168.1.74 and (ip.dst == 13.248.212.111 or ip.dst == 76.223.92.165) and tls.app\_data'. It highlights a series of 110-byte application data frames from the source host (192.168.1.74) to the destination host (13.248.212.111), with the last frame labeled 'Mittente sta scrivendo'. The bottom window is titled 'ip.dst == 192.168.1.74 and (ip.src == 13.248.212.111 or ip.src == 76.223.92.165) and tls.app\_data'. It highlights a series of 269-byte application data frames from the destination host (13.248.212.111) back to the source host, with the last frame labeled 'Destinatario sta scrivendo'. Both windows show the TLSv1.2 protocol being used.

No.	Time	Source	Destination	Protocol	Length	Info
77	10.758472	192.168.1.74	76.223.92.165	TLSv1.2	787 Application Data	Mittente sta scrivendo
180	20.210900	192.168.1.74	76.223.92.165	TLSv1.2	787 Application Data	
241	25.068202	192.168.1.74	76.223.92.165	TLSv1.2	787 Application Data	
341	32.987468	192.168.1.74	13.248.212.111	TLSv1.2	110 Application Data	
425	42.516915	192.168.1.74	13.248.212.111	TLSv1.2	110 Application Data	Messaggi di risposta al server
529	52.521570	192.168.1.74	13.248.212.111	TLSv1.2	110 Application Data	

No.	Time	Source	Destination	Protocol	Length	Info
69	8.885789	76.223.92.165	192.168.1.74	TLSv1.2	269 Application Data	
173	18.337792	76.223.92.165	192.168.1.74	TLSv1.2	269 Application Data	Messaggi di risposta al server
235	23.201859	76.223.92.165	192.168.1.74	TLSv1.2	269 Application Data	
326	30.889116	13.248.212.111	192.168.1.74	TLSv1.2	791 Application Data	
411	40.438626	13.248.212.111	192.168.1.74	TLSv1.2	791 Application Data	Destinatario sta scrivendo
515	50.439863	13.248.212.111	192.168.1.74	TLSv1.2	791 Application Data	

Figura 7: Mittente Destinatario

### 3.2 Gruppi

Dalle catture effettuate si nota come le dimensioni dei pacchetti scambiate nei gruppi sono significativamente più grandi rispetto a quelle di uno stesso messaggio in una chat privata. Le dimensioni dei pacchetti crescono poi in maniera poco significativa al crescere del numero degli utenti del gruppo, questo probabilmente per i meccanismi usati dall'applicazione per la gestione dei gruppi. È ragionevole pensare che in aggiunta al contenuto del messaggio, vi siano ulteriori Byte che contengono le informazioni relative al gruppo ed ai suoi membri. Nel grafico in Figura 8 è rappresentata la quantità di Byte scambiata in funzione del tempo rispettivamente per:

- I) chat singola
- II) chat di gruppo composta da 3 persone
- III) chat di gruppo composta da 4 persone
- IV) chat di gruppo composta da 5 persone

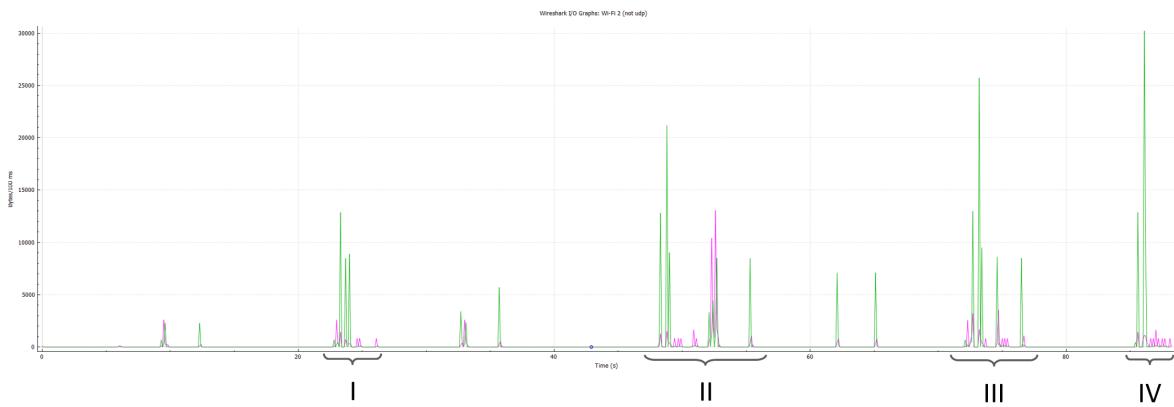


Figura 8: Confronto gruppi con fino a 5 partecipanti

L'incremento dovuto all'intestazione all'interno dei gruppi rispetto alla chat privata si può notare in Figura 8, dove sono rappresentati in verde i Bytes/s inviati dall'host che effettua la cattura al server, e in viola i Bytes/s che esso riceve.

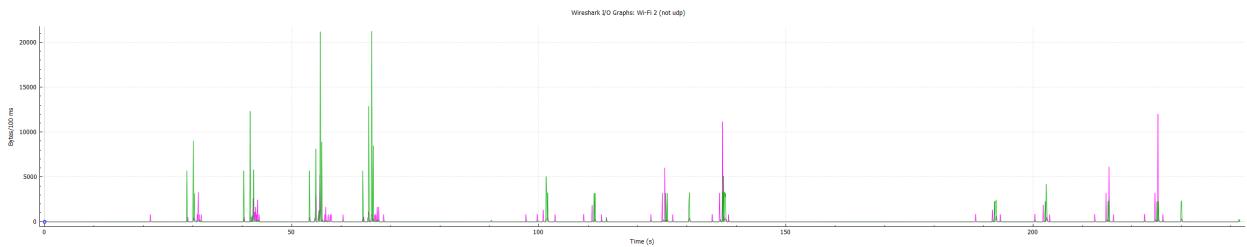


Figura 9: Pacchetti mandati in relazione a ricevuti

Il grafico in Figura 9 mostra i pacchetti generati quando ogni partecipante di un gruppo composto da 3 utenti manda quattro diversi messaggi di dimensione crescente in diversi intervalli di tempo. Nel primo intervallo di tempo è l'utente che esegue la cattura wireshark a mandare i messaggi, ed è possibile distinguere in viola il totale dei Byte al secondo ricevuti e in verde quelli mandati. Nei due intervalli di tempo successivi sono invece gli altri partecipanti del gruppo a scrivere. È possibile notare la differenza tra i Byte mandati e ricevuti in qualità di mittente e destinatario, perché il destinatario risponde con informazioni riguardanti l'avvenuta consegna o lettura del messaggio. Si può inoltre ipotizzare che la differenza tra i Byte mandati nel primo intervallo e quelli ricevuti negli intervalli successivi sia dovuta a intestazioni e informazioni di controllo generati a livello applicativo.

### 3.3 Vocali e Multimediali

Analizzando il traffico dopo aver mandato messaggi vocali o multimediali, abbiamo notato che vengono utilizzati solo i server compresi nella sottorete  $99.182.162.0/24$ , mentre si inviano messaggi di controllo sempre con  $76.223.92.165$  e  $13.248.212.111$ . L'invio dei pacchetti avviene sempre tramite lo stesso meccanismo dei messaggi, con connessione e pacchetti crittografati, facendo uso del protocollo TCP e TSL.

Riportiamo di seguito due grafici riferiti all'invio di un file video di dimensione pari a 88MB.

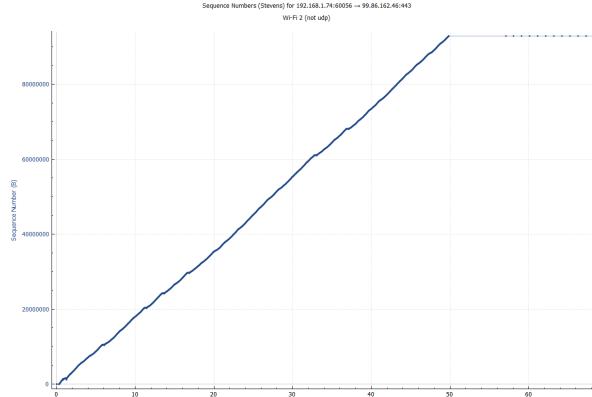


Figura 10: TCP STEVENS

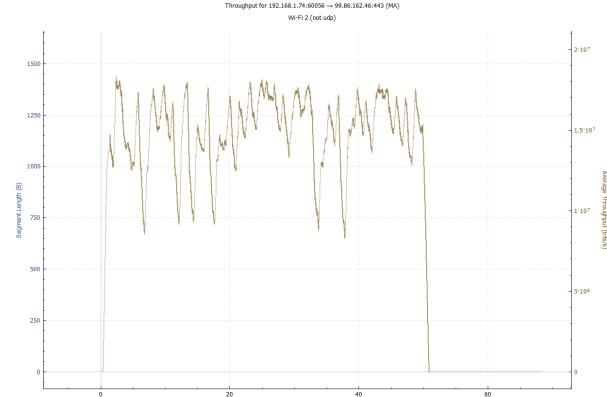


Figura 11: TCP THROUGHPUT

Ipotizzando che la connessione al server sia almeno 1Gb/s, non ci aspettiamo particolari problemi nella trasmissioni TCP causati da bottleneck. Nel grafico in Figura 11 si evidenzia che il throughput rimane abbastanza costante considerando che durante il test è stata utilizzata una connessione con velocità di upload di circa 20MB/s. Nel grafico in Figura 10 si possono notare alcune ritrasmissioni, trascurabili considerando il totale di pacchetti inviati, e l'andamento complessivo quindi è lineare.

I messaggi vocali assumono lo stesso comportamento dei file multimediali, probabilmente perchè sono considerati come file audio nel momento dell'invio.

## 4 Videochiamate

In Figura 12 è riportato lo scenario che rappresenta la videochiamata tra due utenti Signal.

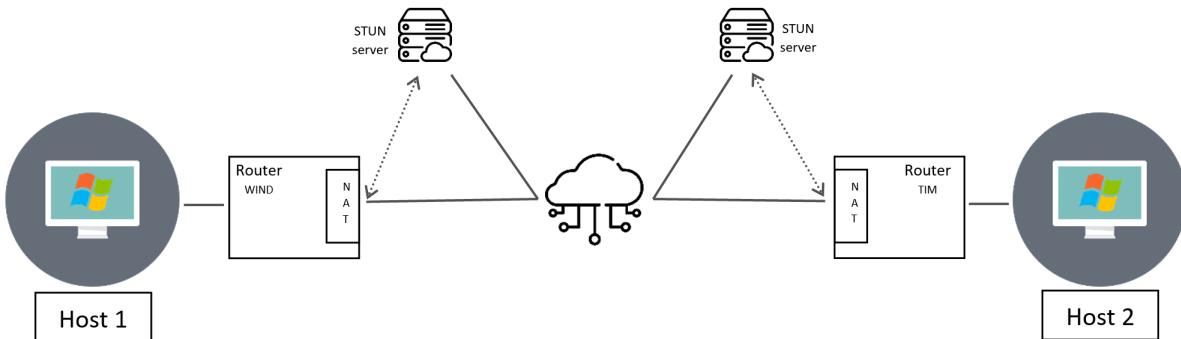


Figura 12: Testbed Videochiamate

No.	Time	Sources	Destination
93	16.152122	192.168.1.74	3.10.215.88
94	16.152353	192.168.1.74	3.10.215.88
95	16.152353	192.168.1.74	3.10.215.88
96	16.152862	192.168.1.74	3.10.215.88
98	16.177328	192.168.1.74	3.10.215.88
99	16.177661	192.168.1.74	3.10.215.88
100	16.177874	192.168.1.74	3.10.215.88
101	16.177874	192.168.1.74	3.10.215.88
102	16.178451	192.168.1.74	3.10.215.88
103	16.178595	192.168.1.74	3.10.215.88
104	16.178595	192.168.1.74	3.10.215.88
105	16.193963	3.10.215.88	192.168.1.74
106	16.194564	3.10.215.88	192.168.1.74
107	16.194564	3.10.215.88	192.168.1.74
108	16.195081	3.10.215.88	192.168.1.74
109	16.196181	3.10.215.88	192.168.1.74
110	16.218592	192.168.1.74	3.10.215.88
111	16.218842	192.168.1.74	3.10.215.88
112	16.219638	3.10.215.88	192.168.1.74

Figura 13: Apertura connessione

No.	Time	Sources	Destination	Protocol	Length	Info
1253..297	612665	3.10.215.88	192.168.1.74	STUN	126	Binding Success Response XOR-MAPPED-ADDRESS: 151.32.224.103:-
1297..397	613082	192.168.1.74	3.10.215.88	STUN	62	Binding Request
1297..397	655241	3.10.215.88	192.168.1.74	STUN	126	Binding Success Response XOR-MAPPED-ADDRESS: 151.32.224.103:-
1318..315	613082	192.168.1.74	3.10.215.88	STUN	154	Refresh Success Response lifetime: 0 user: 1623898629:601489703 realm: realm..
1328..315	648589	192.168.1.74	3.10.215.88	TCP	54 64000 + 80 [PSH, ACK] Seq=5213 Ack=1149 Win=269864 Len=112	
1338..315	648565	192.168.1.74	3.10.215.88	TCP	54 64000 + 80 [FIN, ACK] Seq=5425 Ack=1149 Win=269864 Len=0	
1338..315	648599	192.168.1.74	3.10.215.88	TCP	154 Refresh Request lifetime: 0 user: 1623898629:601489703 realm: realm..	
1338..315	648534	192.168.1.74	3.10.215.88	TCP	166 64000 + 80 [PSH, ACK] Seq=5013 Ack=2705 Win=261120 Len=112	
1338..315	648675	192.168.1.74	3.10.215.88	TCP	166 64000 + 80 [PSH, ACK] Seq=5013 Ack=2705 Win=261120 Len=0	
1338..315	692675	192.168.1.74	3.10.215.88	STUN	110	Refresh Success Response lifetime: 0
1338..315	692675	3.10.215.88	192.168.1.74	STUN	110	Refresh Success Response lifetime: 0
1338..315	692675	3.10.215.88	192.168.1.74	TCP	122 80 + 64000 [PSH, ACK] Seq=3145 Ack=5425 Win=27063 Len=68	
1338..315	692691	3.10.215.88	192.168.1.74	TCP	68 80 + 64000 [PSH, ACK] Seq=3237 Ack=5426 Win=27063 Len=0	
1338..315	694299	3.10.215.88	192.168.1.74	TCP	122 80 + 64000 [PSH, ACK] Seq=2785 Ack=3125 Win=27063 Len=68	
1338..315	694373	3.10.215.88	192.168.1.74	TCP	68 80 + 64000 [PSH, ACK] Seq=27773 Ack=3126 Win=27063 Len=0	
1338..315	694373	192.168.1.74	3.10.215.88	TCP	54 64000 + 80 [PSH, ACK] Seq=5129 Ack=27773 Win=0 Len=0	

Figura 14: Chiusura connessione

Per concludere l’analisi di tutte le funzionalità offerte dall’applicazione resta da osservare il suo comportamento durante le videochiamate. A differenza dei casi precedenti è ragionevole ipotizzare che per questo tipo di servizio sia utilizzato il protocollo UDP, sfruttato a livello applicazione da protocolli quali RTP e RTCP<sup>1</sup>. In apertura della videochiamata si nota subito come venga utilizzato il protocollo STUN<sup>2</sup>. Eseguendo il comando *whois* degli indirizzi utilizzati durante la videochiamata si ottiene un risultato inatteso: un primo utente utilizza un indirizzo appartenente ad una sottorete posseduta da Telecom Italia (Tim), mentre l’altro un indirizzo appartente ad una sottorete di proprietà Wind. Questi due altri non sono che gli internet service provider dei due utenti che hanno effettuato i test. In particolare l’utente TIM si collega con un server Wind che ha indirizzo IP corrispondente all’indirizzo pubblico assegnato al modem dell’interlocutore per mettersi in comunicazione con l’altro host (che ha WIND come ISP) e viceversa.

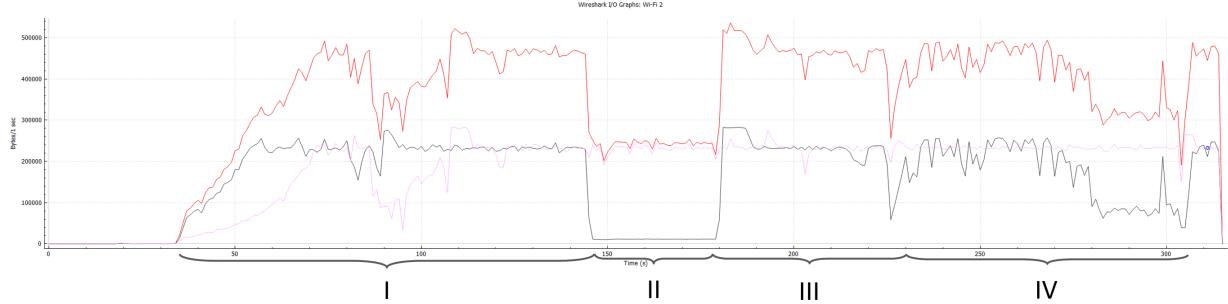


Figura 15: throughput

Per analizzare esaustivamente lo scenario in esame è stato eseguito un test che tenesse conto di tutte le proprietà di una videochiamata. Analizzando la Figura 15, che raffigura in rosa il traffico inbound, in nero il traffico outbound e in rosso quello totale, si notano diverse variazioni del throughput, causate dalle seguenti variazioni:

- I) in questo intervallo [ 35s - 150s ] i due host hanno sia microfono che videocamera accesi. L’host che genera il traffico in arrivo (in rosa avrà la stessa configurazione per tutto il test). Il calo di prestazione del destinatario che si nota in questo intervallo è dovuto ad un calo delle performance della rete.
- II) in questo intervallo [ 150s - 175s ] l’host che esegue la cattura disabilita la videocamera riducendo ad un ventesimo dell’intervallo precedente il traffico generato. Come si poteva intuire questa modifica è quella che più impatta il traffico totale generato.
- III) in questo intervallo [ 190s - 210s ] è disattivato il microfono. Dopo un’iniziale variazione dovuta alla commutazione videocamera accesa - microfono spento, si nota che la differenza rispetto al primo intervallo è minima, seppur presente.
- IV) in quest ultimo intervallo [ 220s - 300s ] viene attivata la condivisione dello schermo, che di default disabilita la videocamera. il traffico generato è inferiore di quello nel primo intervallo, pur trattandosi di una condivisione real time di file video.

Tutto il traffico generato è composto da pacchetti RTP e controllato da RTCP, entrambi *UDP-based*.

<sup>1</sup>[RFC 3550]

<sup>2</sup>Il protocollo STUN [RFC 5389] è un protocollo client-server che permette di sfruttare la tecnologia Voip, ottenendo informazioni su NAT e firewall presenti tra il computer e la rete pubblica. Tramite il protocollo UDP si mette in comunicazione con il client STUN (implementando inoltre un trasferimento affidabile) per venire a conoscenza dell’indirizzo pubblico utilizzato dal NAT, e permettere così la comunicazione con un host nella rete pubblica, facendo uso della rete di server STUN esistenti.

## 4.1 Videochiamate di gruppo

Per le videochiamate di gruppo vengono utilizzati gli stessi protocolli delle videochiamate private tra due persone, però tutti i membri della chiamata utilizzano lo stesso server (localizzato a Francoforte) per mandare e ricevere i pacchetti UDP, sempre di proprietà di Amazon.

IP Address	Country	Region	City
18.185.127.35	Germany 	Hesse	Frankfurt am Main
ISP	Organization	Latitude	Longitude
Amazon.com, Inc.	A100 ROW GmbH (amazon.com)	50.1155	8.6842

Figura 16: iplocation

## 5 Conclusioni

Il forte focus dell'applicazione da noi scelta sul garantire la privacy dei propri utenti salvando poche informazioni nei loro DataBase, utilizzando algoritmi di cifratura e connessioni criptate end-to-end, e la caratteristica dei server Amazon di cambiare spesso i server utilizzati per le conversazioni, non ha reso l'analisi particolarmente semplice.

Nonostante questo abbiamo rilevato dei comportamenti in gran parte prevedibili per un app di messaggistica, sicuramente sarebbero state di maggiore interesse capire i meccanismi di gestione degli header e delle informazioni scambiate a livello applicativo, ma ci è stato impossibile a causa dei protocolli di crittografia. Abbiamo provato a trarre diverse conclusioni sulla base del tipo di messaggi inviati e sul fatto che molti messaggi di lunghezza fissa si ripetessero con una certa frequenza, ad esempio quelli che abbiamo considerato come dei messaggi a livello applicazione che avvisavano gli interlocutori che una determinata persona stesse scrivendo, stesse registrando un audio, avesse ricevuto o visualizzato un determinato messaggio. Sarebbe stato, inoltre, interessante approfondire maggiormente come un messaggio criptato si modificasse in quanto a lunghezza e dimensione, ma non siamo riusciti ad estrarre queste informazioni con una semplice analisi tramite Wireshark.

Molto interessante anche la funzionalità implementata per le chiamate di poter condividere lo schermo del dispositivo, per quanto riguarda l'applicazione Desktop per Windows 10, inoltre in generale sembra avere una buona stabilità di connessione analizzando il throughput.

Sicuramente dai test che abbiamo fatto possiamo concludere che Signal non ha nulla da invidiare alle altre applicazioni di messaggistica, essendo utilizzatori quotidiani di Whatsapp e Telegram, e inoltre molto interessante per i più esperti che l'applicazione sia Open Source e il codice sia presente su GitHub consultabile in qualunque momento.

## A FILTRI WIRESHARK

```
ip.dst == 192.168.1.74 and ( ip.src ==13.248.212.111 or ip.src ==76.223.92.165  
or ip.dst == 99.182.162.0/8 ) and tcp and not tcp.analysis.keep_alive and  
not tcp.analysis.keep_alive_ack
```

```
ip.src == 192.168.1.74 and ( ip.dst ==13.248.212.111 or ip.dst ==76.223.92.165  
or ip.dst == 99.182.162.0/8 ) and tcp and not tcp.analysis.keep_alive and  
not tcp.analysis.keep_alive_ack
```

## B SERVER

13.248.212.111  
76.223.92.165  
99.182.162.0/8

## C Lunghezze standard - Ipotizzate

I numeri riportati nelle prime due colonne della tabella sono le dimensioni dei pacchetti TLS.

H1	H2	Azione
787	269	H1 <i>sta scrivendo</i> a H2
110	791	H2 <i>sta scrivendo</i> a H1
110	827	Conferma arrivo messaggio da H2
110	829	Lettura messaggio di H2
123	124	Keep Alive (sincronizzazione periodica)
187	110	Richiesta di cancellazione di un messaggio