ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΙΓΑΙΟΥ

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

Ζητήματα προστασίας της ανωνυμίας τελικού χρήστη σε υπηρεσίες VoIP

Συγγραφέας
Φακής Αλέξανδρος

Επιβλέπων
Καθ. Γεώργιος Καμπουράκης

ΔΙΑΤΡΙΒΗ
για την απόκτηση Διδακτορικού Διπλώματος
στο

Εργαστήριο Ασφάλειας Πληροφοριακών και Επικοινωνιακών Συστημάτων

Τμήμα Μηχανικών Πληροφοριακών και Επικοινωνιακών Συστημάτων

Πολυτεχνική Σχολή

Πανεπιστήμιο Αιγαίου

Σάμος, Μάιος 2023

University of the Aegean

Doctoral Thesis

Protecting user anonymity in VoIP services

Author
Alexandros Fakis

Supervisor
Professor Georgios Kambourakis

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

at the

Laboratory of Information and Communication Systems Security

Department of Information and Communication Systems Engineering

School of Engineering

University of the Aegean

Samos, May 2023

Υπεύθυνη Δήλωση

Εγώ ο Αλέξανδρος Φακής, δηλώνω ότι είμαι ο αποκλειστικός συγγραφέας της υποβληθείσας Διδακτορικής Διατριβής με τίτλο «Ζητήματα προστασίας της ανωνυμίας τελικού χρήστη σε υπηρεσίες VoIP». Η συγκεκριμένη Διδακτορική Διατριβή είναι πρωτότυπη και εκπονήθηκε αποκλειστικά για την απόκτηση του Διδακτορικού διπλώματος του Τμήματος Μηχανικών Πληροφοριακών και Επικοινωνιακών Συστημάτων. Κάθε βοήθεια, την οποία είχα για την προετοιμασία της, αναγνωρίζεται πλήρως και αναφέρεται επακριβώς στην εργασία.

Επίσης, επακριβώς αναφέρω στην εργασία τις πηγές, τις οποίες χρησιμοποίησα, και μνημονεύω επώνυμα τα δεδομένα ή τις ιδέες που αποτελούν προϊόν πνευματικής ιδιοκτησίας άλλων, ακόμη κι εάν η συμπερίληψη τους στην παρούσα εργασία υπήρξε έμμεση ή παραφρασμένη. Γενικότερα, βεβαιώνω ότι κατά την εκπόνηση της Διδακτορικής Διατριβής έχω τηρήσει απαρέγκλιτα όσα ο νόμος ορίζει περί διανοητικής ιδιοκτησίας, και έχω συμμορφωθεί πλήρως με τα προβλεπόμενα στο νόμο περί προστασίας προσωπικών δεδομένων και τις αρχές της Ακαδημαϊκής Δεοντολογίας.

Υπογραφή: ______________________________

Ημερομηνία: Μάιος , 2023

Declaration of Authorship

I, Alexandros Fakis, declare that this thesis entitled, "Security and privacy in SIP protocol" and the work presented in it are my own. I confirm that:

- This work was done wholly while in candidature for a research degree at this University.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this PhD thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the PhD thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ________________________________

Date: May, 2023

Advising Committee of this Doctoral Thesis:

Professor Georgios Kambourakis, Supervisor
Department of Information and Communication Systems Engineering
University of the Aegean

Professor Stefanos Gritzalis, Advisor
Department of Digital Systems
University of Piraeus

Associate Professor Panagiotis Rizomiliotis, Advisor
Department of Informatics and Telematics
Harokopio University of Athens

University of the Aegean, Greece
2023

Approved by the Examining Committee:

Professor Stefanos Gritzalis
University of Piraeus, Greece

Professor Georgios Kambourakis
University of the Aegean, Greece

Professor Christos Kalloniatis
University of the Aegean, Greece

Associate Professor Panagiotis Rizomiliotis
Harokopio University of Athens, Greece

Associate Professor Elisavet Konstantinou
University of the Aegean, Greece

Assistant Professor Georgios Stergiopoulos
University of the Aegean, Greece

Assistant Professor Marios Anagnostopoulos
Aalborg University, Denmark

University of the Aegean, Greece
2023

Abstract

Department of Information and Communication Systems Engineering
School of Engineering
University of the Aegean

Doctor of Philosophy
by Alexandros Fakis

With the proliferation of high-speed Internet and the latest generations of cellular mobile communications, Session Initiation Protocol (SIP) has become a key component to realizing high-quality multimedia services. Nevertheless, SIP's inherent lack of security measures renders users vulnerable to a range of passive and active attacks. For instance, SIP message headers are transmitted in the clear, allowing eavesdroppers to observe the communications, possibly recording sensitive information such as the identity of the caller and the called parties. On the other hand, relying on protocols like Transport Layer Security (TLS) to encrypt SIP signaling or a Virtual Private Network (VPN) to achieve anonymity may result in significant delays, reducing the Quality of Service (QoS).

SIP is typically used as a signaling protocol for Web Real-Time Communication (WebRTC)-based communication systems. WebRTC also presents certain weaknesses when it comes to the protection of end-user's privacy. For instance, the users' IP address can be exposed if a website exploits hidden Interactive Connectivity Establishment (ICE) requests. Specifically, in this case, a website can use WebRTC to make requests to the Session Traversal Utilities for NAT (STUN) and Traversal Using Relays around NAT (TURN) servers, which can eventually reveal the end-users' IP addresses. Attackers can then capitalize on this information to mount various attacks, including Denial of Service (DoS) or more targeted ones against specific users.

The primary contribution of the present Phd thesis concentrates on enhancing the privacy capacity of SIP and WebRTC protocols, offering and evaluating novel solutions towards improving the end-user's anonymity level. On the one hand, this is done by utilizing the Onion Router (Tor) and I2P anonymity networks as a robust solution towards achieving SIP message obfuscation. This ensures both anonymity and, at a certain degree, confidentiality as the signalling traffic is routed through multiple nodes, enjoying multiple layers of encryption. On the other hand, the thesis explores alternative user anonymity-preserving solutions, including the creation of a custom-tailored, obfuscated, cryptographically-protected application layer scheme to safeguard the privacy and

anonymity of the end-users. No less important, we address the privacy shortcomings of WebRTC, i.e., the potential leak of sensitive information to malicious websites. This is done through the implementation of two distinct schemes for inspecting such sites prior to their loading into the end-user's browser.

Greek Abstract

(Εκτεταμένη Περίληψη)

Τμήμα Μηχανικών Πληροφοριακών και Επικοινωνιακών Συστημάτων
Πολυτεχνική Σχολή
Πανεπιστήμιο Αιγαίου

Διδακτορική διατριβή
του Αλέξανδρου Φακή

Με την εξάπλωση του Διαδικτύου υψηλών ταχυτήτων σε συνδυασμό με τις σύγχρονες γενιές κυψελωτών δικτύων επικοινωνιών, το πρωτόκολλο SIP αποτελεί βασική συνιστώσα για την παροχή προηγμένων πολυμεσικών υπηρεσιών. Ωστόσο, η εγγενής έλλειψη μέτρων ασφαλείας του SIP καθιστά τους τελικούς χρήστες ευάλωτους σε μια πλειάδα παθητικών και ενεργητικών επιθέσεων. Παραδείγματος χάριν, οι επικεφαλίδες των μηνυμάτων SIP μεταδίδονται σε μορφή αρχικού κειμένου (cleartext), επιτρέποντας στους ωτακουστές να παρακολουθούν την επικοινωνία, καταγράφοντας ενδεχομένως ευαίσθητες πληροφορίες, όπως η ταυτότητα των επικοινωνούντων μερών. Από την άλλη πλευρά, η χρήση παραδοσιακών λύσεων όπως το πρωτόκολλο TLS για την κρυπτογραφική εξασφάλιση των μηνυμάτων σηματοδοσίας SIP στο επίπεδο μεταφοράς ή η αξιοποίηση εικονικών ιδιωτικών δικτύων (VPN) για την επίτευξη ανωνυμίας μπορεί να οδηγήσει σε σημαντικές καθυστερήσεις, με αποτέλεσμα τη μείωση του επιπέδου ποιότητας της υπηρεσίας.

Επιπλέον, το SIP χρησιμοποιείται συνήθως ως πρωτόκολλο σηματοδοσίας για συστήματα επικοινωνίας που βασίζονται στην τεχνολογία WebRTC. Το WebRTC παρουσιάζει κι αυτό συγκεκριμένες αδυναμίες όσον αφορά στην προστασία της ιδιωτικότητας των τελικών χρηστών. Παραδείγματος χάριν, η διεύθυνση IP των χρηστών μπορεί να εκτεθεί εάν ένας ιστότοπος επιχειρεί να εκτελέσει αιτήματα Interactive Connectivity Establishment (ICE). Συγκεκριμένα, σε αυτήν την περίπτωση, ένας ιστότοπος μπορεί να χρησιμοποιήσει το WebRTC ώστε να υποβάλει αιτήματα προς κάποιον STUN ή TURN εξυπηρετητή, με αποτέλεσμα να αποκαλύπτονται οι διευθύνσεις IP των τελικών χρηστών. Οι επιτιθέμενοι μπορούν στη συνέχεια να αξιοποιήσουν αυτές τις πληροφορίες για να πραγματοποιήσουν διάφορες επιθέσεις, συμπεριλαμβανομένης της άρνησης υπηρεσίας ή άλλων περισσότερο στοχευμένων επιθέσεων εναντίον συγκεκριμένων χρηστών.

Η βασική συνεισφορά της παρούσας διδακτορικής διατριβής επικεντρώνεται στην ενίσχυση της ιδιωτικότητας των πρωτοκόλλων SIP και WebRTC, εξετάζοντας, προτείνοντας, και αξιολογώντας καινοτόμες λύσεις για τη βελτίωση του επιπέδου ανωνυμίας του τελικού χρήστη. Αρχικά, αυτό πραγματοποιείται αξιοποιώντας τα δίκτυα ανωνυμίας Tor και I2P ως ισχυρές λύσεις για την επίτευξη ανωνυμίας στα δεδομένα της σηματοδοσίας SIP. Η χρήση των παραπάνω δικτύων εξασφαλίζει τόσο την ανωνυμία όσο και, σε κάποιο βαθμό, την εμπιστευτικότητα της πληροφορίας, καθώς η μετάδοση των σχετικών μηνυμάτων δρομολογείται μέσω πολλαπλών κόμβων παρέχοντας ταυτόχρονα πολλαπλά επίπεδα κρυπτογράφησης. Επιπλέον, η παρούσα διατριβή προτείνει και αξιολογεί πειραματικά εναλλακτικές, προσαρμοσμένες στο συγκεκριμένο πρόβλημα, λύσεις, συμπεριλαμβανομένης της δημιουργίας κρυπτογραφικού τούνελ (tunnel) επιπέδου εφαρμογής για τη διασφάλιση της ιδιωτικότητας και της ανωνυμίας των τελικών χρηστών σε επικοινωνίες που βασίζονται στο SIP.

Επιπροσθέτως, η διατριβή προτείνει, υλοποιεί, και αξιολογεί λύσεις για τη θεραπεία των προβλημάτων ιδιωτικότητας υπηρεσιών WebRTC, συγκεκριμένα τη διαρροή ευαίσθητων πληροφοριών σε κακόβουλους ιστότοπους. Αυτό επιτυγχάνεται μέσω της εφαρμογής δύο διαφορετικών μεθόδων ελέγχου τέτοιων ιστότοπων πριν από την καταφόρτωση του πηγαίου κώδικά τους στο πρόγραμμα περιήγησης (browser) του τελικού χρήστη.

Acknowledgements

I would like to express my deepest gratitude to my advisor Professor Georgios Kambourakis, for the invaluable guidance, unwavering support, and boundless expertise throughout the entire journey of my PhD. His encouragement, insightful feedback, and mentorship have been instrumental in shaping my research and pushing me to achieve my best.

In addition, I want to thank Dr. Georgios Karopoulos, an exceptional and dedicated researcher who had a significant and influential role in my PhD progression. His expertise, guidance, and collaborative efforts have been invaluable.

I would like to express my heartfelt gratitude to Dr. Zisis Tsiatsikas. His encouragement, reassurance, and belief in my abilities have helped me overcome moments of self-doubt and have motivated me to persevere. Whether it was through a meaningful conversation over a cup of coffee or his empathetic presence during times of stress, he consistently demonstrated their commitment to my personal and professional growth.

I would also like to acknowledge the support and encouragement of my colleague and fellow researcher, Dr. Dimitrios Papamartzivanos. His insightful discussions and collaborative spirit have greatly contributed to the development and refinement of my research ideas.

Moreover, I would like to express my heartfelt gratitude to Mrs Sofia Anagnwstou, for her unwavering love, support, and understanding throughout my PhD journey. Her presence, encouragement, and belief in me have been a constant source of strength, and her unwavering support has helped me navigate the challenges with resilience.

I am incredibly fortunate to have the unwavering support and love of my parents, Nikolao and Evgenia Faki. From the very beginning of this journey, they have stood by my side as not only my parents, but also my best friends. Their constant encouragement, belief in my abilities, and sacrifices have been the bedrock of my success. Without their guidance, understanding, and unconditional support, this entire journey would have been impossible.

Dedicated to my parents

Nikolao Faki & Eugenia Faki.

Contents

List of Figures

List of Tables

Abbreviations

AON	**A**nonymization **O**verlay **N**etwork
API	**A**pplication **P**rogramming **I**nterface
CA	**C**ertificate **A**uthority
CDR	**C**all **D**etail **R**ecord
CDF	**C**umulative **D**istribution **F**unction
DOM	**D**ocument **O**bject **M**odel
DOS	**D**enial **O**f **S**ervice
DTLS	**D**atagram **T**ransport **L**ayer **S**ecurity
HTTP	**H**yper**T**ext **T**ransfer **P**rotocol
I2P	**I**nvisible **I**nternet **P**roject
ICE	**I**nteractive **C**onnectivity **E**stablishment
ICT	**I**nformation and **C**ommunications **T**echnology
JSEP	**J**avaScript **S**ession **E**stablishment **P**rotocol
KDC	**K**ey **D**istribution **C**enter
LDAP	**L**ightweight **D**irectory **A**ccess **P**rotocol
OSI	**O**pen **S**ystems **I**nterconnection
OS	**O**perating **S**ystem
P2P	**P**eer to **P**eer
QoS	**Q**uality **o**f **S**ervice
RFC	**R**equest **F**or **C**omments
SDP	**S**ession **D**escription **P**rotocol
SIP	**S**ession **I**nitiation **P**rotocol
SMTP	**S**imple **M**ail **T**ransfer **P**rotocol
SSL	**S**ecure **S**ockets **L**ayer
STUN	**S**ession **T**raversal **U**tilities for **NAT**

S/MIME	**S**/**M**ultipurpose **I**nternet **M**ail **E**xtensions
TCP	**T**ransmission **C**ontrol **P**rotocol
TLS	**T**ransport **L**ayer **S**ecurity
TOR	**T**he **O**nion **R**outing
TTFB	**T**ime **T**o **F**irst **B**yte
TURN	**T**raversal **U**sing **R**elays around **NAT**
UDP	**U**ser **D**atagram **P**rotocol
URI	**U**niform **R**esource **I**dentifier
VM	**V**irtual **M**achine
VPN	**V**irtual **P**rivate **N**etwork
VoIP	**V**oice **o**ver **I**nternet **P**rotocol
WebRTC	**W**eb **R**eal**T**ime **C**ommunication
W3C	**W**ide **W**eb**C**onsortium

Chapter 1

Introduction

As Information and Communications Technology (ICT) technological means become increasingly integrated in our daily lives, privacy violations become more severe, particularly in user-to-user communication. Multimedia services, e.g., Voice and video calling, have become the most prevalent forms of modern communication as a result of the proliferation of high-speed Internet and the advancements of cellular mobile communications. In particular, a steep increase in the use of multimedia modes of communication has been observed in the past three years, mostly due to the emergence of the COVID-19 pandemic. Namely, many organizations resorted to remote communication and collaboration, leading to a surge in the use of multimedia modes of communication [1].

Due to low end-to-end delays and high bandwidth requirements, Voice over IP (VoIP) protocols have experienced unprecedented growth in recent years. Its cost-effectiveness and superior functionality over the traditional Public Switched Telephone Network (PSTN) revolutionized formal telephone communication in a profound way. Among others, VoIP reduces the cost of international phone calls, online meetings, and education. The term “Copper Switch Off” [2] refers to the plan by Openreach Limited, the company that maintains nearly all telephone cables in the United Kingdom, to replace all traditional telephone lines with newer, faster, and more dependable technology by 2027. According to the Copper Switch Off directive, the entire analog network used for telephone communications in most of the European countries will be replaced by VoIP technology by 2025 for facilitating the technological evolution of communications.

Security and privacy are top concerns for any business or individual, so it is no wonder that Session Initiation Protocol (SIP) security and privacy is one of the most commonly discussed topics in VoIP. SIP is an application layer signaling protocol responsible for establishing, modifying, and terminating multimedia sessions. Because SIP allows users to establish and terminate voice and multimedia sessions over the Internet, it brings along additional risks not present in traditional voice calls, such as eavesdropping or data theft. These threats can leave businesses and customers vulnerable to unwanted intrusions and security breaches that can jeopardize sensitive personal and business information.

SIP is a text-based protocol, incorporating many elements of the Hypertext Transfer Protocol (HTTP) and the Simple Mail Transfer Protocol (SMTP). Similarly to these protocols, SIP transmits data in plaintext without using any kind of encryption, which means that any information included in the message headers or body can be intercepted and read by third-parties. Specifically, the SIP header contains information about the caller and the callee, including their respective IP addresses.

Bearing in mind the lack of privacy and anonymity in the SIP protocol, a variety of solutions are required for SIP's full message obfuscation. The first solution involves the use of the well-known Onion Router (Tor) network to conceal the IP addresses of both sides. Tor is a free and open source software application that enables anonymous Internet communication. Tor functions by routing traffic through a vast network of relays operated by volunteers around the globe. These relays anonymize the traffic, making it more challenging to identify the source or destination of communications.

Initially, the client program utilizes a central directory known as the Onion Router to route traffic through the Tor network. This central directory provides a list of other Tor nodes and routes traffic through them before transmitting it to its ultimate destination. The network's nodes employ specialized encryption techniques to conceal the origin of traffic from the relay. This method is known as telescopic encryption, a cryptographic technique where the final layer of encryption is split into multiple smaller layers, with each layer corresponding to a different node in the network. This allows the message to be routed through multiple nodes without revealing the final destination until it reaches the last node, which decrypts the final layer and forwards the message to its destination. This procedure ensures that no one can trace the origin of the traffic, preventing it from being monitored by third parties. When utilizing Tor, both parties will have the capability to conceal their actual IP addresses. However, to achieve this, it is imperative that both parties establish a connection with the Tor network. In addition, any communication between the parties will be encrypted and routed through multiple nodes, making it nearly impossible for a third party to eavesdrop. By routing SIP traffic through the Tor network, not only are the messages themselves encrypted, but the IP addresses of the sending and receiving devices are also concealed, ensuring complete confidentiality and anonymity at all levels of the communication stack.

Invisible Internet Protocol, also known as I2P [3], is an alternative anonymity solution that can also be used to safeguard SIP signaling. I2P is a network layer designed to provide its users with privacy, security, and anonymity. I2P functions by routing traffic through multiple nodes and employing a variety of techniques, such as onion routing and garlic routing, to conceal the origin and destination of data. In addition, it employs layered encryption to prevent traffic analysis and man-in-the-middle attacks. Despite I2P's lower adoption rate compared to Tor, it can, with proper configuration, offer a strong alternative solution for keeping VoIP calls private. Other anonymity networks could be considered, although most of them will either come with the cost of increased communication latency [4] or the possibility of connection to untrusted nodes.

Privacy can also be achieved either by the construction of lower level tunnels, say, via the use of Transport Layer Security (TLS) or IPsec protocols, or by employing a custom-tailored solution. However, TLS and IPsec are known for leading to undesirable, non-affordable delays, and thus the need for a SIP-oriented solution seems preferable.

Another prevalent VoIP technology that is currently utilized by numerous web applications is Web Real-Time Communication (WebRTC) [5]. WebRTC is an application protocol that employs SIP protocol to enable secure and reliable communications between multiple parties. To establish peer-to-peer (P2P) connections and achieve real-time web-based communication, the WebRTC framework requires address information of the communicating peers. This means that users behind, say, Network Address Translation (NAT) or firewalls normally rely on the Interactive Connectivity Establishment (ICE) framework for the sake of negotiating information about the connection and media transferring. This typically involves Session Traversal Utilities for NAT (STUN) or Traversal using Relays around NAT (TURN) servers.

The latter entities assist peers to discover each other's private and public socket (IP:port), and appropriately relay traffic if direct connection fails. Nevertheless, these *IP:port* pieces of data can be easily captured by anyone who controls the corresponding STUN/TURN server, and even more become readily available to the JavaScript application running on the webpage. While this is acceptable for a user who deliberately initiates a WebRTC connection, it becomes a worrisome privacy issue for those who are unaware that such a connection is attempted. Furthermore, the STUN and TURN servers can acquire more information about the local network architecture compared to what is exposed in usual HTTP(S) interactions, where only the public source IP address of the client performing an HTTP(S) request is visible. Even though this problem is well-known in the related literature, no practical solution has been proposed so far. While intermediate proxies could be a cost-effective solution for protecting SIP's privacy and anonymity, in WebRTC, an intermediate gateway could detect any WebRTC Application Programming Interface (API) call prior to its execution and inform the end-user accordingly about the intentions of the visited webpage.

Keeping in mind the privacy and anonymity shortcomings existing in VoIP protocols, specifically in SIP and WebRTC, the purpose of the current thesis is twofold. First, to provide a range of robust solutions to mitigate the corresponding shortcomings and compare the proposed solutions to one another. Second, to detail the way analogous solutions could be bypassed by opponents.

1.1 Motivation

With the proliferation of VoIP technologies, it is imperative to protect user privacy and anonymity. Several legacy protocols in this domain are several years old and lack any by-design security and privacy mechanisms. For instance, as previously indicated, SIP does not provide any measure to protect user's privacy. End-users must therefore utilize add-on or external solutions. The latter focus on the encryption of sensitive SIP header fields and may be combined with an anonymization network to provide a comprehensive, cross-layer solution. To this end, as already pointed out, this thesis details a number of application and network layer anonymity-preserving schemes for protecting the identities of the communicating peers.

Even among the most recent VoIP technologies, the issue of user anonymity persists, despite the numerous research studies devoted to safeguarding anonymity and privacy, while maintaining low latency to satisfy quality of service (QoS) requirements [6, 7, 8]. This is also showcased by the WebRTC protocol design, which is one of the most modern VoIP technologies used by main multimedia software, such as Google Meet and Discord. As part of WebRTC, TURN and STUN servers are utilized to establish peer-to-peer communications between two or more peers over the Internet. In particular, STUN servers assist the user's device in identifying its public IP address and port when this functionality is blocked by NAT routers or firewalls. The STUN server will respond with the IP address of the requesting device when the webpage makes a request. This response will then be available to the JavaScript code on the webpage, allowing it to retrieve the peer's IP address. As a (negative) result, any website can leverage Javascript to force the user's device to connect to a STUN server controlled by the owner of the website for revealing the user's actual IP address. Consequently, even if an anonymity network is in use, such as Tor, a malicious website is still capable of discerning the actual identity of a user in terms of its IP address.

The key motivations of this work can be summarized as follows:

- There is a substantial amount of literature focused on protecting SIP users' privacy or anonymity at the application layer, whereas solutions for the transport layer have yet to be developed.
- In the majority of cases, ready-to-deploy, well-respected solutions like Tor are effective for achieving cross-layer anonymity; nevertheless, the additional penalty in terms of network delay should be examined. As an alternative, particularly for SIP, a more straightforward and lightweight approach could provide adequate anonymity with significantly fewer delays.
- Currently, in the absence of a straightforward method, it is infeasible to detect whether a website is sneakily tracking end-users' IP addresses via the ICE protocol.

1.2 Methodology and Milestones

The research methodology involved in this PhD thesis comprises four stages aimed at addressing both SIP and WebRTC anonymity issues, as mentioned in subsection 1.1. These stages include:

i. Provide a thorough study of the privacy, particularly anonymity, issues existing in the most common VoIP protocols. The purpose of this analysis is to offer a comprehensive, overarching understanding of existing VoIP anonymity issues and possible solutions.

ii. Design and implementation of a SIP-over-Tor scheme to assess the performance of the most popular anonymity overlay when used to establish a SIP session. That is, the SIP traffic is routed through Tor using open-source tools, and multiple performance tests are conducted for evaluating its performance. As part of our efforts to achieve a higher level of security, namely security in a cross-layer fashion, we combine the privacy provided by the PrivaSIP scheme [9, 10] with the anonymity provided by Tor. Additionally, we evaluate the impact on SIP performance in terms of additional delays introduced by any of the previously mentioned solutions.

iii. The design and implementation of a novel privacy-preserving SIP signaling scheme. To this end, we exploit the knowledge gained from the second milestone. Specifically, we design and evaluate an "onion-routing" approach, similar to Tor, where selected sensitive fields of SIP messages are encrypted using either asymmetric or symmetric encryption. In addition to Tor, which was utilized in the preceding milestone, we will also employ I2P. These additional schemes cater for a comprehensive comparison among the various anonymity solutions, including a clearer assessment of their performance.

iv. The implementation of novel security solutions to mitigate the issue of detecting and preventing in real-time the execution of STUN/TURN hidden, privacy-invading requests. This comes in the form of two distinct mechanisms, namely, a browser extension and an HTTP gateway. The intended purpose of both these mechanisms is to scrutinize the Javascript code embedded within a webpage, prior to its execution on the user's browser.

1.3 Thesis contributions

The main goal of this PhD research is the design and implementation of various methods to successfully preserve the anonymity and privacy of VoIP users.

As an initial contribution, we present techniques for enhancing the level of anonymity of VoIP users utilizing existing anonymity technologies. Specifically, we combined a privacy-preserving solution known as PrivaSIP [9, 10] with the most prevalent anonymity network, namely Tor. To protect sensitive fields in SIP messages, PrivaSIP implements field-level encryption, enabling privacy at the application layer. On the other hand, Tor offers anonymity at the third layer of the Open Systems Interconnection (OSI) model, i.e., the network level. By combining these solutions, we achieve anonymity in a cross-layer fashion.

Moreover, a major contribution of this PhD thesis is the development of a novel tailor-made encryption-based privacy-preserving scheme, where sensitive information contained in a SIP message is encrypted in a multilayer fashion. This is done by capitalizing on the onion routing concept.

An important part of this study is to evaluate the performance of the SIP protocol in conjunction with the previous mentioned solutions. To this end, we compared the session initialization delay of the aforementioned solutions. Additionally, we present a brief comparison of the evaluated schemes, indicating the key criteria each one satisfies.

No less important, we design and implement two different kinds of privacy-reserving solutions for tackling WebRTC IP leaks. The proposed solutions are implemented differently, but in essence, use the same detection technique. Namely, the first one is a browser extension, which basically uses a preload mechanism to prevent the corresponding JavaScript calls before the actual HTML Document Object Model (DOM) loading starts. The second uses a gateway, either local or third-party, to inspect the JavaScript code the user is about to download.

Particularly, this PhD thesis has contributed to publications in scientific journals and conference proceedings in the following three ways. These contributions correspond to and fulfill an equal number of thesis objectives.

- **Obj. 1:** Enhancing the level of anonymity provided by SIP-based VoIP services by utilizing PrivaSIP in conjunction with Tor anonymity network[1].

- **Obj. 2:** Designing and implementing an encryption-based privacy-preserving scheme in which all sensitive fields are encrypted using onion routing[2]. Specifically, this implementation comprises two variants:

  - Public-key encryption based on SIP proxy server certificate.
  - Symmetric key encryption.

- **Obj. 3:** The provision of an effective means by which a user can be notified if a webpage attempts to perform WebRTC requests without their consent[3]. Two different methods were presented:

  - A browser extension, which uses preload mechanisms to prevent such JavaScript calls before the actual DOM loading starts.
  - A custom gateway that inspects the JavaScript code of the webpage the user is about to download.

[1]Karopoulos, G., Fakis, A., & Kambourakis, G. (2014). Complete SIP message obfuscation: PrivaSIP over Tor. Proc. of the 9th International Conference on Availability, Reliability and Security (ARES), pp. 217-226, IEEE. https://doi.org/10.1109/ares.2014.36

[2]Fakis, A., Karopoulos, G., & Kambourakis, G. (2017). OnionSIP: Preserving Privacy in SIP with Onion Routing. J. Univers. Comput. Sci., 23(10), 969-991. https://doi.org/10.3217/jucs-023-10-0969

[3]Fakis, A.; Karopoulos, G.; Kambourakis, G. Neither Denied nor Exposed: Fixing WebRTC Privacy Leaks. Future Internet 2020, 12, 92. https://doi.org/10.3390/fi12050092

1.4 Thesis structure

The next chapter briefly presents a background on the communications technologies that are relevant to the present PhD thesis. It specifically details SIP and WebRTC protocols, alongside their respective vulnerabilities regarding privacy and anonymity.

Chapter 3 offers a literature review, emphasizing on four main axes. The first concerns existing solutions to protect user's privacy in SIP protocol. The second pertains to schemes that could be used to anonymize SIP traffic in the transport layer. The third axis outlines a range of suggested remedies, which users can exploit to safeguard their anonymity whilst utilizing WebRTC. The final axis centers on ways to deanonymize Tor traffic.

The proposed SIP-over-Tor scheme is presented and assessed in Chapter 4. Specifically, this chapter details the performance of both plain SIP and PrivaSIP-over-Tor.

A novel approach to safeguard users' privacy and anonymity in the SIP protocol is given in Chapter 5. This includes the implementation of an onion architecture, similar to that utilized by Tor, which conceals the user's IP address and simultaneously encrypts any sensitive data contained in SIP messages.

Chapter 6 addresses WebRTC privacy leaks that can expose the user's IP address while they visit a potentially malicious webpage. Two distinct solutions are presented: a browser plugin and a gateway. Both these options are capable of examining HTTP traffic and detect and block suspicious Javascript pieces of code.

The last chapter completes this PhD thesis by concluding the results of the conducted research. Directions for future work are also presented at the end of the same chapter.

Chapter 2

Background

2.1 VoIP services

The immense growth of ICT technologies and the mushrooming of high-speed wireless networks, led to a drastic shift in the ways people communicate. The traditional methods of telephone communication and face-to-face meetings are no longer as efficient or convenient as they used to be. Nowadays, the use of telecommunications technology is becoming increasingly prevalent in everyday tasks, such as communicating and conducting business. This has resulted in a massive increase in demand for high-quality, always-available telecommunications services. Such a key service is VoIP, which allows people to communicate using the Internet instead of traditional landlines. Without a doubt, modern VoIP technologies offer a number of advantages over traditional forms of telecommunication. Users can access the service from anywhere with an Internet connection and receive improved audio quality at a lower cost than standard telephone services. However, when it comes to security and privacy, as with any popular Internet technology or protocol, VoIP services are susceptible to cyberattacks [11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21], which can affect its Quality of Service (QoS) and compromise end-users' privacy.

2.2 SIP protocol

SIP is a signaling protocol defined in [22] for initiating, maintaining, updating, and terminating real-time sessions involving video, phone, message, and other multimedia applications and services between two or more Internet endpoints. Similar to HTTP, SIP is a request-response protocol that relies on text-based communications and proxy servers to establish, manage, and terminate sessions between two or more peers.

Various communication systems use SIP to administer VoIP sessions. SIP inherently supports different communication modes, such as audio, video, text, and multimedia, and can be used across several transport protocols, including UDP and TCP [22].

SIP relies on protocols such as Real-Time Transport Protocol (RTP) [23] and Secure Real-Time Transport Protocol (SRTP) [24] to transport media streams between endpoints [25]. The media and communication parameters of a session are typically described through a session description protocol, such as Session Description Protocol (SDP) [26].

To relay SIP requests from one server or client to another, SIP proxies are commonly used as intermediary servers. Essential features of SIP gateways include message routing and the setup and maintenance of communication sessions. SIP proxies may further offer authentication, authorization, billing, and NAT traversal services [22].

Similar to HTTP, SIP does not use any message encryption by default. Therefore, if unencrypted, SIP messages can be easily intercepted and read by anyone who has access to the network. This means that the contents of the messages, including sensitive information such as call setup information, caller ID, and call content, can be accessed by unauthorized eavesdroppers. Possible malicious or honest-but-curious opponents include network administrators, Internet service providers (ISPs), and other parties who are able to access and sniff the network.

To prevent SIP message interception, it is critical to encrypt the messages' contents. A straightforward solution is the creation of a TLS tunnel, providing hop-by-hop security for communication between SIP clients and servers. However, even if TLS is implemented, there is no guarantee that encryption will be used in all the network hops, thus users' information hiding is not to be taken for granted.

Finally, the SIP protocol was designed to work with traditional telephony systems where anonymity is not an issue. As a result, it lacks by-design mechanisms for concealing users' identities, IP addresses, and other sensitive information. Therefore, among others, end-users may be vulnerable to eavesdropping, man-in-the-middle attacks, and denial-of-service (DoS) attacks. Overall, SIP users who wish anonymity protection should consider exploiting add-on security measures, including Virtual Private Networks (VPNs), Tor, or custom-built encryption schemes.

2.3 PrivaSIP

PrivaSIP [9, 10] is an application-layer privacy-preserving scheme, whose main idea is that each end-user's real ID shown in a SIP message should be individually encrypted in a way that it can be recovered only by authorized entities. In this way, untrusted proxies or malicious users performing traffic analysis, cannot eavesdrop or even recover from the corresponding ciphertext the real ID of the user. This way, a basic level of unlinkability [27] of SIP transactions made by the same user in the course of time can be retained as well. There exist two versions of PrivaSIP depending on level of privacy offered:

- PrivaSIP-1 [9]: It offers caller identity privacy
- PrivaSIP-2 [10]: It caters for both caller and callee identity privacy.

PrivaSIP-1, provides only identity privacy for the person making the call. This is achieved by encrypting the username of the caller and using the created ciphertext on every SIP message that will be exchanged between the caller and the callee. To avoid personal information leakage, the display name field on the SIP message is removed. On the other hand, PrivaSIP-2, provides identity privacy for both the caller and the callee. Precisely, in PrivaSIP-2, the caller's SIP application encrypts both caller's and callee's username, while removing the display names that appear in every SIP message. PrivaSIP also varies in the encryption methods it employs: a user can choose between symmetric or asymmetric cryptography. Specifically, different implementations have been presented, including symmetric, traditional asymmetric, and Elliptic Curve cryptography:

- Symmetric cryptographic algorithm
  - PrivaSIP-1 using AES
- Asymmetric cryptographic algorithms
  - PrivaSIP-1 or -2 using RSA

  - PrivaSIP-1 or -2 using Elliptic Curve (ECIES)

In PrivaSIP-AES, AES is utilized for the encryption of the caller ID. Since the caller and their home proxy share a password, which is used for digest authentication, this password can also be used as a key (or as a key seed or master key) for the encryption of the user's ID. PrivaSIP-RSA uses the caller and callee's proxy public keys to encrypt the caller's user ID and digest username, and the callee's user ID, respectively. PrivaSIP-ECIES works similarly with the difference of using Elliptic Curve cryptography.

The cost that comes along with PrivaSIP is negligible concerning the privacy features offered, when symmetric cryptography is used. Nevertheless, as argued in [9, 10], a user may perceive an affordable latency ranging between 0.5 to 2 sec for the first SIP message when asymmetric cryptography is chosen, depending on the server load. In subsequent SIP messages, this time penalty can be minimized, provided that some caching method is used for the correspondence between real and encrypted IDs. In any case, the aforementioned latency occurs only during the setup phase of the call. For more details regarding the PrivaSIP schemes, the reader is referred to [9, 10].

2.4 Tor anonymity network

Nowadays, Tor [28] is probably the most popular open-source implementation of a distributed overlay network, which aims to anonymize TCP-based traffic. The Tor software, namely Onion Proxy (OP), at the user side chooses a random path through the available Onion Routers (OR) in the network and builds a circuit, in which each OR in the path knows only its predecessor and successor, but no other nodes. After that, the user's traffic flows down the selected circuit using onion routing, which is based on layered encryption, also known as "telescopic encryption". To further block analysis of data based on the traffic characteristics, the Tor overlay uses cells of fixed size to communicate users' data among the different ORs. Overall, as already pointed out, this approach blocks traffic analysis because no entity is aware of the full path a packet (cell) has traveled.

There are three types of relays in the Tor network: Directory Authorities (DA), Exit Relays (ER), and Guard Relays (GR). Each of them is responsible for a specific part of the network. A DA provides current information about other relays on the network.

ER allow users' traffic to exit the Tor overlay by forwarding their traffic to its final destination. Finally, GRs provide security and integrity to the rest of the network by preventing hostile third parties from gaining access to it [29].

To build a circuit, the OP retrieves the list of available ORs from the DA and randomly selects a set of ORs forming the circuit between the sender and destination node. The OP establishes an ephemeral session key with each OR in the circuit by means of a Diffie-Hellman handshake. After that, the sender can initiate anonymous communications with the receiver. All information entering a circuit is in fixed-size cells (data-packets) of 512 bytes. Tor mandates all the cells transmitted from an OP toward the receiver to be repeatedly enciphered under the session keys shared with ORs consisting the circuit. This way, no individual relay is aware of the complete path a given cell has followed. The OR lying at the edge of the circuit will eventually remove the last layer of encryption, obtaining the original data. Furthermore, in order to safeguard the transmission of data between each node in the circuit, Tor employs a TLS tunnel. This ensures that observers cannot intercept and examine the contents of the tunnel to identify the destination circuit of a given cell. The TLS tunnel is established between each pair of adjacent ORs in the circuit. This trasport layer tunnel ensures that any piece of data transmitted between ORs is protected and cannot be read by an attacker.

Overall, Tor capitalizes on the onion routing mechanism to deliver a low-latency Internet networking service designed to anonymize the data relayed through it. As a result, Tor is also employed as a powerful weapon against Internet censorship carried out by governments or by private organizations on behalf of others. BitTorrent and HTTP are well-known to use Tor services to enhance their level of anonymity. Tor traffic passes through a large number (more than 6K) of relays that are distributed globally and operated by volunteers [30]. Even though Tor is considered one of the most effective tools for achieving online privacy, it also has a number of limitations that make its use more cumbersome than traditional web services. For example, the Tor network is typically quite slow and requires significant bandwidth in order to function effectively. This makes it difficult to download large files and use the system for streaming content. However, these limitations are relatively minor when compared to the benefits in terms of anonymity. Altogether, Tor provides a useful way for accessing the Internet without the fear of being monitored or tracked by third parties.

2.5 WebRTC

Real-Time Communication (RTC) refers to the technology that enables individuals to communicate and exchange information in real-time over the internet without requiring physical proximity. This technology has become increasingly important in recent years as individuals and businesses rely on online communication for work, education, and social interaction among others. RTC encompasses a variety of communication modalities, including video and voice chat, text messaging, and online gaming.

Even more, the significant popularity of the browser platform, which was not originally intended for interactive RTCs, has resulted in the emergence of browser-based applications that utilize real-time communication on the web [31]. The resulting WebRTC [5] technology is a relatively new, open-source project that enables two-way communication between browsers and other applications. Specifically, WebRTC has gained significant popularity as a technology for real-time communication, with increasing adoption and support from industry leaders including Google and Microsoft. Its usage and acceptance continue to expand, reflecting its growing prominence in the field of real-time communication. In 2019, the worldwide market size for WebRTC was valued at USD 2.3 billion. It is projected to exhibit a compound annual growth rate of 43.4% from 2020 to 2027 [32].

WebRTC uses a signaling protocol to initiate and manage calls, and a communication channel to carry audio, video, or text data between the endpoints. The latter can be legacy desktop computers, mobile devices, or even cloud-based Web applications. Moreover, WebRTC uses a combination of JavaScript APIs and communication protocols such as RTP and SRTP to establish P2P communication channels between browsers without requiring any additional plugins or software installations. This allows developers to easily add real-time communication capabilities to their web applications, enabling seamless communication between users across different devices and platforms.

Despite its popularity, WebRTC has a potential security issue that could compromise the privacy of its users. Specifically, WebRTC can expose the IP address of the user, which can reveal sensitive information about their location and online activities. This happens when a website forces a user to use the ICE protocol, which is a key component of WebRTC. Namely, the ICE protocol is used to establish a connection between two devices, involving exchanging information about network addresses. Unfortunately, it is obvious that this information can be used to identify a user's IP address, even if they are behind a VPN or other privacy-enhancing technology.

2.6 Discussion

Taking all the above into account, we conclude that SIP, despite being the commonest signaling protocol, it lacks security protections, leaving user's sensitive information unprotected. PrivaSIP is a protocol that effectively addresses the major privacy gaps present in SIP. However, it is worth noting that the absence of cross-layer anonymity mechanisms remains a significant concern. As such, it is essential to develop a comprehensive solution to protect the confidentiality of user information during the initiation of SIP sessions and call control operations. In reference to this matter, it is imperative that the development of a solution should prioritize two distinct aspects: (i) obfuscating sensitive SIP data and selectively making it available only to trusted entities, and (b) providing anonymity in a cross-layer fashion.

The utilization of the Tor network may constitute a viable and concrete solution for addressing the privacy gap of SIP, augmenting user anonymity. That is, integrating Tor with PrivaSIP could potentially result in complete message obfuscation, thereby offering privacy protection in both application and transport layers. Nevertheless, it should be emphasized that the integration of Tor and PrivaSIP poses technical complexities and may necessitate additional resources to ensure seamless operation. It is also important to conduct a careful assessment of the potential trade-offs between security, privacy, and performance while deploying this solution. Despite these challenges, the employment of the Tor network could represent a promising avenue for enhancing the privacy and anonymity of SIP users.

Equally important, WebRTC is a widely adopted, modern technology that facilitates real-time communication among users. Nevertheless, it is key that users are made aware of the potential privacy risks associated with this technology, which may leak their actual IP address to an interested party. As detailed in Chapter 3, to the best of our knowledge, the existing literature provides no practical solution to safeguard one's privacy other than completely disabling WebRTC. Consequently, a more practical and potentially transparent to the user solution to this problem is needed.

Chapter 3

Related work

This chapter offers a comprehensive survey regarding VoIP privacy and anonymity. Specifically, it first details works pertaining to anonymity issues in the SIP protocol in Section 3.1, while contributions referring to WebRTC technology privacy leaks are discussed in Section 3.2. Additionally, Section 3.3 overviews Tor deanonymization issues.

3.1 Protecting User Information in SIP Networks

This section briefly reports on works that have been presented in the literature so far regarding SIP privacy. It also presents related work on several solutions that could potentially be used for anonymizing SIP traffic. As explained further down, despite the various proposals, most of them are more or less unsuited to SIP.

There is one group of mechanisms based on PGPfone and its successor, Zfone. Zfone [33] is a software for secure VoIP communications, using the ZRTP protocol [34]. Zfone works on top of existing SIP and RTP protocols and the security it offers is limited to encrypting and decrypting voice packets only. Moreover, it seems that since 2009 its development has stopped.

Silent Circle [35] is a company offering secure communications to its subscribers. In order to protect voice and video call data on cellular and Wi-Fi networks from interception and eavesdropping, the company developed a private communication infrastructure called

Silent Network. The connection between the caller and the Silent Network, as well as the connection between the Silent Network and the callee, are both encrypted provided that both users are subscribers of the service. If one of the two users is not a subscriber, then the respective connection is unencrypted. Its operation is based on the ZRTP protocol, mostly protecting from Man-in-the-Middle (MiTM) attacks without taking care of end-users' identity privacy.

RedPhone [36] uses a signaling protocol that is custom to RedPhone and the voice traffic is encrypted using ZRTP. Its signaling protocol is similar to HTTP and protected by means of a TLS tunnel. However, since TLS is used between clients and relay servers, the communicating parties are indeed identifiable by their IP addresses. Jitsi [37], formerly known as SIP Communicator, supports many protocols including SIP, Jabber/XMPP (GoogleTalk and Facebook), MSN, ICQ, etc., and offers call encryption to SIP and those protocols that are based on Extensible Messaging and Presence Protocol (XMPP) through ZRTP. For SIP, secure signaling with the use of TLS is also supported. Nevertheless, the same problem still stands; the communicating end-users are identifiable by their IP addresses. The Open {Secure, Source, Standards} Telephony Network (OSTN) project [38] is foreseen to provide an end-to-end secure VoIP service based on open standards. SIP signaling is protected with Secure Socket Layer (SSL), while voice traffic is secured by ZRTP. As with other solutions, the issue with this system is that end-users are left protected against traffic analysis attacks.

There is also a group of solutions that are based on proprietary protocols and/or are unsupported. These include Speak Freely [39] (since 2002) and the "I Hear U" (IHU) project [40] (since 2008). GSMK CryptoPhone [41] provides a range of hardware and software products including mobile, landline, and satellite phones, softphones, and even a crypto PBX. It is a proprietary solution and the source code is offered for download; however, it seems outdated since the last version is from 2003, and no safe claims can be made about its secure operation. Mumble [42] is an open-source, voice chat software mainly intended for use while gaming. Communication to and from the server is protected through encryption, which is mandatory and cannot be disabled. The voice as well as the control channel, which transports chat messages and other non-time critical information, are both encrypted. The latter is protected by TLS, making IP address discovery of end-users possible.

Nautilus Secure Phone [43] uses the Diffie-Hellman key agreement, followed by a verbal comparison of the hashed value of the agreed session key, to prevent a "man in the middle" attack from compromising the system. Three voice encryption algorithms, namely Triple-DES, Blowfish, and IDEA, are supported.

Regarding the protection of end-user's anonymity, most anonymity networks target diverse needs, so existing systems are implemented with a specific purpose in mind. First and foremost, the majority of the considered anonymity systems or tools rely on Tor or other widespread networks to achieve anonymity, wrapped and foisted with some minor changes. Other systems, like iMule [44], StealthNet [45] or Mixmaster [46], are used for different purposes such as sending emails or exchange files anonymously, without giving the choice of proxy for a third app like a SIP client.

There is also a plethora of networks proposed in the bibliography that promise anonymity, but they are on an early or prototype stage, such as Tarzan [47], MorphMix [48], and Phantom protocol [49]. Hornet [50], which name is an acronym for High-speed Onion Routing at the Network Layer, is yet another anonymity network, similar to Tor. Their creators claim that Hornet will be able to reach speeds exceeding Tor network. However, there is still, by the time this PhD thesis was written, no tangible implementation of Hornet, and thus there is no way to test it. Last but not least, in 2016 MIT researchers presented a new onion-routing based anonymity scheme called Riffle [51]. The authors claim that Riffle network affords robust security and anonymity, utilizing bandwidth with superior efficiency compared to alternative anonymity networks, all while employing security methodologies akin to Tor. Nevertheless, the research prototype currently showcased is not yet prepared for testing, as it is solely operational in two distinct modes: filesharing and microblogging. In the latter, a client can transmit a message that will be disseminated to other users within the network. Consequently, although Riffle holds promise, our ability to evaluate it with SIP was impeded.

Other researchers used various approaches to anonymize or preserve privacy in VoIP systems. In [52] the authors proposed a premature effort on using Tor to anonymize SIP traffic. They succeeded in providing full SIP message obfuscation by adapting an application-layer, encryption framework, called PrivaSIP [10, 9]. Using PrivaSIP over Tor, they were able to encrypt the sensitive contents of SIP messages, including "From" and "To" header fields, thus preventing unauthorized users from reading these fields. As also explained in Section 4, the combination of PrivaSIP with Tor can be considered a complete SIP privacy solution, as it achieves both application and network layer privacy.

In [53], the authors proposed a scheme which is resistant against two attacks (complementary matching and watermark) that a malicious user can perform to discover "who calls whom", even though traffic is proxyfied through an Anonymization Overlay Network (AON). The authors claim that they devised a way to prevent malicious actors from performing such attacks, by applying Voice Activity Detection in every call. That is, the originator's user-agent can produce voice packets in a constant rate and instruct the AON proxies to drop those packets, hindering the attacker to disclose users identities. Nevertheless, as the authors mention, the above defense is not a comprehensive solution that can provide highly usable, efficient, and practical anonymity for all VoIP users.

In [54], the author details the different aspects of anonymity and accountability in the SIP protocol. Moreover, he presents the two major categories that privacy mechanisms are divided into, the cryptographic-based solutions, like SIP over TLS and SIP using S/MIME, and the non-cryptographic ones, which anonymize the SIP URI. Based on this separation, a discussion is made about the custom solutions, plus the generic and lower layer ones. The protection of SIP using S/MIME is documented extensively in RFC 3853 [55]. Each message is treated like an email attachment and so it is encrypted, using an algorithm like AES, and signed via S/MIME. Using such a methodology, all the security services that S/MIME provides can be added to SIP, including authentication, message integrity, non-repudiation of origin, privacy, and data security. However, in S/MIME, some of the header fields of the message must always remain in plaintext, as they are necessary for successful message delivery. More specifically, the "To", "From", "Call-ID", and "Contact" fields are still unprotected.

Herd [56] is a scalable, traffic analysis resistant anonymity network designed for VoIP systems. The authors claim that Herd provides user anonymity by forwarding traffic through cloud-based proxies that can be considered as low-delay circuits. Herd uses a hop-by-hop and layered encryption over a circuit of nodes, using Datagram TLS (DTLS) [57] to accomplish lower latency than other similar anonymity networks. The proposed solution is claimed by the authors to be quite efficient with lower bandwidth requirements, however, it achieves that through a dedicated infrastructure, incurring additional costs and delays.

In [58], the authors highlight the problem arising from the metadata contained in Call Detail Records (CDRs), which contain sensitive information such as source, destination, start time, and duration of a call. They propose Phonion, an architecture where every call is routed over the telephony infrastructure. The authors claim that Phonion can achieve high quality calls while obfuscating call data records. However, Phonion only protects the CDRs and not the actual contents of the VoIP messages, therefore, a call is still vulnerable to traffic analysis attacks.

Finally, a product that offers anonymity in VoIP is TORFone [59], which is a product that is independent of the Tor Project. It is based on Tor and PGPfone, and since the connection between users passes through six transit nodes located anywhere in the world, the voice latency can reach 4 to 5 secs. Apart from that, the main issue with TORFone is that it is based on outdated security software (PGPfone is unmaintained since 1999) and it does not use standard protocols, like SIP.

3.2 WebRTC privacy leaks

At the time of writing this PhD thesis, the WebRTC API privacy issue has been explored by a handful of works in the literature. The work in [60] details the privacy threat occurring when WebRTC is exploited by browsers. The authors conduct different experiments in order to examine which types of IP addresses can get compromised. To this end, they tested a variety of widely used VPN services on different browsers and presented the results of their experiments, concluding that none of the tested VPN services has proven to be a perfect privacy solution. From the results offered by the authors, one can realize that each VPN service exposes a different set of user's IPs, with TorGuard being the preferred privacy-preserving VPN service, exposing just the private IPv4 of the user assigned by the VPN server. However, based on their results, a VPN solution is neither a silver bullet to the IP leaking issue, nor efficient, as VPN networks are highly dependent on the bandwidth of their VPN server, which are often geographically dispersed.

The issue of using WebRTC for mapping the intranet topology from an external attacker is discussed in [61]. The authors present a JavaScript piece of code, namely Malice-Script, which collects intranet information abusing the features of WebRTC, and then infiltrates a targeted website from its own intranet. This work concentrates on the attack implementation and only provides recommendations for end-user protection. One of the defensive measures is to notify the user of the browser and ask for explicit authorization before calling the WebRTC API, which is actually what our implementations in Chapter 6 provide.

Among other security and privacy risks, the authors in [62] concentrate on that of WebRTC. Specifically, they were able to perform a network scan on victim networks and collect sensitive network information about other internal nodes, such as open ports, based on heuristics like round-trip delays. They also claim that they developed a browser extension that can warn users about possible malicious activities, when a certain number of connections made to the private IP address of a user is surpassed. However, the countermeasure they propose for the specific privacy issue is just disabling WebRTC or using other extensions that can expose only the user's public IP.

Additionally, the work in [63] capitalizes on WebRTC in an attempt to disclose the user's private IP and execute network scans using an open source JavaScript scanner, called jslanscanner [64]. The network scan is supposed to detect not only possible online local network nodes, but routers as well. The authors extend the way a malicious party can exploit WebRTC to expose a user's IP; nonetheless, they omit proposing a solution to safeguard against these kinds of attacks.

A Browser scanner is proposed in [65] where an online service uses an XMLHttpRequest and WebRTC for the purpose of collecting information about the network a user resides in. The authors present the results they were able to gather from the scanned networks; however, they suggest that JavaScipt and WebRTC should be disabled for avoiding such scans by malicious websites.

In [66], the use of mDNS [67] is proposed for protecting private IP addresses. In the proposed mechanism, the web browser registers an ephemeral mDNS name for the private IP address of the host and then provides this name in its ICE candidates (explained in Chapter 1) instead of the IP address. The ICE candidates negotiation can proceed as usual, and the mDNS names can be resolved to IP addresses only when necessary and without revealing the actual address to the other endpoint. The main issue with this approach is that it can break WebRTC applications. There are various reasons for that: (a) mDNS is built on the assumption that the names never leave the local domain, (b) the SDP protocol expects IP addresses instead of DNS names, and (c) if a DNS server tries to resolve an mDNS name it will fail as no relevant resource record exists.

WebRTC's IP leak issue is also addressed by some browser add-ons for content-filtering, including ad-blocking. Two of them, namely Privacy Badger [68] and uBlock Origin [69], both multi-platform and open-source, seem to stand out. However, while these extensions prevent WebRTC from leaking the peer machine's private IP address by blocking it, they do not solve the problem of public IP leakage happening through STUN requests.

There is also a mass of works which utilize the WebRTC leak in order to fulfil other purposes. In [70, 71, 72], WebRTC is used to collect private IP addresses for user tracking and device fingerprinting. The authors in [73] utilize WebRTC for device fingerprinting with the aim of tracing web attackers even when they are hidden behind VPN or anonymous networks. WebRTC is also exploited in [74] for device fingerprinting for improving user authentication. In all the previous cases, no countermeasures against the WebRTC leak have been implemented.

It is worth noting here that, as explained in [75] the signaling protocol between web servers is not part of WebRTC. This means that, even if a competent solution that protects end-user's privacy is employed on WebRTC, personal user information can leak from the signaling protocol [54]. On the application level, this can be prevented with solutions, including [9, 10] when SIP is utilized as the signaling protocol. On the network level, Tor-based solutions ([4, 52]) can prevent leaking IP addresses to unauthorized entities.

Overall, while the given vulnerability of WebRTC is addressed by a number of works in the literature, most of them either use this vulnerability to fingerprint clients inside a network or propose countermeasures that partially protect user privacy, such as the use of a VPN or a combination of browser extensions. In addition, some works propose the complete deactivation of WebRTC on the browser, which would obligate the user to re-enable it every time they need it. To the best of our knowledge, the work detailed in Chapter 6 is the first to propose two different practical and effective schemes which can detect the use of WebRTC by a webpage in time, i.e., before any WebRTC requests are made. Namely, the proposed solutions can inform the user on-the-fly about whether their privacy is at risk by WebRTC when they try to visit a webpage. Thus, if the user is really in need of WebRTC capabilities, then the webpage is allowed to load; otherwise, the user can choose to continue by disabling the corresponding JavaScript code.

3.3 Tor network deanonymization

Based on the literature, the most powerful deanonymization attacks on Tor are based on some scheme that renders possible the correlation of the data between two points, preferably the Entry and Exit node, of the Tor overlay. This strategy is also followed by "SnoopyBot" [76], which is detailed in Section 3.3.1.

Undoubtedly, the weakest node in a Tor circuit is the Exit node. This is because at this node the actual data sent by the Tor user is decrypted and forwarded toward its final destination. An Exit node can behave maliciously in a variety of ways, having a key role in the identification of a Tor user. For instance, as described in [77], the Exit node can inject a piece of software that will be executed by the victim's browser. Another approach that makes use of a malicious Exit node was given by the authors in [78]. That is, they either injected a forged web page back to the Entry node or they modified the HTTP traffic passing through it, mounting a web page modification attack. Other contributions like the one described by [79] focus on timing attacks done by a malicious Entry node. Similar to the previous one, the method proposed by [80] places the attacker at both a malicious entry and exit OR. Specifically, the attacker manipulates the outbound cells originating from the various OPs, causing them to be determined at the exit node, and thus eventually confirming the communication relationship between the sender and the receiver.

A second category of deanonymization methods capitalizes on web browsers with enabled Javascript, Flash, or similar technologies. Although these features are designed to enhance user's experience when browsing, they are potentially privacy-invasive, as described by [81]. An attack taking advantage of this fact can use a malicious Exit node for modifying web pages through the injection of JavaScript code that repeatedly connects to a logger server sending a distinctive signal.

Overall, the majority of the attacks in the literature are mainly based on infected Entry or Exit nodes, considering that even security-savvy users cannot detect any untrustworthy node in the Tor Network. Naturally, this capacity enables attackers to stelathly eavesdrop on traffic produced by any torified network application, including multimedia ones. To delve into this situation, the rest of this section details "SnoopyBot" [76] spyware, which demonstrates that the interlinking of Tor endpoints are feasible, given certain limitations as discussed in section 3.3.2[1]. Nevertheless, the way SnoopyBot operates differs from the related work in two aspects. First, the manipulation of data is performed at the client before the user's data reaches the OP. Secondly, it does not require the attacker to possess special equipment or a large mass of computing resources. No less important, its operation at both ends, i.e., the client and the Exit node, is as stealthy as possible.

3.3.1 Implementation

The architecture of the authors' implementation [76] consists of two main components. The first one, is a user's side spyware, named SnoopyBot, which is essentially a malicious Android application that must be installed on user's smart device. SnoopyBot was developed for Android v4.4 and has also been successfully tested on Android v5.1.1. The second component of authors' architecture is at least one specially configured Tor Exit node that sniffs user traffic passing through it. When the Exit node receives data having the unique signature of SnoopyBot, it instantly logs and deanonymizes it. Specifically, SnoopyBot was developed and used on the test smartphone, and the only traffic that was logged on the Exit node was solely the one stemming from that smartphone.

At a high level, assuming an HTTP or HTTPS connection, SnoopyBot has three main goals. First, it modifies the default settings of any Tor application running on the smart device and are necessary to access Tor network. These applications are, Orbot and Orweb. Second, it obtains the public IP of the user, and third, it hijacks (acting as a man-in-the-middle) the connection in order to inject the user's personal information to the requested URL.

SnoopyBot is designed to instantly launch right after its installation, as well as after every reboot of the smart device. After its successful installation, SnoopyBot masquerades itself as Adobe Flash v2.0, which is a well-known software and will normally present itself as a benign application to the owner of the smart device. Moreover, it removes any application (SuperSU, Superuser) that gives root permissions to the user, including pop-up permission request dialogs. Pop-up dialogs are shown to the user by means of

[1]The article [76] is not considered part of the core contributions of this PhD thesis.

toast messages, every time an application requests root permissions. This kind of action can attract the victim's attention, and such a suspicion could eventually lead to have SnoopyBot uninstalled. Besides, the authors particularly concentrated on the stealthy operation of SnoopyBot to make harder its detection by anti-spyware software. Towards this goal, the authors minimized the number of connections SnoopyBot makes outside the Tor network. Precisely, SnoopyBot generates only one request to a public web service, to obtain the user's public IP, while the rest of the communication remains within the Tor network. Naturally, every time the IP of the user changes, the same request must be repeated.

As already pointed out, the spyware needs to modify Tor settings at the client side. Figure 3.1A depicts a typical HTTP or HTTPS GET transaction when processed by Tor at the client side. The spyware invades this procedure and modifies Orbot configuration settings inside the torrc file, to always use the attacker's exit node during the creation of any Tor circuit. After that, it prohibits any further modifications to torrc. Note that Orbot consists of two main components: the Polipo [82] HTTP proxy that listens on port 8118, and the Privoxy SOCKS proxy [83] that listens on port 9050. Essentially, Polipo forwards the HTTP traffic to Privoxy, which subsequently forwards the traffic to Tor. Additionally, as depicted in Figure 3.1B, SnoopyBot alters the HTTP proxy settings of Orweb to point to destination port 8119 instead of the default 8118. This forces all the victim's traffic to be proxied through the SnoopyBot HTTP proxy. After manipulating the GET request, SnoopyBot forwards it to HTTP port 8118 as normal, which is Orbot's HTTP Port. Simply put, SnoopyBot HTTP proxy acts as a MiTM between Orweb and Orbot, therefore it can eavesdrop on and modify any data passing through this network link. Next, SnoopyBot triggers a connection to a public service to obtain the user's (victim's) public IP. Since this is the only outbound connection made by the spyware, its footprint on the system is minimal. Having the public IP of the victim, the SnoopyBot proxy injects it along with a certain identification string (i.e., SnoopyBot's signature) into all URLs requested by the victim.

For example, assuming that the requested URL is http://www.example.com before injection, it will be modified to http://www.example.com/&sourceip = 65.65.65.65 and be forwarded to Orbot. The SnoopyBot signature's in this case is the "&sourceip" string. Naturally, this string is entirely up to the spyware coder and can be changed between the different versions of the spyware. As illustrated in fig. 3.2, this signature will be

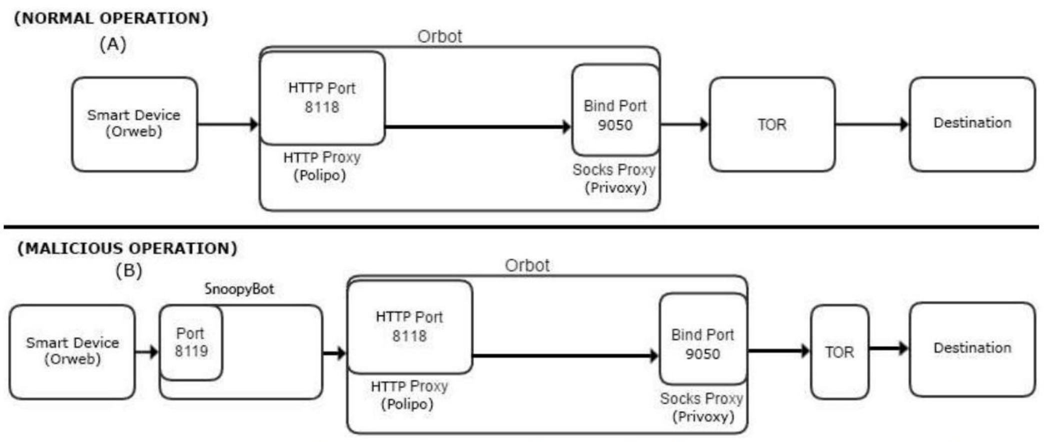


FIGURE 3.1: Tor operation at the client side: (a) Normal, (b) After infection by SnoopyBot

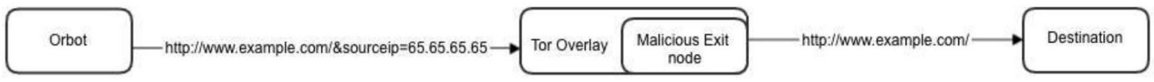


FIGURE 3.2: High-level description of the attack for an HTTP request

removed at the Exit node. Moreover, as shown in fig. 3.3, in case an HTTPS request is detected, SnoopyBot momentarily blocks it and then constructs and forwards an identical HTTP one, injected with the victim's public IP as normal. At nearly the same time, it forwards the original HTTPS request to Orbot to be dispatched via the Tor overlay. This time, however, SnoopyBot's signature in the HTTP bogus request is changed to "&sourceiphttps". As explained in section 3.3.1.1, this alteration will enable the Exit node to log and then discard the corresponding GET message. Several scans using popular mobile antivirus applications (including Kasperksy, Bitdefender and CleanMaster) did not detect SnoopyBot.

3.3.1.1 Exit node

For the exit node, the authors used the official Tor software on a Linux Ubuntu server, and configured the torre file so as for the authors' server to operate as a trusted Exit

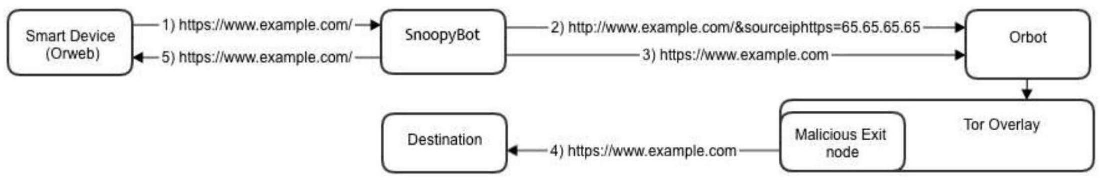
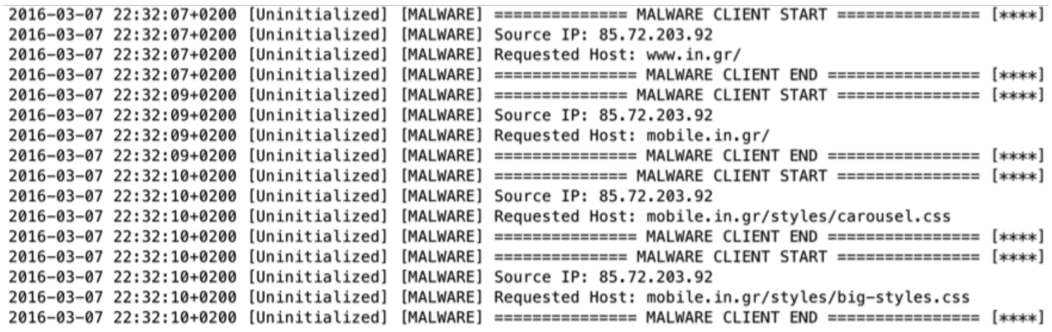

Smart Device (Orweb)
1) https://www.example.com/
5) https://www.example.com/
SnoopyBot
2) http://www.example.com/&sourceiphttps=65.65.65.65
3) https://www.example.com
Orbot
Tor Overlay
Malicious Exit node
4) https://www.example.com
Destination

FIGURE 3.3: High-level description of the attack for an HTTPS request (basic steps)

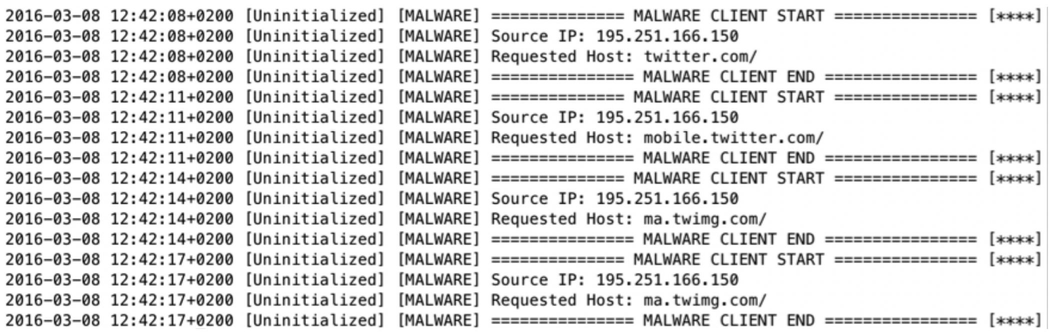
node. Particularly, they configured the torrc file in the Exit node to route HTTP and HTTPS traffic (ExitPolicy accept $*$: 80 and ExitPolicy accept $*$: 443). The authors adjusted the amount of bandwidth that will be made available to Tor, and provided a name for the Exit node.

Moreover, a Python HTTP proxy was implemented. This proxy listens on a different port on the same box as Tor and eavesdrops on incoming traffic. By using iptables, the authors redirected all the incoming traffic with destination port 80 or 8080 (http-alt) to HTTP proxy owned by the authors. As observed from fig. 3.3, upon the reception of any HTTP request that carries one of the SnoopyBot's signatures, the HTTP proxy logs it, strips the signature from the request, and forwards the request to its destination, e.g., a webserver as normal. An exception to this rule is any incoming bogus HTTP request, which is blocked in order for the destination not to receive an extra HTTP request corresponding to the original HTTPS one. As explained in section 3.3.1, all bogus HTTP requests carry a special SnoopyBot's signature. The HTTP response from the webserver is forwarded to the Tor network with no further manipulation. As shown in fig. 3.4 and 3.5, a bash script was created to filter the logged traffic and present the results containing the victim's source IP to the attacker.



```
2016-03-07 22:32:07+0200 [Uninitialized] [MALWARE] ============== MALWARE CLIENT START =============== [****]
2016-03-07 22:32:07+0200 [Uninitialized] [MALWARE] Source IP: 85.72.203.92
2016-03-07 22:32:07+0200 [Uninitialized] [MALWARE] Requested Host: www.in.gr/
2016-03-07 22:32:07+0200 [Uninitialized] [MALWARE] =============== MALWARE CLIENT END ================ [****]
2016-03-07 22:32:09+0200 [Uninitialized] [MALWARE] ============== MALWARE CLIENT START =============== [****]
2016-03-07 22:32:09+0200 [Uninitialized] [MALWARE] Source IP: 85.72.203.92
2016-03-07 22:32:09+0200 [Uninitialized] [MALWARE] Requested Host: mobile.in.gr/
2016-03-07 22:32:09+0200 [Uninitialized] [MALWARE] =============== MALWARE CLIENT END ================ [****]
2016-03-07 22:32:10+0200 [Uninitialized] [MALWARE] ============== MALWARE CLIENT START =============== [****]
2016-03-07 22:32:10+0200 [Uninitialized] [MALWARE] Source IP: 85.72.203.92
2016-03-07 22:32:10+0200 [Uninitialized] [MALWARE] Requested Host: mobile.in.gr/styles/carousel.css
2016-03-07 22:32:10+0200 [Uninitialized] [MALWARE] =============== MALWARE CLIENT END ================ [****]
2016-03-07 22:32:10+0200 [Uninitialized] [MALWARE] ============== MALWARE CLIENT START =============== [****]
2016-03-07 22:32:10+0200 [Uninitialized] [MALWARE] Source IP: 85.72.203.92
2016-03-07 22:32:10+0200 [Uninitialized] [MALWARE] Requested Host: mobile.in.gr/styles/big-styles.css
2016-03-07 22:32:10+0200 [Uninitialized] [MALWARE] =============== MALWARE CLIENT END ================ [****]
```

FIGURE 3.4: Filtered log file records at the Exit node for the (HTTP traffic)



```
2016-03-08 12:42:08+0200 [Uninitialized] [MALWARE] ============== MALWARE CLIENT START =============== [****]
2016-03-08 12:42:08+0200 [Uninitialized] [MALWARE] Source IP: 195.251.166.150
2016-03-08 12:42:08+0200 [Uninitialized] [MALWARE] Requested Host: twitter.com/
2016-03-08 12:42:08+0200 [Uninitialized] [MALWARE] =============== MALWARE CLIENT END ================ [****]
2016-03-08 12:42:11+0200 [Uninitialized] [MALWARE] ============== MALWARE CLIENT START =============== [****]
2016-03-08 12:42:11+0200 [Uninitialized] [MALWARE] Source IP: 195.251.166.150
2016-03-08 12:42:11+0200 [Uninitialized] [MALWARE] Requested Host: mobile.twitter.com/
2016-03-08 12:42:11+0200 [Uninitialized] [MALWARE] =============== MALWARE CLIENT END ================ [****]
2016-03-08 12:42:14+0200 [Uninitialized] [MALWARE] ============== MALWARE CLIENT START =============== [****]
2016-03-08 12:42:14+0200 [Uninitialized] [MALWARE] Source IP: 195.251.166.150
2016-03-08 12:42:14+0200 [Uninitialized] [MALWARE] Requested Host: ma.twimg.com/
2016-03-08 12:42:14+0200 [Uninitialized] [MALWARE] =============== MALWARE CLIENT END ================ [****]
2016-03-08 12:42:17+0200 [Uninitialized] [MALWARE] ============== MALWARE CLIENT START =============== [****]
2016-03-08 12:42:17+0200 [Uninitialized] [MALWARE] Source IP: 195.251.166.150
2016-03-08 12:42:17+0200 [Uninitialized] [MALWARE] Requested Host: ma.twimg.com/
2016-03-08 12:42:17+0200 [Uninitialized] [MALWARE] =============== MALWARE CLIENT END ================ [****]
```

FIGURE 3.5: Filtered log file records at the Exit node (HTTPS traffic)

3.3.2 Limitations and Countermeasures

The current version of SnoopyBot presents the following three limitations.

- SnoopyBot needs to somehow infect and spread amongst users. One way to do so is to bundle the spyware apk setup file with another apk, belonging to an app that requires root permissions during installation. The latter apk may belong to a legitimate app that the user would download from an alternative Android app market. Using this method, during apk installation, the root permissions that the legitimate app would ask from the user for performing its tasks would also be given to the spyware after installation. Another way to propagate SnoopyBot is to make an on-the-fly bundle and inject the spyware when a user makes a request to the Exit node controlled by the attacker for an .apk download. Nevertheless, this method augments the chances of the malevolent Exit node to be detected by Tor. It is also implied that the infection of specific users is considered more difficult than spreading SnoopyBot among the public at large.
- Currently, SnoopyBot works only with Orweb. However, there are several other web browsers that can be used in cooperation with Orbot, and thus SnoopyBot needs to be modified to co-work with each one of them.
- The user's smart device must be rooted. This condition is necessary for the spyware to perform its actions.

Basically, to cope with the attack described in [76], defenders need to take precautions for both the Exit node and the mobile device. This is because the only thing the attack does is to build a covert channel between the mobile device and the Exit node, which are both controlled by the attacker. Regarding the Exit node, traffic sniffing is very hard to detect; practically, there is no foolproof way for the Tor network to tell if an Exit node monitors traffic. This means that the only effective countermeasure against a rogue Exit node is to only use Exit nodes that are known to be trusted, at least up to a certain degree, e.g., by consulting [84].

SnoopyBot requires a rooted device, which in turn means that a non-rooted device is invulnerable. If the mobile device is rooted, then the settings for Orbot can be stored in an encrypted database to avoid being manipulated by an evildoer. Barring the situation that a great number of malicious Exit nodes exist, if SnoopyBot cannot access Orbot settings, the probability for a mobile user to get infected by SnoopyBot and use a malicious Exit node is tiny. Naturally, another defensive measure is to alert the end-user upon detecting any unprovoked change to the mobile browser settings. This would be enough to raise the user's suspicion, making them to initiate a device scan for malicious apps. Moreover, the user should avoid installing any app that is not in, say, the Play Store and/or its origin is vague.

Lastly, provided that there is an Intrusion Detection System (IDS) installed in the device (or on the same network with the mobile device), there should be an alert for any internal IP that queries websites for obtaining the user's public IP, especially if it is a frequent request from the same internal IP.

3.3.3 Discussion

As already mentioned, Tor's major weakness is the Exit node of each circuit, as all traffic that passes through this node is potentially unprotected. This shortcoming attracted several researchers to develop methods of tracking and avoiding malicious Exit nodes. However, as with SnoopyBot, this is hard to achieve in cases where the Exit node just silently logs the traffic passing via it, leaving no other trace of its privacy-invasive activity. In this case, countermeasures need to be taken on the client side as well. For instance, the apps that provide access to Tor must encrypt their settings or use a secure database for storing them. In any case, however, the root cause of the Exit node's problem is not because of the internal workings of Tor, but to end-users not employing HTTPS connections or other means of protection at the application layer. This problem is even aggravated by badly configured web browsers or other applications and the rise of privacy-invasive software, as in our case.

Moreover, one of the most prominent security issues that can occur during Tor installation on, e.g., a smartphone is the root permissions Tor requires to anonymize outgoing traffic stemming from any application other than Tor's official browser. This requirement leads many users to root their smart device, which, as a direct consequence, enables any malicious application to gain access to critical files on the Android system.

Chapter 4

Complete SIP message obfuscation

4.1 Introduction

As already mentioned in Section 1, the demand for private communications is high among businesses, activists, journalists, military, and law enforcement [85]. Nowadays, telecommunication providers increasingly shift their business model towards VoIP communications which are more flexible and inexpensive, but with more security and privacy issues compared to traditional communications. One of the most prominent protocols supporting multimedia services is the SIP [86], which is an application layer, text-based, signaling protocol responsible for session management. With reference to Section 2.2, despite its popularity, SIP still suffers from privacy issues, two of the most notable of which are (a) user identity, and (b) IP address disclosure. Simply put, since SIP signaling messages are in plaintext, an eavesdropper can acquire sensitive data such as: communicating parties' names and affiliations, IP addresses and hostnames, and SIP Uniform Resource Identifiers (URIs). The SIP header fields that reveal these details are mainly `From`, `To`, `Contact`, and `Call-ID`. An example of a SIP message header format is presented in Table 4.1; here, some data like `branch` and `tag` values were omitted for readability.

Apart from SIP signaling messages, the architecture of the protocol per se is also another source of problems. That is, SIP can be employed in either client/server or P2P architectures; in both cases the participation of intermediary servers complicates the privacy problem. Previous work on these issues [87, 88] has shown that several header fields

Table 4.1: SIP message header format

```
INVITE sip:al@agn.org SIP/2.0
Via:  SIP/2.0/UDP pc8.agn.org;branch=...
Max-Forwards:  70
To:  Al <sip:al@agn.org>
From:  Geo <sip:geo@agn.org>;tag=...
Call-ID: a84b4c76e66710@pc8.agn.org
CSeq:  314159 INVITE
Contact:  <sip:geo@pc8.agn.org>
Content-Type:  application/sdp
Content-Length:  142
```

can provide private pieces of data to eavesdroppers. Although for some data privacy protection is straightforward if the user selects not to provide them, other header data cannot be omitted since they are needed for the correct routing of SIP messages to their final destination.

Ideally, any complete anonymity solution for SIP should be designed and provided in a cross-layer manner. That is, while SIP operates at the application layer, several other sensitive information regarding its signaling inevitably leak from lower layers. For example, while it is possible to conceal the IDs of the communicating parties by applying, say, a pseudonymity scheme to `Via` and `From` headers, the IP addresses of both ends are available to an observer by just tracking the headers of the IP packets conveying SIP messages.

In line with Obj. 1 defined in Section 1.3, this shortcoming motivated us to think of taking advantage of the services of a generic anonymization system with the aim to apply a holistic anonymity solution for SIP. On the one hand, such a solution seems promising as anonymization systems like Tor [89] are self-reliant, i.e., normally their operation does not hinge on the protocols of upper layers, hence they can be straightforwardly combined with them. Moreover, as discussed further in Section 4.2.2, such a system can protect SIP signaling from a plethora of other type of attacks including timing and collusion ones. However, on the negative side, taking Tor as an example, the problem with SIP is that currently Tor only supports TCP for its transport layer. As a result, although RFC 3261 [86] requires all SIP entities to mandatory implement both UDP and TCP, many real-world VoIP applications rely solely on UDP for latency reasons. Therefore, at least for the time being, this is a serious impediment for VoIP users to enjoy strong anonymity to real-time voice communication. Tunneling of the UDP traffic through Tor does not really solve this issue because the traffic would be encapsulated in TCP. The latency induced by Tor is also known to be quite heavy, as the system relays and mixes its traffic via multiple nodes.

And if so, to which network hops should be preferably Tor activated for achieving a fair balance between the level of anonymity and the time penalty introduced? Moreover, what is the additional delay if one considers to even anonymize network links that cannot be covered by Tor (think of the first hop between the caller and the outbound SIP proxy or the registrar).

To shed light on the aforementioned questions, we implemented a proof-of-concept SIP-over-Tor system and conducted measurements to assess its performance in terms of service times. Additionally, we combine this system with a pure application layer anonymization solution for SIP to make a decision whether an end-to-end preservation of anonymity is affordable. The results we obtained seem quite promising, showing a latency in the vicinity of 2 secs across all the tested scenarios. It should be noted here that our solution, as well as the aforementioned delay, concerns SIP signaling only; namely, media protection should be considered separately.

As discussed in Section 4.2, previous work in the same topic is fragmentary and has only touched upon these issues, not considering SIP at all. Therefore, to the best of our knowledge, this is the first work that elucidates on the foregoing issues and provides real-life results that can be used as a reference towards building truly anonymous VoIP systems.

4.2 SIP Torification

This section details how the combination of PrivaSIP [9, 10] with Tor can be highly profitable in terms of preserving end-users' privacy.

4.2.1 PrivaSIP and PrivaSIP over Tor

PrivaSIP provides an advanced level of privacy compared with plain SIP; however, due to its application-oriented nature, it is not a complete solution. The main issue is that, while real user IDs are concealed, the IP addresses of the communicating peers are still visible. This is because the IP addresses are needed for the proper SIP message routing.

A secondary issue is that end-users' network domains are visible for the same reason, but this is rather minor since domains can also be derived from IP addresses.

The above-mentioned issues of PrivaSIP motivated us to employ Tor to alleviate them. In the rest of this chapter, the term "PrivaSIP" refers to PrivaSIP-2 which obfuscates both the caller and the callee's real IDs. The advantages of using PrivaSIP over Tor are briefly the following; we will further elaborate on them later in this section:

- Third parties cannot mount traffic analysis attacks.
- The real user ID is not leaked to intermediate SIP proxies.
- Log files in intermediate SIP proxies do not contain the real user's ID.
- The real caller ID is unknown to the callee.
- The real caller ID is unknown to the callee's proxy.
- The real callee ID is unknown to the caller's proxy.
- User authentication and accountability, say, for billing purposes, are still supported.

A prototype architecture of our proposed scheme is shown in Figure 4.1. Here we assume that Client A resides in a corporate network and their SIP proxy is placed in the same local network as well. Therefore, it is not necessary to protect this communication with Tor; we do protect, however, the real IDs of the end-users from other corporate users by employing PrivaSIP. The traffic data (a) between Proxy A and Proxy B, and (b) between Proxy B and Client B, are protected from third parties by Tor. To avoid the Tor exit node eavesdropping issue as explained in Section 3.3, it is required that Proxy B and Client B act as Tor relays, so that no plaintext SIP messages are transmitted through the Internet.

One major question related to our choice of protocols is why someone would use PrivaSIP over Tor and not plain SIP. The short answer is because PrivaSIP offers a more advanced level of privacy. First off, since the first hop in the path (i.e., between the caller and the proxy in the corporate network) is not protected by Tor, PrivaSIP assures that the real IDs of the communicating parties are not revealed to the rest of the corporate users. Tor protects the traffic data from third parties; with PrivaSIP we also hide the real ID

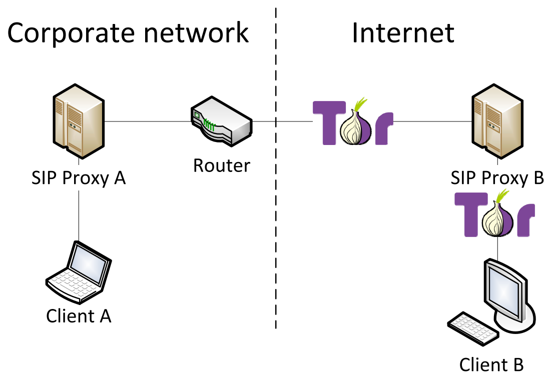


FIGURE 4.1: Prototype PrivaSIP over Tor architecture

of the caller from the callee. Last but not least, SIP messages are delivered as plaintext to intermediate SIP proxies; by employing PrivaSIP we allow their proper routing while protecting the real IDs of the communicating parties at the same time. This also has an implication on proxies' log files, where no real user IDs are stored. This works in favor of unlinkability as well, i.e., certain SIP transactions cannot be correlated in such a way that can be traced back to the same user.

Apart from the aforementioned advantages, the main benefit of our proposal is the protection of traffic data offered by Tor. Essentially, Tor solves the inherent issues of PrivaSIP as presented in Section 2.3. Thus, it prevents traffic analysis attacks by obfuscating traffic data, including SIP end-users' IP addresses and domain names. A more detailed analysis of Tor's benefits can be found in Section 2.4.

Another point of discussion is related to the deployment of the proposed solution and its compatibility vis-à-vis existing SIP infrastructures. Regarding PrivaSIP, some modifications are needed in SIP proxies and user clients in order to properly encrypt/decrypt obfuscated user IDs; more information on this point can be found in [9, 10]. Tor, on the other hand, acts as a proxy, so it can be transparently utilized by end-users and servers. It is, however, needed to be installed and configured so that the callee's SIP proxy and client act as Tor relays.

4.2.2 Privacy analysis

As discussed in the previous section, the qualities of Tor can be of great value in SIP. Specifically, enabling SIP over Tor communications can lead to a robust cross-layer privacy preserving system capable of dealing with a variety of major privacy attacks [89, 90, 91, 92]. First off, message size types of attack are avoided. With reference to Section 2.4, this is because Tor mandates the use of fixed-length cells. As a result, an observer is unable to infer any usable information when examining a cell's length. Moreover, due to the use of encryption, no one is in position to change a cell's coding when in transit through the Tor network. This particular quality also works in favor of protecting from packet context oriented attacks. Simply put, the IP address, application port, etc., included in the TCP header remain well-hidden. In any case, all connections in Tor use TLS link encryption based on ephemeral keys. This way, connections between entities enjoy perfect forward secrecy, preventing medication of data while in transit. For the same reason, attacks aiming at masquerading an OR are also considered unpractical.

In addition, observers are blocked from spying on which circuit a given cell is intended for. Periodical and independent re-keying of TLS ephemeral keys imposed by Tor reduces the impact of a potential key leak. Attacks based on collusion of nodes are also considered highly unpractical. This stands true as all information entering the Tor network is routed through a private network pathway of ORs (circuit) where each relay only knows the previous and the next relay. Lastly, Tor is known to generally defeat privacy attacks based on message timing. In fact, to our knowledge, this issue has been already investigated in [93, 94, 95, 96]. Specifically, the work in [93] argues that a timing attack in Tor is feasible under the global attacker model. However, to be profitable, this attack requires the aggressor to be able to eavesdrop on all network nodes. Moreover, the authors in [94, 95] bring into the foreground some viable attacks under the weaker threat model (no global adversary).

These attacks however are known to be repelled if using supportive protection schemes such as adaptive padding or the insertion of cover traffic in a way that the network cannot perceive between the different network streams [94, 95, 96]. Last but not least, Tor gives the opportunity to the end-user to also join Tor and become an OR by itself (either a relay or a bridge). This situation presents two significant advantages. First, it is anticipated to gradually reduce the latency perceived by all the nodes, and also results in a safer network. Secondly, it would end up in a situation in which a malicious user would be unable to distinguish which connection is initiated as a user and which as an OR.

A last remark here, also succinctly pointed out in the previous sections and in Chapter 2, is that the boundaries of the system do not enjoy protection by Tor. That is, the endmost communication link between the last OR and the callee cannot be protected, therefore an ill motivated entity could eavesdrop on the packet content. In our case, this problem is tackled by requiring the callee's SIP proxy and client to act as Tor relays so that the packet content is not revealed to third parties. For the other end of the communication path, i.e., between the sender and the OP, it is argued that no protection is required as an OP runs locally in the caller's machine. Overall, it can be said that the synergistic operation of PrivaSIP with Tor assembles a powerful solution that achieves the protection of end-users' privacy in a cross-layer fashion. Naturally, PrivaSIP could be easily co-work with other privacy solutions for TCP/IP layer, including MorphMix [97] and Tarzan [98] ones.

4.3 Testbed and performance evaluation

Since Tor is known to cause long delays, our biggest concern is whether the latency in the proposed system is affordable from the user viewpoint. The current section presents the architecture used in our experiments and shows that the delay perceived by the end-users is relatively low, within the range of 1.8 to 2 sec in the worst case. This time penalty is in absolute relation with the protection of users' privacy, more specifically the preservation of their anonymity. As a reference, in previous works [9, 10] the delays were approximately 0.5 sec for plain SIP and 1.2 sec for PrivaSIP-2-RSA. Those measurements, however, were taken with a different hardware setup and server traffic and cannot be directly compared with the present results.

4.3.1 Testbed

The architecture used for the experiments is a simplified interpretation of a real-life case, described in the following and depicted in Figure 4.1. Alice (Client A) wants to call Bob (Client B). Alice is registered to a network domain served by SIP Proxy A, which is situated within a corporate network. Since Alice and SIP Proxy A belong to the same network, which is considered trusted in the normal case, we avoid using Tor on this link. However, Tor is employed outside the corporate network, that is, both between the two SIP proxies, and SIP Proxy B and Bob.

First, we describe the underlying hardware used in our testbed. All servers and clients were hosted on Virtual Machines (VM). To host those VMs, we chose Okeanos[1], a cloud service provided for the Greek Research and Academic Community. With Okeanos, we had the ability to create VMs with dual-core processors, 4 GB of RAM, and 60 GB storage. Okeanos provides high speed Internet connection to its users which can reach up to 520 Mbps. There is also the possibility to choose between numerous operating systems; for our scenarios we chose CentOS 6 for the servers, and Ubuntu 12.04 for the clients.

Both SIP proxies were based on SER 0.9.6[2], and modified accordingly as in [9, 10] to support PrivaSIP. For traffic generation on the caller side, IPp[3] was used, along with sipsak[4]. On the callee side, a modified User Agent (UA) that supports PrivaSIP was used, based on Twinkle software phone[5].

4.3.2 Results

This section describes the procedures followed and the metrics used during the experiments, as well as the results obtained. First, Bob's UA acting as the caller sends an INVITE message using PrivaSIP-2 to conceal both the caller's and the callee's IDs using RSA. As a consequence, the caller first receives back a 100 (TRYING) message followed by a 180 (RINGING) signifying that Alice's phone is ringing. In this context, we measure the time starting from when the INVITE was sent until the RINGING message is being received back at the caller side. This network time includes the operations needed for PrivaSIP, as well as delays imposed by Tor. We should note here that we take the worst case scenario for Tor delay. Since Tor utilizes the same circuit for sessions that take place within the same 10 min or so, we force each new call to be placed over a new circuit.

[1]https://okeanos.grnet.gr
[2]http://www.iptel.org/ser
[3]http://sipp.sourceforge.net
[4]http://sourceforge.net/projects/sipsak.berlios
[5]http://www.twinklephone.com

For the sake of comparison, we measured call delays for two scenarios: (a) plain SIP over Tor, and (b) PrivaSIP over Tor. These two scenarios do not have the same level of privacy protection, so they cannot be directly compared. The reason we chose them is to identify the sources of delays, since more than one protocols are involved. In each scenario we followed the aforementioned procedure and 100 calls were sequentially produced with the help of SIPp tool. Using Wireshark on the caller's side, we were able to compute all call delays. The derived values were rounded to one decimal digit and the frequency per value was counted.

For the plain SIP scenario, the results are summarized in Table 4.2 and illustrated in Figure 4.2. As it is easily observed from the table, the majority of the delays span between 0.9 and 1.1 sec. On the other hand, for the PrivaSIP scenario, the results are presented in Table 4.3 and Figure 4.3. In that case, the majority of the delays are between 1.8 and 2 sec, leading to the conclusion that the perceived delay from the end-users while waiting for the call to be established is relatively low. What these results show us is that PrivaSIP together with SIP operations adds a delay of approximately 1 sec to the whole scheme, and the rest is caused by Tor. Even if the plain SIP scenario has better performance, the one employing PrivaSIP is preferred, since it comes with more advanced privacy preserving features. In any case, both these methods can be offered to the end-users in an opt-in basis.

4.4 Discussion

Few will argue that the preservation of user anonymity is an important issue which pertains to almost any protocol or technology deployed in the wired or wireless Internet.

TABLE 4.2: Range of call delays for plain SIP over Tor

Delay (sec)	**Frequency**
0.6	3
0.7	5
0.8	2
0.9	27
1	43
1.1	18
1.2	1
1.3	0
1.4	1

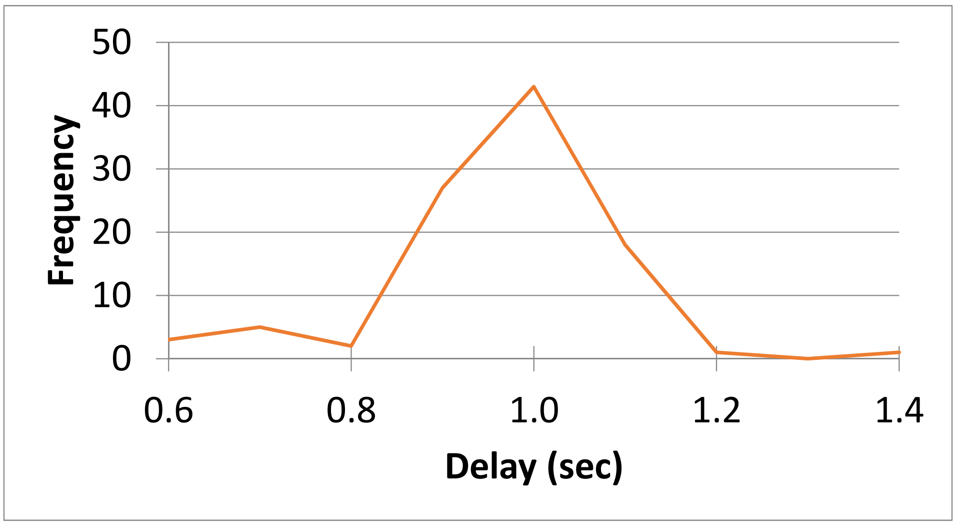


Figure 4.2: Frequency of call delays for plain SIP over Tor

Table 4.3: Range of call delays for PrivaSIP over Tor

Delay (sec)	**Frequency**
1.5	1
1.6	0
1.7	3
1.8	26
1.9	37
2	27
2.1	0
2.2	1
2.3	0
2.4	2
2.5	2
2.6	0
2.7	0
2.8	0
2.9	0
3	1

When it comes to VoIP, it is for sure that not only a broad category of users would highly appreciate anonymous communications, but also several providers would value such a service towards expanding their market share. Unfortunately, all works proposed so far in the literature – either standardization efforts or custom-made solutions – tackle

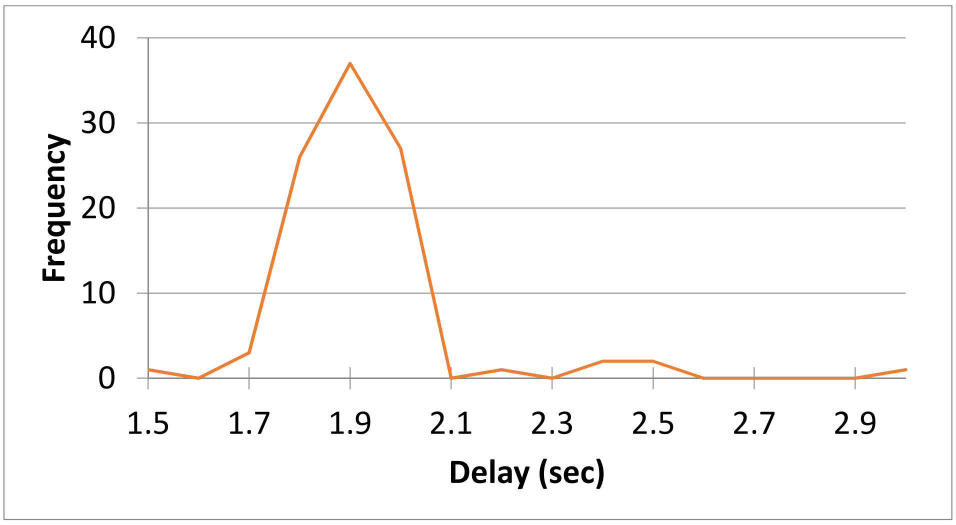


FIGURE 4.3: Frequency of call delays for PrivaSIP over Tor

this problem in an unsatisfactory way. Some of them focus solely on the application layer, thus neglecting sensitive data leaking from lower layers, while others propose inflexible or difficult to deploy mechanisms that either require external infrastructures or are in direct contrast with user accountability. On the other hand, secure tunneling of VoIP traffic by means of, say, a TLS tunnel is considered mostly unpractical.

Compelled by this fact, we came with the idea of taking advantage of the well-known Tor anonymization overlay to achieve complete SIP message privacy. While this may be seen straightforward, it is quite tricky because the majority of SIP apps are designed to operate over UDP for increased performance. Tor on the other hand works solidly over TCP and introduces additional delays due to the use of public-key cryptography, complex segmentation, and routing of its messages via a set of nodes. Therefore, the main impediment here is performance in terms of service times. In this context, our experiments employing different setups showed that SIP over Tor is quite affordable, as it adds a time penalty that fluctuates between 1.8 and 2 sec in the great majority of cases. Overall, we think that the extra security and privacy gains that Tor brings along compensate for this penalization, which, after all, concerns the establishment of the call and not the multimedia session itself.

The work involved with the present chapter can be used as a reference towards building anonymization solutions that consider privacy in a cross-layer fashion. This is not only bound to VoIP applications, but also for other protocols as that in [99], where its authors have already identified this need and provide the necessary background towards a full-fledged solution.

Chapter 5

OnionSIP: Preserving Privacy in SIP with Onion Routing

5.1 Motivation

The need for ways of communication with lower cost and less maintenance than traditional ones based on PSTN, led most users and organizations to turn their attention to more flexible solutions, like VoIP. The most concrete VoIP benefit that is easily noticed vis-à-vis PSTN is the cost savings, although the high scalability and many free added-value features cannot be ignored either. Recall from Section 4.1 that one of the most prominent VoIP protocols offering multimedia services is SIP [100]; it is an application layer, text-based protocol, used for signaling and managing multimedia sessions. Despite its popularity, SIP still suffers from basic privacy weaknesses and security issues, e.g., information about the users participating in a voice call can be easily exposed to a third party user.

Specifically, with reference to Sections 2.2 and 4.1, due to its text-based nature, SIP suffers from two main weaknesses: the disclosure of (a) user identities, and (b) user IP addresses. Overall, the main reason why SIP is not considered a secure protocol is because the contents of each SIP message are transferred in plaintext, rather than being encrypted in any way. Having that in mind, any malicious user or intermediate proxy, which is able to read the SIP message contents, can reveal sensitive information about the users participating in the call. Specifically, a malicious user may reveal: (a) the caller's name, username, and IP address, and (b) the callee's name, username, and IP address. As already mentioned in Section 4.1, these pieces of information derive from the unprotected `From, To, Contact` and `Call-ID` fields. As with Figure 4.1, an example of a typical SIP message is shown in Table 5.1; here some data, like branch and tag values, were omitted for readability.

```
INVITE sip:alfakis@agn.org SIP/2.0
Via:  SIP/2.0/UDP pc8.agn.org;branch=...
Max-Forwards:  70
To:  Alex <sip:alfakis@agn.org>
From:  George <sip:geokarop@agn.org>;tag=...
Call-ID: a84b4c76e66710@pc8.agn.org
CSeq:  314159 INVITE
Contact:  <sip:geokarop@194.252.165.10>
Content-Type:  application/sdp
Content-Length:  142
```

Table 5.1: SIP message header format

Apart from the fact that SIP messages are transferred in plaintext format and user IPs are exposed through SIP header fields, IPs are also disclosed from packet exchange on the network layer. In that case, any observer can obtain the IP addresses of both ends, just by tracking the headers of the IP packets conveying SIP messages. Thus, it is meaningless to try concealing any IP address appearing in SIP header fields, for instance the ones in `From` and `To` fields. Obfuscating only the application layer information would have no actual effect, as the observer can easily correlate a message leaving the caller with another message that is reaching the callee.

Although standard security protocols, like IPsec and TLS, constitute an effective solution for initializing a VoIP session securely, they come with significant implementation demands, while call setup delays are not negligible either. In [101], the delay of a SIP call setup is measured in two scenarios: (a) using plain IP, and (b) using IPsec. In SIP over plain IP, delays are between 0.5 and 1 sec, while the utilization of IPsec for initiating a call can approach a delay of 7.5 seconds, using the encryption scheme only. The introduced delays concern the SIP protocol and occur only once during session establishment, without affecting the actual call data, which are exchanged in a later phase.

The lack of a truly efficient and easily deployable privacy solution for SIP motivated us to search for a different approach to deal with the user identity and IP address disclosure issues. Our interest was spurred by how a VoIP protocol, like SIP, would act if an anonymizing network intervened. Our research started from the two most widespread anonymizing networks, Tor [28] and I2P [102]. At first, Tor offers the ability to a user to use it as a SOCKS proxy [103], making it a suitable solution for anonymizing SIP traffic. On the other hand, as I2P works quite different from Tor, one will need an outbound proxy to communicate with non-I2P users. At the moment, built-in SOCKS proxy support is not available in I2P, so the only alternative way to anonymize traffic is by using its HTTP proxies or creating a SOCKS client tunnel and forward any traffic to I2P using a third-party SOCKS proxy.

Even though we examined other solutions, apart from Tor and I2P, they cannot support SIP anonymization with reasonable deployment effort. Some are in development or in early experimental stage, like Tarzan [104] or Riffle [51], while others do not provide an applicable proxy to pass SIP traffic through. The above-mentioned facts restrict the diverse techniques to anonymize SIP traffic.

In search of a simpler and more efficient solution, and in accordance to Obj. 2 of Section 1.3, we decided to design and implement an encryption-based privacy-preserving scheme, called “OnionSIP”, where all SIP sensitive fields are encrypted. The same information can later be recovered only by authorized parties, i.e., the proxies of each user, as well as the users participating in the call. In the proposed onion-routing based scheme given in the present Chapter, sensitive information contained in a SIP message is encrypted in a multilayer fashion, capitalizing on the onion routing concept [105]. This way, inbound and outbound SIP messages cannot be correlated, as long as the application layer fields are hidden. However, traffic analysis could still be possible through various techniques, including timing attacks. In order to defend against such attacks, numerous solutions have been proposed in the literature. In [106], the authors propose adding dummy traffic to hide the actual amount of the original traffic, while others [107] suggest the use of constant rate padding between each hop to protect users anonymity. Another approach [108] used a random delay in each message for preventing the execution of timing or similar kind of attacks.

For the scheme contributed by this chapter, we implemented two different types of scenarios. In the first, public-key encryption is used for field protection, based on SIP proxy server certificates, while in the second symmetric-key encryption is used. We compared our solution with Tor, I2P, Orbot [109], which is a smartphone proxy app allowing traffic redirection through Tor, and a previous approach of ours called PrivaSIP over

Tor [52] based on Tor network and PrivaSIP [9, 10]; recall that the latter is described in Chapter 2. Briefly, the experimental results presented in Section 5.5, suggest that PrivaSIP over Tor is less efficient than plain SIP over Tor by almost 1 sec. However, only the former protects user's ID from intermediate SIP proxies; thus the trade-off here is more delay for more privacy. Overall, PrivaSIP over Tor is still affordable, since it takes around 2 secs for a call to be established. OnionSIP, on the other hand, presents better performance from related solutions, while at the same time offers more advanced privacy protection compared to PrivaSIP over Tor by not exposing communicating domains to intermediate proxies.

5.2 Attacker's model

Our proposal focuses on privacy protection of SIP signaling, thus, in the below attacker's model we only consider privacy-related attacks. As a communication model, we assume the traditional SIP trapezoid, where two users communicate using two intermediate SIP proxies. We identify two main classes of attackers:

- **SIP proxies:** While SIP proxies are system entities, they cannot be completely trusted. That is, a mobile user can utilize an outbound proxy that does not belong to their service provider. Here, we argue that SIP proxies follow the honest-but-curious model. According to this model, the proxies are assumed not to drop or modify messages routed through them, allowing the system to run smoothly, while at the same time try to infer private information from the exchanged messages. This way, the proxies, as well as the entities who control them, can mount traffic analysis attacks and get access to private user information, including caller and callee usernames, IP addresses, and network domains.
- **Third parties:** This class concerns external attackers that are neither system entities nor participate in the call. As SIP is a text-based protocol and messages are sent in plaintext, third parties can eavesdrop on all messages that are exchanged among users and SIP proxies. These type of attackers can perform traffic analysis attacks and access caller and callee usernames, IP addresses, and network domains. The difference from the previous class is that third parties can also modify or drop SIP messages in order to mount Denial-of-Service or call hijacking.

5.3 OnionSIP

As already pointed out, SIP is a protocol whose messages are transferred in plaintext, thus, any third party is in position to identify the caller, the callee, as well as their locations. To protect these sensitive pieces of information, in OnionSIP we encrypt any fields that should be hidden from any parties that are not authorized to read. Namely, any user or proxy should be able to read only the information needed to make the call establishment possible. Our proposal is a multilayered encryption scheme, similar to onion-routing, which gives the ability to every node to decrypt the appropriate fields and forward the message to the next node, based on the information it decrypted. The above idea is based on the fact that intermediate proxies in a SIP path can forward packets without problems, although unused sensitive information are hidden from their sight. Similarly to other anonymity networks, OnionSIP's degree of anonymity is highly dependent on the number of gateways that are used between each party. The more the gateways, the higher the level of privacy.

With reference to Chapter 4, OnionSIP is a novel scheme which constitutes a more efficient solution than the previously proposed PrivaSIP over Tor [9, 10], while at the same time preserves the high level of privacy and anonymity the latter offered, i.e., protecting call metadata even from intermediate proxies. In contrast to PrivaSIP over Tor, OnionSIP does not use any third party anonymization system, like Tor. Instead, to achieve user anonymity and privacy, OnionSIP hides the content of each SIP message individually by encrypting the necessary fields and not the whole message.

To choose which fields are sensitive and should be protected, we first consider which ones contain useful information about the call or the users participating in it. First off, the INVITE URI contains information about the final user, i.e., the username and the destination domain. In addition, `From` field contains information about the caller, while `To` field holds the "logical recipient" of the message, which may or may not be the ultimate recipient of the request. These fields include the username, or in some cases even the real name of the user, as well as the domain name of their SIP Registar. Moreover, the `Call-ID` is used for offering uniqueness to a session. Apart from the fact that `Call-ID` could help an observer correlate two different SIP messages, it also includes the sender's IP address, making it possible for the eavesdropper to identify every node that is involved in the session. Last but not least, the `Contact` field is used to represent a direct route to contact the caller. All the above fields are considered sensitive, as they can easily expose information about the call, therefore they should be protected. For the proposed OnionSIP system, we apply two different encryption schemes: an asymmetric, and a symmetric one.

5.3.1 Asymmetric OnionSIP

The first variation of our proposal, also called OnionSIP-RSA, makes use of public-key encryption to encrypt the appropriate fields of SIP messages. It is assumed that the caller already possesses the public keys of all the network hops in the path of the call towards the callee, including the callee's public key. To explain how our framework works, we assume that Alice, registered to SIP Proxy A, wants to call Bob, who is registered to SIP Proxy B, as depicted in Figure 5.1. Sensitive fields in the INVITE message are being encrypted by Alice in layers in such a way that each party can decrypt one layer, obtaining information for the next hop only. In the following, we present the steps that take place for the call establishment:

Step 1: Alice concatenates `INVITE, From, To, Call-ID` and `Contact` fields forming the Layer1 message shown in Figure 5.1. Then, she encrypts it with Bob's public key (KC).

Step 2: Alice appends the string "`To: Bob@proxyB.org`" to this encrypted message and encrypts the result with the public key of Proxy B (KB).

Step 3: Alice concatenates the string "`To: anon@proxyB.org`" to the last encrypted message and encrypts the result with Proxy's A public key (KA).

Step 4: The encrypted message is placed in the message body of the SIP message, while SIP header fields are all replaced with anonymous SIP URIs, for example "`To: anon@hidden.org`".

Step 5: Alice forwards the resulting SIP message to its own proxy, i.e., Proxy A.

Step 6: Proxy A decrypts the message it finds in the SIP message body using its private key (KA^{-1}).

Step 7: Proxy A replaces, in the SIP message it received, the header `To` with the decrypted value, i.e., "`To: anon@proxyB.org`"; this way it knows the next hop which is Proxy B without knowing the final destination, i.e., Bob.

Step 8: Proxy A replaces the SIP message body with the decrypted message without including the "`To: anon@proxyB.org`" value.

Step 9: Before forwarding the INVITE message, Proxy A generates a TRYING message and encrypts any sensitive information contained in it, using Alice's public key, and sends it back to her.

Step 10: Similarly, Proxy B decrypts the message it finds in the message body, using its private key (KB^{-1}), replaces the `To` field with "`To: Bob@proxyB.org`", and also replaces the message body with the decrypted message, responding back to Proxy A with an onion-encrypted TRYING message.

Step 11: Finally, Bob decrypts the message it finds in the SIP message body with his own private key (KC^{-1}), where he can find all the necessary information about the caller.

Step 12: If Bob needs to send a reply, he follows the same concept and encrypts sensitive information using the public keys of the same path in the reverse order or a totally different path for enhanced privacy.

The above procedure is presented in Figure 5.1. As one can easily observe, the domain of the `To` header field in Msg. 1 is not hidden, as it contains information about the next hop.

5.3.2 Symmetric OnionSIP

In this variation, also called OnionSIP-AES, the OnionSIP framework works as mentioned above, with the main difference being that the parties that need to communicate with each other already share a symmetric key to encrypt SIP message contents. We assume that the involved parties already have digital certificates and the caller has earlier established a different session key with each of the parties along the path to the callee, i.e., Alice should negotiate session keys with Proxy A, Proxy B, and Bob during the registration process by means of, say, an authenticated Diffie-Hellman handshake. This

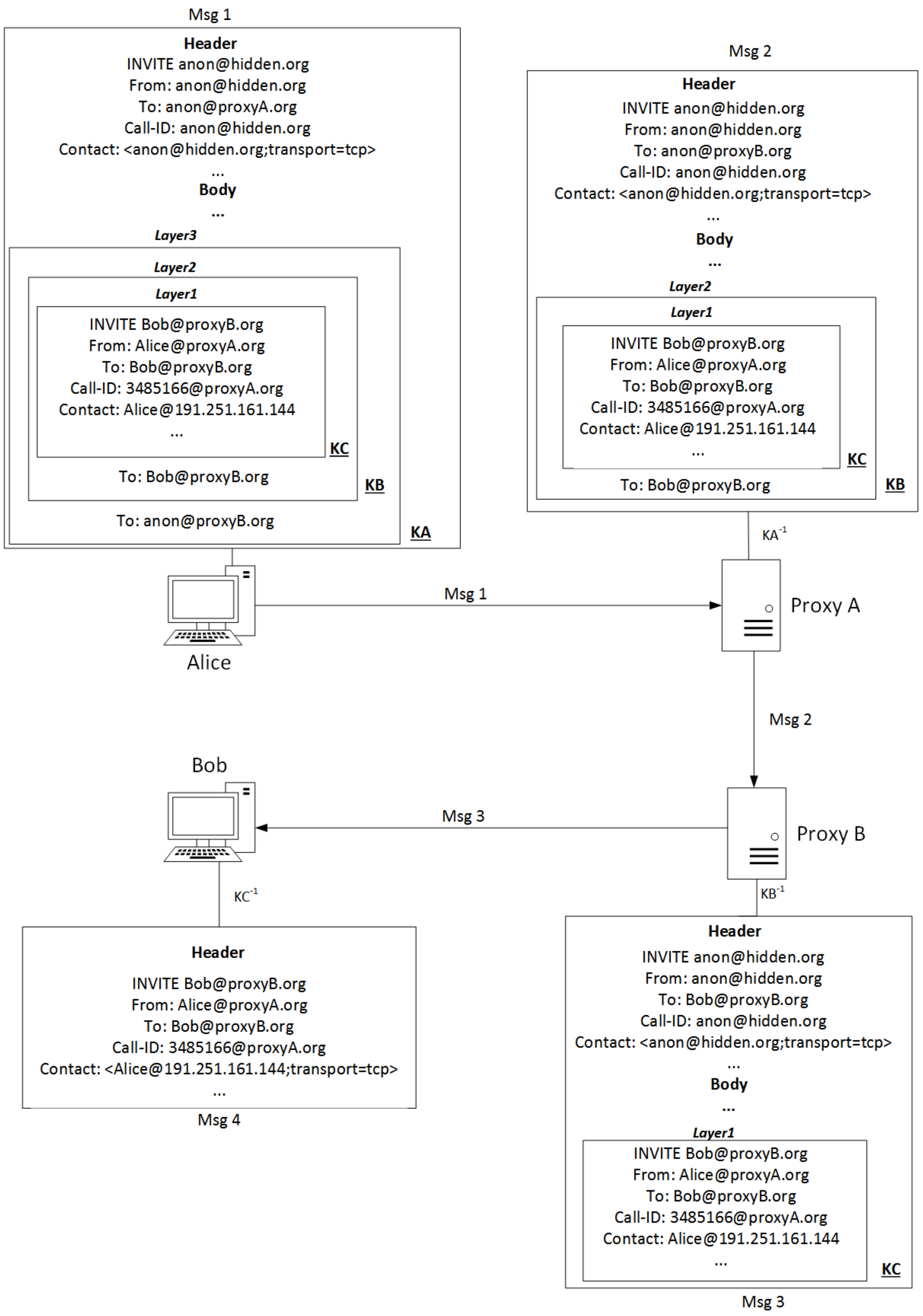


Figure 5.1: Example of anonymizing a SIP INVITE message

implementation is close to how modern anonymization networks using onion routing (like Tor) work, as they use symmetric encryption to protect traffic traversing through each node of the chain.

5.3.3 Key exchange

The correct operation of OnionSIP requires that Alice and Bob exchange their public keys or agree to a common key. This kind of authentication and key exchange can lead to privacy leakage to an adversary that observes communicating IP addresses; however, apart from the direct exchange of keys, there are alternative solutions that do not breach user privacy. First, Alice can acquire Bob's certificate from a Lightweight Directory Access Protocol (LDAP) server or a Certificate Authority (CA) and vice versa. This is a common solution that can occur whenever two users need to communicate with each other for the first time. Moreover, when a user receives a certificate, the latter can be stored for future transactions, as long as the key is considered long-termed. In the case of symmetric keys, a Key Distribution Center (KDC) can be used in order to avoid direct communication; these session keys can also be stored for a limited period and used for more than one call.

As mentioned in Section 5.1, a comparison was conducted among six candidate solutions, including the solution proposed in this chapter (both asymmetric and symmetric Onion-SIP), SIP-over-I2P, Orbot, SIP-over-Tor, and PrivaSIP-over-Tor, with respect to delay. In our experimental results given in Section 5.5, when it comes to OnionSIP, we have not considered key exchange/agreement delays, as a plethora of parameters and conditions could affect the key exchange phase in different ways. For instance, some keys can be stored during previous sessions and reused in future ones. In addition, Tor's performance is highly dependent upon various parameters of each node that is part of the final circuit, like the location and the bandwidth of each node. As the authors in [110] mention, it usually takes almost 4 sec in average for a Tor client to establish a circuit. This is a significant delay, meaning that in a real-world application an established circuit would be reused, leaving the key agreement procedure out of the session establishment.

5.4 Implementation

As presented in Table 5.2, for the first five candidate solutions of our comparison, we utilized Kamailio Server v.4.4 [111] on Cloud-based VM's hosted on our own infrastructure for SIP Proxies, while SER v.0.9.6 was used on the PrivaSIP-over-Tor, given in [52]. The Kamailio servers were installed on Ubuntu 14.04. Each VM, of both clients and servers, incorporated 4 GB RAM, an Intel Xeon E5-2690 processor and 60 GB of SSD storage. Each SIP client, behaves differently to each of the anonymization systems, therefore we exploited different clients for each scenario, depending on which one worked. We chose to use Twinkle v1.4.2 [112] for both Tor and I2P, while for Orbot we used CSipSimple [113] on the caller's side and Sipdroid [114] on the callee's one. Finally, for the OnionSIP-AES and OnionSIP-RSA schemes, we used Jitsi v2.9 for both caller and callee. The above software characteristics are summarized in Table 5.2, while Table 5.3 presents the hardware characteristics of each host used in our experiments.

In Table 5.4 we make a layered representation of the different platforms we examined in the context of OnionSIP. Specifically, we present each system we used and in which OSI layer it belongs, along with the type of each system. OnionSIP along with the two SIP clients are both sitting on the application layer. Tor and its mobile version Orbot are using a SOCKS proxy to forward traffic through the onion-routing network, so they both operate on the Session Layer [115]. On the other hand, I2P uses its own API to

No.	Scheme name	Proxy A	Proxy B	Client A	Client B
1	OnionSIP-AES	Kamailio v.4.4.0	Kamailio v.4.4.0	Jitsi v.2.9	Jitsi v.2.9
2	OnionSIP-RSA	Kamailio v.4.4.0	Kamailio v.4.4.0	Jitsi v.2.9	Jitsi v.2.9
3	Tor	Kamailio v.4.4.0	Kamailio v.4.4.0	Twinkle v.1.4.2	Twinkle v.1.4.2
4	I2P	Kamailio v.4.4.0	Kamailio v.4.4.0	Twinkle v.1.4.2	Twinkle v.1.4.2
5	Orbot	Kamailio v.4.4.0	Kamailio v.4.4.0	CSipSimple v.1.02.03	Sipdroid v.3.0
6	PrivaSIP over Tor	SER v.0.9.6	SER v.0.9.6	Twinkle v.1.4.2	Twinkle v.1.4.2

Table 5.2: Software characteristics per node

Machine	CPU	RAM	OS
Proxy A	Single-Core Intel Xeon E5-2690	4 GB	Centos 7 kernel v.3.10
Proxy B	Single-Core Intel Xeon E5-2690	4 GB	Centos 7 kernel v.3.10
Client A	Single-Core Intel Xeon E5-2690	4 GB	Ubuntu 14.04 kernel v.3.16
Client B	Single-Core Intel Xeon E5-2690	4 GB	Ubuntu 14.04 kernel v.3.16
Android Caller	Snapdragon 800 Quad-core 2.3 GHz	2 GB	Android 6.0
Android Callee	Snapdragon 800 Quad-core 2.3 GHz	2 GB	Android 5.0

TABLE 5.3: Hardware characteristics per node

anonymize traffic rather than a SOCKS proxy, so it is considered to work on top of the Network Layer [116].

5.4.1 Tor

To anonymize network traffic originating from a SIP client or a SIP Proxy, we should make use of Tor network as an intermediate, using Tor's default port 9050 on localhost. As there is no SIP client able to handle SOCKS5 connections, we used Proxychains [117], a software which can tunnel traffic through any proxy server in order to force Twinkle to use Tor. Proxychains advantage is its SOCKS5 support, which is the main protocol for transferring packets through Tor. On the other hand, we used `iptables` routing rules

OSI Layers	Platforms/Systems	Type
Application	Jitsi	SIP Client
	SipDroid	Android SIP Client
	CSipSimple	Android SIP Client
	Twinkle	SIP Client
	OnionSIP	Anonymity Network
Presentation		
Session	Tor	Anonymity Network
	Orbot	Anonymity Network
Transport		
Network	I2P	Anonymity Network
Data Link		
Physical		

TABLE 5.4: OSI layer placement of used platforms

to force any traffic originating from the caller's proxy to get through the Tor network via Tor's standard port 9050. However, this is necessary only at the caller's proxy side while communicating with the callee's proxy. Both the caller and callee have already established a connection through Tor with their proxies during Register.

5.4.2 Orbot

For testing Orbot, we used two rooted Android smartphones, a Nexus 5 with Qualcomm MSM8974 Snapdragon 800 processor and 2 GB of RAM as a caller, and an LG G3, with a Qualcomm MSM8974AC Snapdragon 801 processor and 3 GB of RAM as a callee. We also used CSipSimple hlADD REF as the caller and set up Orbot to anonymize any traffic coming from it. Lastly, the callee used Sipdroid hlADD REF.

5.4.3 I2P

I2P administrators, have cut down SOCKS outproxies support, making it impossible to create and use a SOCKS proxy as an exit-node to the Internet. The only alternative is to use Tor as an exit node, but this is considered already a heavy burden to carry. However, considering the "HTMLish" form of SIP, it is possible to use HTTP proxies that are provided for anonymity networks, like I2P. The clients are using Proxychains to connect to I2P network through I2P's default port (4444) on localhost. Additionally, the caller's proxy is using `iptables` routing rules to redirect any outgoing traffic through port 4444. Nevertheless, if two users wish to communicate securely, they both need to be part of the I2P network. If they are not, an outproxy needs to be used, transferring traffic from I2P to the Internet. In that case, the outproxy acts similarly to a Tor exit node, thus after that step all traffic is in its original form, i.e., if no encryption was initially used, then it is in plaintext.

5.4.4 OnionSIP

As described in Section 5.3, for OnionSIP, we implemented two different schemes, with the first using asymmetric encryption to achieve onion-routing, while the other is exploiting symmetric encryption. For the first version, we chose RSA as the public-key algorithm, with each key having a 2048 bit size. The symmetric-key of the latter is implemented using AES with 128 bit long symmetric keys, similarly to Tor [28].

We implemented six different SOCKS gateways written in Python, which intervene between the communicating parties as shown in Figure 5.2. The caller and the callee have one gateway proxy each, serving both for encryption and decryption. Each SIP proxy has two gateways: the first is used for decrypting incoming traffic, while the second is used for encrypting outgoing traffic. For example, Proxy's B Gateway 1, is responsible for encrypting and decrypting any traffic that is sent to or received by Bob, while Gateway 2 is responsible for handling communication between Proxy B and Proxy A. The gateways may coexist with the proxies and/or the end-user machine or operate in separate machines. Each party communicates using a TCP connection, as SOCKS5 requires applications to establish a TCP connection to the SOCKS proxy before it forwards any packet.

It is worth noting that by using such a scheme, SIP messages contain extra, unknown information in their body, without affecting the call establishment in any way. This means that OnionSIP is fully compatible with SIP and can be used without modifications

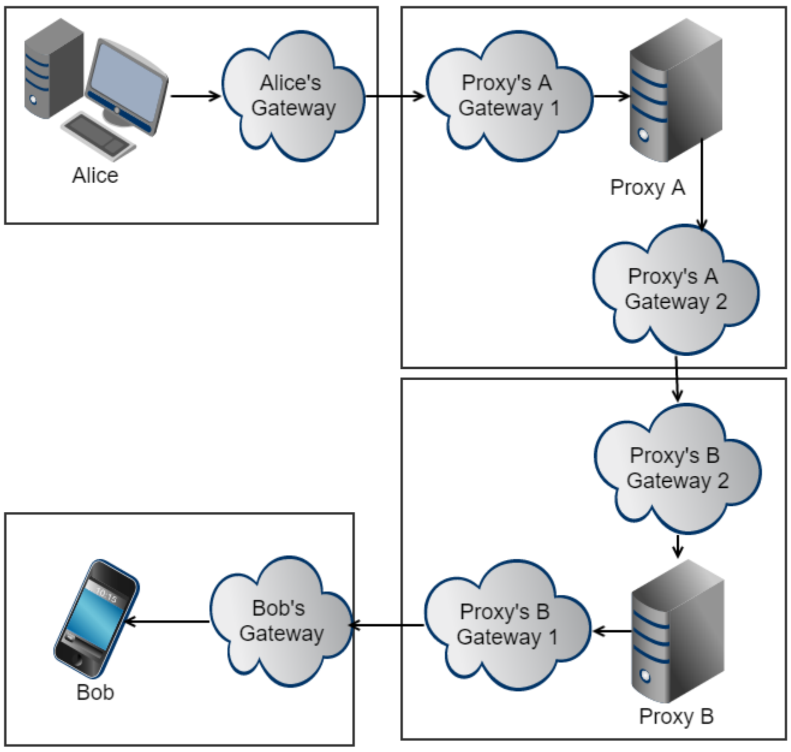


FIGURE 5.2: Architecture of OnionSIP multi-layer encryption

to SIP infrastructure except the addition of the corresponding gateways. Kamailio does not use any SDP parser to check if the messages transferred are valid or malicious in any way. This can constitute a major security flaw for a widely-used SIP proxy, like Kamailio, thus it should be taken into account before wide deployment of this solution [118].

5.5 Evaluation

For the evaluation and comparison of the systems contained in Table 5.2, we measured the establishment delay of at least 100 SIP calls per scheme. First, Alice, who acts as a caller, sends an INVITE message to Bob, who accordingly acts as a callee. Alice receives a 100 (TRYING) message while the proxies, which intervene between the call, try to forward the request to Bob. Finally, when Bob receives the INVITE message, he replies to Alice through the involved proxies with a 180 (RINGING) message. We measure the time between the moment Alice sends the INVITE message and the time she receives the RINGING message back from Bob. Following the above procedure, we produced more than 100 calls sequentially with the help of the SIPp tool hlADD REF. We utilized Wireshark on the caller's side, enabling us to compute each call establishment delay. The derived values were rounded to one decimal digit and the frequency per value was counted.

In Figure 5.3, we present the comparison of the six different alternatives which attempt to preserve user privacy in SIP, using the Cumulative Distribution Function (CDF). OnionSIP based on AES is the most efficient scheme, according to our results, followed by OnionSIP based on RSA, which has a slightly better performance than Tor. In order to have comparable results with Tor, in our AES variation experiments we do not take into account the session key establishment delays. This operation, just like in Tor, takes place one time and, after that, the call establishment delays are those shown in Figure 5.3. Tor has close performance with Orbot, something that was expected since Orbot is based on Tor; they both show delays mainly between 0.9 and 1.1 sec. I2P presents delays between 1.1 and 1.5 sec; this result was expected since SOCKS proxies are by definition noticeable faster that HTTP ones. Finally, PrivaSIP over Tor [52] is the least efficient scheme in terms of performance vis-à-vis the rest of the schemes.

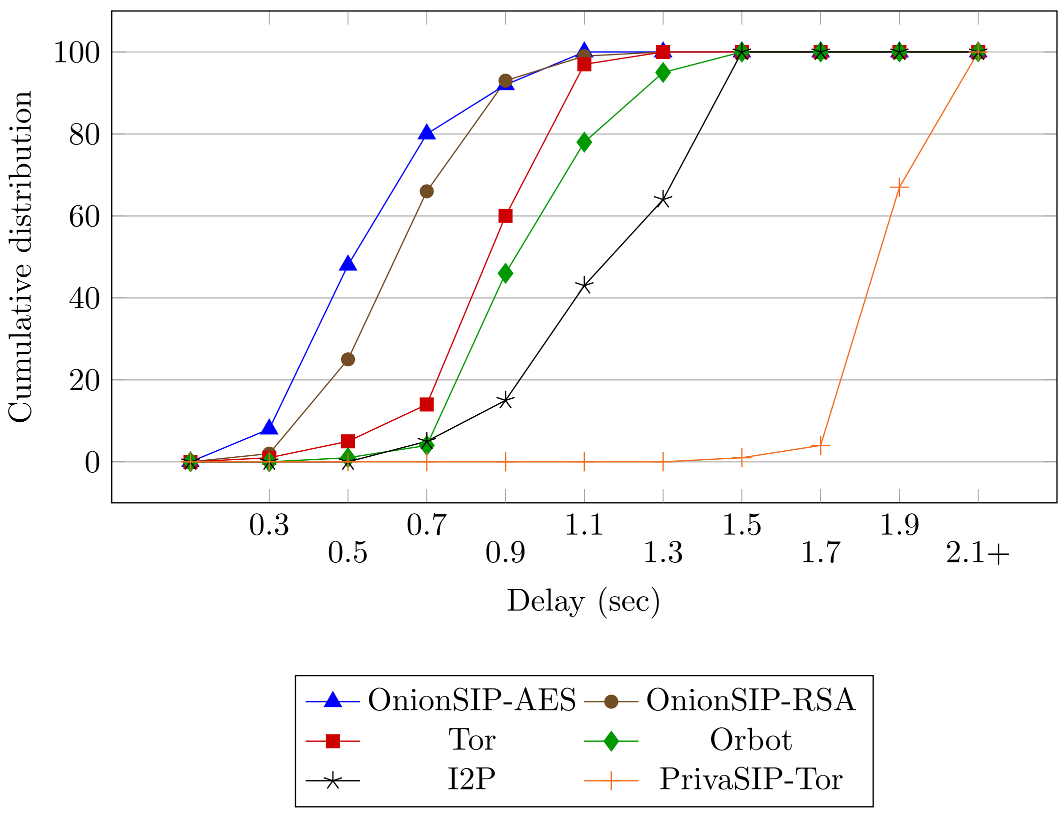


FIGURE 5.3: Delay comparison of privacy schemes for SIP

Taking into consideration the results produced by the CDF graph in Figure 5.3, one can discern that PrivaSIP-over-Tor has a narrow range between 1.7 and 2.1 sec, while other schemes present a wider range, approximately between 0.3 and 1.5 sec. Moreover, Orbot and PrivaSIP-over-Tor present larger delays compared to Tor as expected, since they are based on Tor adding other mechanisms on top of it. Table 5.5 includes the mean and standard deviation metrics for each solution. In this table, one can observe that both OnionSIP and I2P solutions produce the lowest standard deviation compared to the rest of the proposals. Bear in mind that the lower the standard deviation, the more data are clustered around the mean value. Therefore, despite the fact that PrivaSIP-over-Tor seems to have a narrower range (1.7-2.1 sec), OnionSIP introduces more predictable delays; this observation holds both for OnionSIP based on AES and on RSA. Even if the two flavors of OnionSIP are based on symmetric and asymmetric cryptography respectively, we chose to compare them so that adopters can decide whether to take the extra burden required for key management in the symmetric case or not.

The box-and-whisker plot presented in Figure 5.4 illustrates the statistical distribution of

Solutions	Mean	Standard Deviation
OnionSIP-AES	0.6528	14.514
OnionSIP-RSA	0.7265	14.8324
Tor	0.9505	16.970
Orbot	1.0508	15.318
I2P	1.2486	13.523
PrivaSIP over Tor	1.9582	21.260

TABLE 5.5: Comparison of each solution's standard deviation

the call establishment delays. Specifically, this plot presents the median, the interquartile range, and the range. This plot confirms that OnionSIP, in both of its variations, outperforms other privacy preserving schemes.

Table 5.6 presents a brief comparison of the evaluated schemes, indicating the most important criteria each one satisfies. The first two criteria are related to anonymity and privacy, which all the schemes satisfy. The third one concerns the type of message

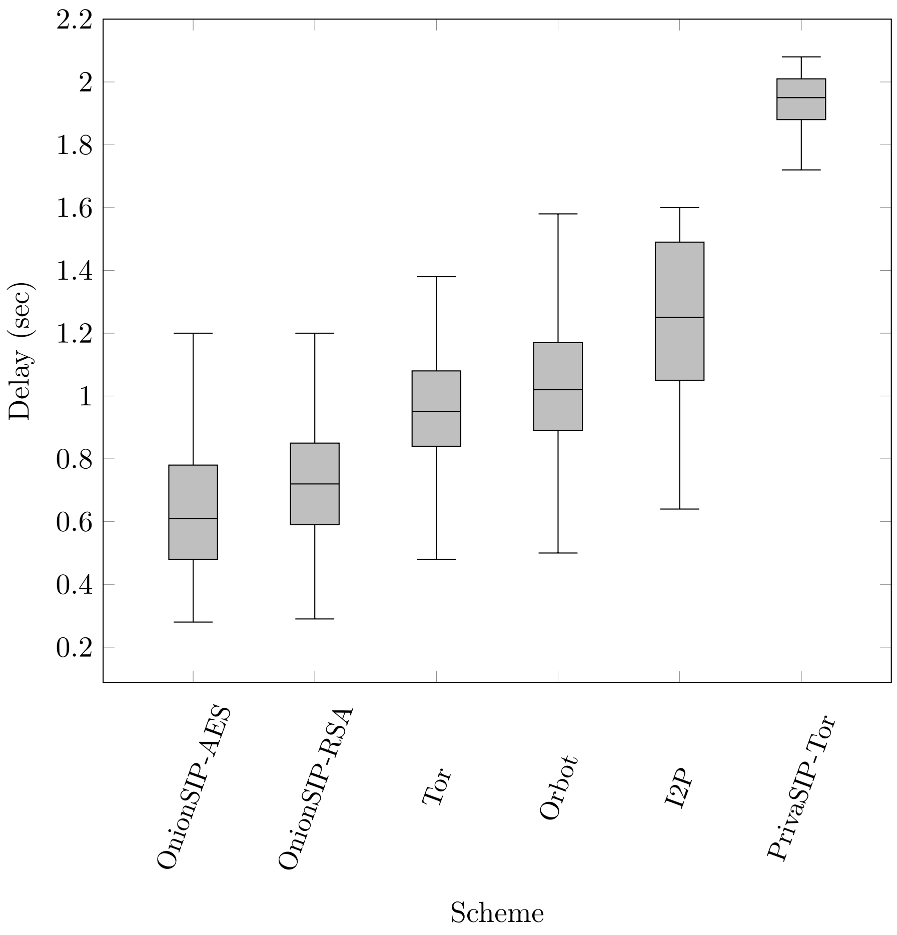


FIGURE 5.4: Box-and-Whisker plots representation of SIP call establishment delays

encryption per utilized scheme. For instance, while OnionSIP encrypts only the sensitive fields of a SIP message, Tor and I2P use a different method by encrypting the whole message. Obviously, the type of encryption each scheme uses, affects the way each proxy stores its logs. Consequently, while both OnionSIP and PrivaSIP encrypt only selected SIP message fields, the intermediate parties will finally store logs where the sensitive information of each message is hidden. On the other hand, in Tor and I2P, the SIP proxies will store logs with each message having its initial form. According to the above, while all solutions offer privacy against third parties, only OnionSIP and PrivaSIP obfuscate user identity from intermediate SIP proxies.

Furthermore, as all the schemes use some kind of encryption, we should distinguish which of them need extra time to establish any keys needed; apart from OnionSIP-RSA, the rest of the schemes will need extra time for key establishment, circuit creation, and so on. Another important criterion is which scheme needs an extra application or system configuration, in order to function properly. In that case, schemes using Tor or I2P need a proxifier to redirect any traffic through them; OnionSIP variations work on the go, therefore no external application is needed. None of these solutions are based on any single point or central server, so all of them offer decentralization.

When it comes to the maturity of the codebase, only Tor offers a mature solution, as I2P is an anonymization network which requires many improvements, and the rest of the solutions are in proof-of-concept status. Finally, based on the fact that I2P requires both users to be part of I2P network or specify an outproxy in combination with the fact that an external application is needed, makes the use of I2P a rather cumbersome task. Additionally, Tor and PrivaSIP over Tor, require the use of an external software to redirect traffic through Tor network. Considering all these facts, OnionSIP forms a privacy solution which could be characterized by ease of deployment.

Criteria	OnionSIP-AES	OnionSIP-RSA	Tor	I2P	PrivaSIP over Tor
Anonymity	✓	✓	✓	✓	✓
Privacy	✓	✓	✓	✓	✓
Encryption	Certain fields	Certain fields	Whole message	Whole message	Certain fields
Privacy against proxies	✓	✓	✗	✗	✓
Privacy against third parties	✓	✓	✓	✓	✓
No extra time for key establishment	✗	✓	✗	✗	✗
External application independent	✓	✓	✗	✗	✗
Decentralization	3	3	3	3	3
Maturity of the code-base	1	1	3	2	1
Ease of Deployment	3	3	2	1	2

Table 5.6: Comparison of SIP anonymity schemes (✓: included, ✗: not included, 1: low, 2: medium, 3: high)

Chapter 6

Fixing WebRTC privacy leaks

In Chapters 4 and 5, we presented a range of schemes to preserve the anonymity of users utilizing multimedia services over the SIP protocol. Nevertheless, as mentioned in Chapter 1, SIP is also utilized by other communication technologies, like WebRTC, which are lacking a session initiation protocol. On the other hand, WebRTC requires the identification of users' actual IP addresses for establishing a connection among peers, which in turn can potentially expose their privacy when abused by malicious users or websites. In accordance with Obj. 3 as defined in Chapter 1, the present Chapter elaborates on this issue and proposes and evaluates practical solutions for solving it.

6.1 Introduction

The need for enabling P2P communication without the requirement of providing extra plugins or native applications to the peers has been of utmost importance for years. In 2011, WebRTC [5] was developed, offering high-quality real-time communication between browsers, mobile applications and IoT devices. WebRTC uses JavaScript APIs, defined by World Wide Web Consortium (W3C), in order to enable multi-platform voice, text, and video communications.

Typically, WebRTC is used for realizing real-time audio and video calls, web conferencing between a number of peers, and even direct file transfers, without the need for extra extensions or additional applications. The biggest benefit of this technology is that the entire communication is achieved through a typical web browser, like Chrome or Mozilla.

Although WebRTC is supported by companies such as Google, Microsoft, Mozilla and Opera, and despite WebRTC's popularity [119], a significant privacy issue remains unresolved: the IP address, both local and public, of each client visiting a website, which is capable of performing a WebRTC peer connection request, can be potentially revealed to the website host. This enables malicious websites to attack clients which would otherwise be inaccessible, e.g., behind a firewall or NAT. It is worth noting here that such an attack is not due to vulnerabilities in the WebRTC protocol per se, but only by abusing its legitimate features.

In more detail, any user browsing the web can to visit a webpage or use services, say, sending an email. This is feasible as web servers have either a public or a port-forwarded IP address, exchanging data with any device. Specifically, in case the client is behind a NAT gateway, NAT maps the device's private IP address to a public IP address and forwards each request to the server and each response back to the client. However, when it comes to P2P services like videotelephony where both devices, i.e., the caller and callee, are behind different NAT gateways, an obstacle arises when trying to establish the connection. This is because, when the caller tries to initiate a session, the callee's NAT box will be unaware of the callee behind it, and thus will drop the connection [5]. For bypassing such impediments and explore different connection methods, WebRTC deploys some extra protocols.

WebRTC allows media to get transferred between peers, regardless their network topology, even if the peers are behind NAT. Nevertheless, for successfully establishing a connection, each peer needs to dynamically generate and discover the most effective path for sending media to the rest of the peers. WebRTC achieves that by taking advantage of the ICE [100] protocol. Precisely, ICE uses two different protocols, STUN [120] and TURN [121], to correspondingly assist with the user's browser to identify their public IP address and port, and relay traffic if a direct connection fails. A STUN server is used by a peer if the latter needs to identify its IP:port socket information as seen from a public perspective, i.e., as it is generated by the NAT box closest to the server, while the TURN server acts as an intermediate when the peer's network forbids P2P connections. The response received by STUN is made available to the JavaScript code that initiated the request; since this piece of code runs locally on the user's system, it has access to the private IP address of the user as well. This procedure is transparent to the end-user, so they are completely unaware of these STUN/TURN requests when they visit such a webpage.

An attacker is able to set up their own STUN/TURN server, forcing the user's browser to perform queries to this server, by placing a short JavaScript code to a webpage before the user visits it. As a result, anyone who has access to this rogue or compromised server is able to obtain the private and the public IP of the user, which is a piece of personal information that may lead to the identification, say, at minimum by means of geolocation, of the end-user.

WebRTC technology exists in all modern web browsers, so any security and privacy issues would affect any end-user browsing the web, and that is the reason why proposing an effective solution is important. In order to avoid risking their endpoint security, most security-savvy users that are aware of WebRTC's issues, prefer to disable any WebRTC services. Naturally, this is a temporary and certainly incommodious solution, as real-time communication is an integral part of most messaging platforms, including Messenger, Google Hangouts, and others. Some users even employ a VPN solution or connect via Tor or the I2P anonymity networks, but this may come at the expense of experiencing low quality real-time communication. Furthermore, as detailed in Section 6.4, the aforementioned solutions are sometimes ineffective.

A more reasonable solution would inform the user whether the visited website is trying to use the browser's WebRTC capabilities [61], rather than completely disabling it for all websites. Any website that is not related to web communication between users, for instance a "news" webpage, should probably not try to make any WebRTC requests, thus the user could safely select to block it.

To solve the aforementioned vulnerability following a user-awareness approach, we implemented and propose two novel solutions: (a) a browser extension and (b) an intermediate trusted gateway, both able to examine a webpage before it is loaded on the browser and inform the user about any WebRTC actions that the webpage is about to execute. We scrupulously evaluate the proposed schemes in terms of performance and show that the additional delay is most of the time negligible, even for websites with an above average size.

6.2 Media Connections

There are two basic call topology types in WebRTC. In the simplest scenario, the peers are employing the same WebRTC service offered by the same webpage, creating a triangle model as Figure 6.1 depicts. On the other hand, if the peers' browser are using services hosted by separate web servers, then the model becomes a trapezoid, inspired by the so-called SIP trapezoid model, as illustrated in Figure 6.2. In this latter case, the two different web servers need a way to communicate, and thus, signaling messages are used to set up, manage and terminate the communication. The protocol used for this communication is not part of WebRTC, however, a standard protocol, such as SIP, JavaScript Session Establishment Protocol (JSEP) [122], or a proprietary one can be used [123]. In both the triangle and trapezoid models, the caller's and the callee's browser is communicating with the corresponding server, exchanging signaling messages using either HTTP or WebSocket protocol, in a way also external to WebRTC.

Most of the time, P2P communication over the Internet is complicated despite the simplicity of the network architecture. Precisely, the peers might either be behind a firewall restricting network access, sitting behind a NAT box lacking of a public IP, or their router may forbid a direct connection to peers, so the use of a relay is necessary. In the

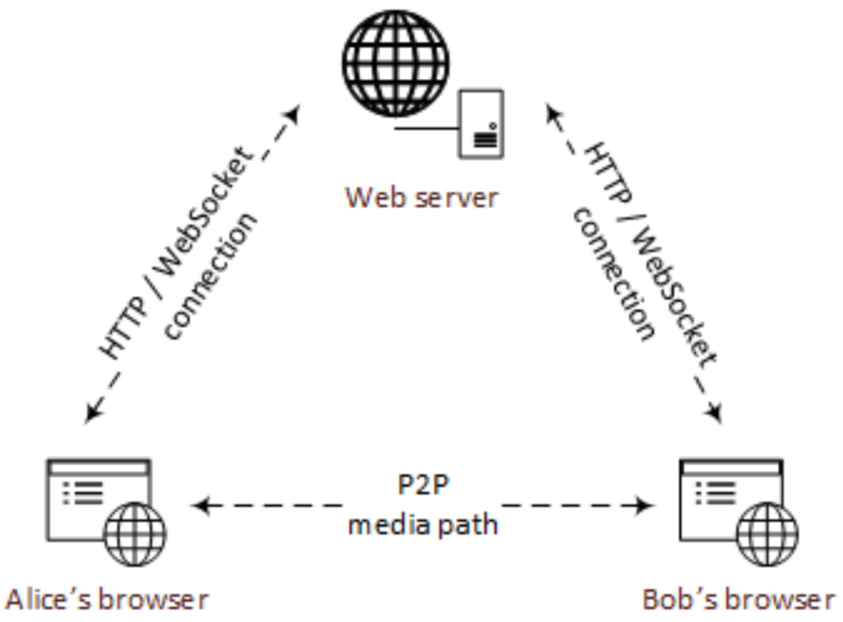


Figure 6.1: WebRTC triangle network architecture

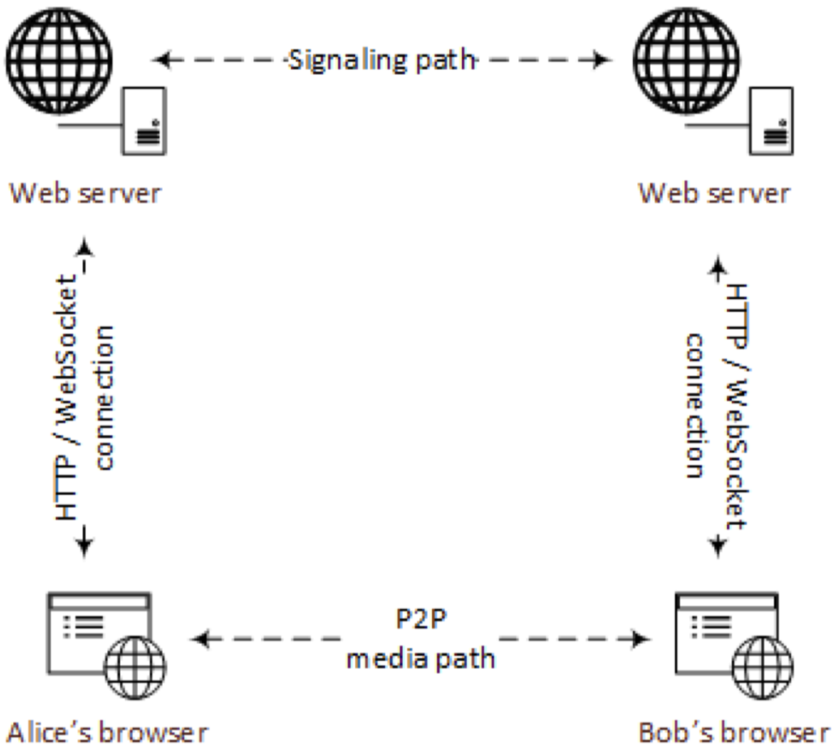


FIGURE 6.2: WebRTC trapezoid network architecture

context of WebRTC, the ICE framework offers a solution to the aforementioned problems. Specifically, as depicted in Figure 6.3, ICE enables the caller's and callee's device, to be informed about their public and private IP (IP:port), by making requests to a STUN server. In case of using the same WebRTC service, the peers will most probably use a common STUN server; otherwise, different STUN servers will serve them.

After the peers become aware of their IP:port address as seen from the STUN's perspective, they can easily establish a direct communication over (preferably) UDP or TCP, if UDP fails. If at least one of them is unable to do so in the presence of one or more of the aforementioned network obstacles, say, a non-STUN compatible NAT, then a TURN server can be used as a fallback mechanism. As illustrated in Figure 6.4, this server acts as an intermediate node to relay data packets between the peers. Note that, typically, a TURN server offers STUN functionality too. Moreover, due to the provision of real-time voice and video services, WebRTC preferably runs over UDP. Thus, in addition, "UDP hole punching" may be needed for establishing a direct connection between the peers successfully.

Browsers supporting WebRTC can make STUN requests and get informed about the

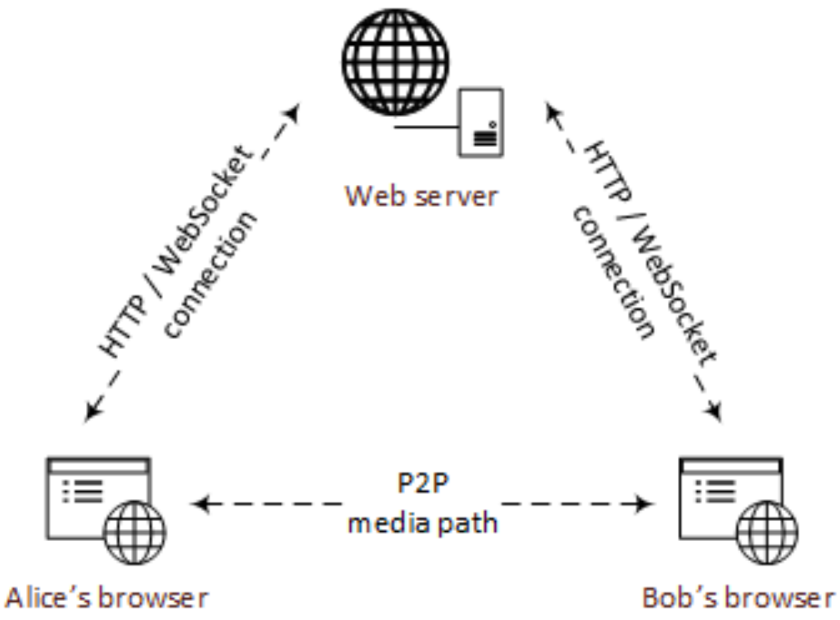


FIGURE 6.3: Using a STUN server.

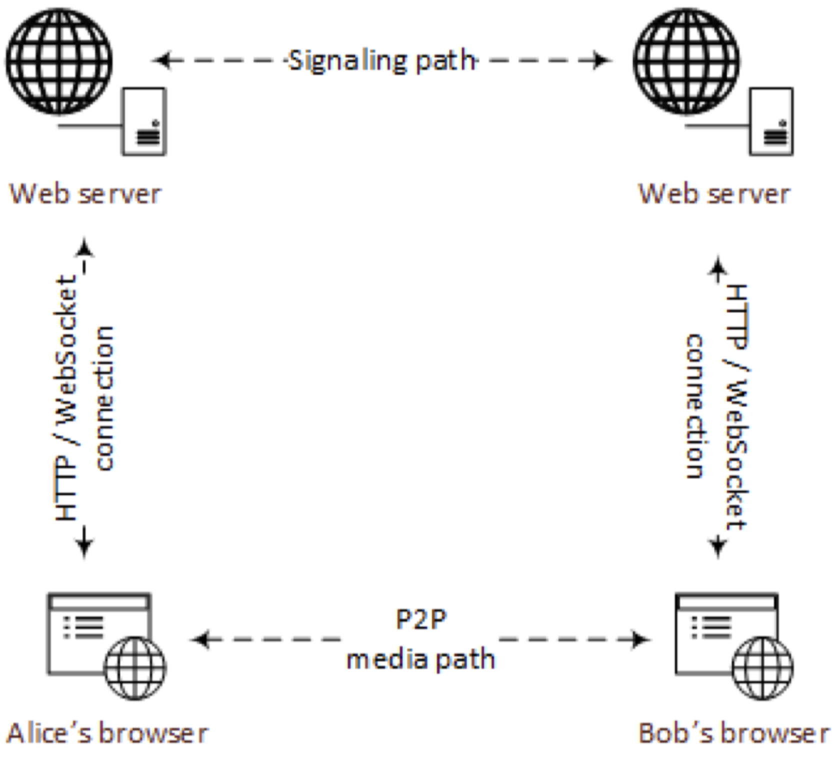


FIGURE 6.4: Using a TURN server.

actual private and public IP address of the end-user before passing this information to the JavaScript API. To initialize a connection to a remote peer, the local browser will use the RTCPeerConnection interface. Plainly, an RTCPeerConnection represents a WebRTC connection between the local machine and a remote peer, providing all the necessary methods to the user's browser to connect to a remote peer and then maintain, manage, and monitor the established connection.

To demonstrate a common scenario, we use an example where a client is behind a symmetric NAT and thus, TURN is used instead of STUN. In such a case, the caller needs to ask the TURN server to allocate some of its resources for them, as the TURN server will be used as a relay to contact the other peer [121]. In the log entry in Listing 6.1, one can easily observe the information that the TURN server logs after the client initiated an "Allocate transaction", using an Allocate type of request. First, a time handler responsible for the process termination starts. The default lifetime of an allocation is 10 min, but the actual time is defined in the initial Allocate request [120]. Later on, the server logs the public IP and the port of the client for completing the allocation for this particular client, as presented on the "remote" information, while "local" information is the public IP and the port of the TURN server. Following a successful allocation process, the initiator can either keep the connection alive using refresh requests, or terminate it.

```
client_to_be_allocated_timeout_handler:start
shutdown_client_connection:start
session 00100001: close (2nd stage), user
realm <> origin <>, local 198.201.166.150:3478
remote 31.132.100.72:1348, reason: allocation
watchdog determined stale session state
shutdown_client_connection:end
cliend_to_be_allocated_timeout_handler:end
```

Listing 6.1: TURN server log

6.3 WebRTC background

Before explaining how a malicious user can exploit any browser offering WebRTC capabilities, we should detail the way WebRTC works. Let us consider an example with two users, Alice and Bob, who both use a WebRTC client, and Alice wishes to call Bob, as depicted in Figure 6.3. To create a P2P connection, Alice needs to generate an SDP offer. Note that SDP [26] is a format protocol intended to describe media details, transport addresses, and any other session description metadata to the peers. In our example, the SDP message is sent between the peers, using HTTP or WebSockets. That is, Alice and Bob have to exchange SDP data using the existing signaling channel for negotiating audio and video media parameters, such as media codecs and video resolution. Additional information regarding setting up, updating, and tearing down the WebRTC session, such as transport addresses and related metadata, will also be exchanged using SDP. This process is initiated along with the creation of an RTCPeerConnection object as explained in Section 6.2.

The SDP offer includes information about Alice's network connection, and as Alice's data may follow different communication paths before they reach the outside, this offer must contain the shortest and most efficient network path. This is where ICE is applied to help gather the different nodes of Alice's network, known as ICE candidates. The reason why the SDP offer is vital in this step is that it includes all the ICE candidates that Bob can use for communicating with Alice, along with others, important for the call information.

For building the list of ICE candidates, Alice makes a STUN request to a STUN server, which is expected to respond with her public IP address and port(s). At this point, the STUN server allows Alice's client to discover its public IP address and, say, the type of NAT it is behind from, by sending back to her a success response which contains all the appropriate information.

All in all, every node that forwards Alice's traffic from the local network to the Internet is considered an ICE candidate, and the whole process of IP and port identification is called ICE candidate gathering. There are three types of candidates:

i. **Host Candidate:** Contains the private IP address and local UDP and TCP ports which are associated with the user's local network interface. They are generated by the client itself.

ii. **Server Reflexive Candidate:** Contains the public IP address and UDP and TCP port of the user that is returned by the STUN server. In contrast to Host Candidate, the client sends query messages to the STUN server, which will pass through the NAT, creating a NAT binding that is a public-private IP address mapping. The response contains the public IP and port (IP:port) generated for the binding.

iii. **Relayed Candidate:** Similar to Server Reflexive Candidate, this type of candidate contains the translated public address of the user; however, the NAT binding is obtained by a TURN server, instead of a STUN.

An example of an SDP message following a "RTCPeerConnection" initiation procedure is presented in Listing 6.2. As observed from the listing, three different candidates were gathered. The first consists of the user's interface (private) IP address, along with the port the browser is listening to. In this case, if the remote peer, i.e., the callee, is part of the same intranet, this candidate will be selected, and the RTP, which is the protocol used for delivering audio and video, will run over UDP. The second candidate is identical to the first; however, TCP will be used instead of UDP. The last candidate is a reflexive one, as a STUN server was used to return the user's public IP:port. Therefore, it is a lower priority candidate, but it will be nonetheless probed if the peers reside in different networks. It is to be noted that the returned number of candidates could be augmented depending on whether the peer's machine is multi-homed or is connected via a VPN or an anonymity network.

```
a=candidate:0 1 UDP 2122252543 remote 31.132.100.72:13481 typ host
a=candidate:2 1 TCP 2105524479 remote 31.132.100.72:13482 typ host tcptype active
a=candidate:1 1 UDP 1686052863 remote 31.132.100.72:13483 typ srflx raddr
    remote 31.132.100.72:13484
```

Listing 6.2: List of ICE Candidates

After receiving Alice's SDP message, Bob needs to follow the same procedure to produce an SDP message as a response and send it back to Alice. After the two users obtain the appropriate information via the exchange of SDP messages, they both perform a number of connectivity checks. These checks comprise a series of STUN requests to each IP and port pair of the ICE candidates. If the other party's browser responds, the originated request is considered successful and the checked pair is marked as valid. After the checks are completed, the ICE algorithm will decide which of the valid candidate pairs are the most efficient, based on a list of rules, including the IP address family, the utilization of media intermediary, say, a TURN server, and the peer's connection security, i.e., the use of VPN or not. If no valid pair is found, then the peers will make TURN requests to the TURN server in order to use it as an intermediate. After the connection has been established, the TURN server will forward any packets transferred between the two parties, as depicted in Figure 6.4. More specifically, in case of using a TURN server, TURN will remain in the media path, even after the connection has been established, and will act as a relay between the two ends.

6.4 IP Disclosure

6.4.1 Adversary Model

The present study mainly targets the privacy threats while users are operating a typical web browser. Consequently, adversaries are individuals or organizations which attempt to compromise the privacy of any Internet user. We consider an adversary with the following capabilities: (1) they can intercept, modify, or inject any message in the public communication channels; (2) they adhere to all cryptographic assumptions, e.g., an adversary is unable to decrypt an encrypted message without knowledge of the decryption key; (3) they are able to set up and operate their own STUN/TURN server; (4) they are able to inject JavaScript code to any webpage; (5) they can lure individuals into visiting certain webpages by, say, exercising social engineering techniques. An extensive threat model for WebRTC can also be found in [124].

6.4.2 Problem Statement

Bear in mind that there are two possible ways for the browser to execute a piece of JavaScript code on a webpage: (a) either immediately in the order it appears, or (b) wait for a triggered event to be executed. In any case, injected JavaScript code will always be executed transparently in the background without the user's permission. Taking this into account, a webpage that supports WebRTC will execute some WebRTC API calls in the background and will try to initiate a call to the specified remote party. A code snippet responsible for such an action is given in Listing 6.3.

```
//initialize list of ICE servers
var servers = {iceServers:
 [{url:"stun:stunserver.org",
 "credential":"my_password"}]};

//construct a new RTCPeerConnection
var rtc = new RTCPeerConnection(
 servers);

//Event Handler for new ICE candidate
rtc.onicecandidate = function (ice) {
   if (ice.candidate) {
   //Returns a DOMString describing
   //the candidate in detail
      console.log(ice.candidate.candidate);
   }
};

//create a bogus data channel
rtc.createDataChannel("");
//create an offer sdp
rtc.createOffer(function (result) {
   //trigger the stun server request
   rtc.setLocalDescription(result,
   function () { }, function () { });
}, function () { });
```

Listing 6.3: Initializing a WebRTC call

This small piece of JavaScript code actually configures the STUN server that will be used for the initiation of the call and then constructs an RTCPeerConnection object, which executes the STUN or TURN requests. Precisely, the code first initializes a JavaScript object with a list of possible ICE servers that could be used during the procedure, adding for each candidate the IP/domain of the server and the user's password, if needed (lines 2 to 4). As observed, in line 3, the URI scheme "stun" is used, followed by the server domain, while the credential field in line 4 carries the authentication password.

In most cases, the selected STUN server will be one of the available servers that are offered by organizations like Google, e.g., stun.l.google.com:19302. In lines 7 to 8 of Listing 6.3, the RTCPeerConnection object gets instantiated and saved in the "servers" variable. The "onicecandidate" in line 11 is an EventHandler, where a function is specified, and it is called when an "icecandidate" event occurs, namely, when an ICE candidate is discovered. This is the function via which the caller/callee will retrieve any information about herself, namely any local or external IPs they are using. Finally, the "createDataChannel" function in line 20 creates a new communication channel with the callee, over which any data can be transmitted. On the other hand, the "createOffer" function in line 22 initializes an SDP offer, with the purpose of creating a WebRTC connection with the callee. After the JavaScript code is executed, the peer will communicate with the declared STUN server and exchange data with it.

However, setting up and running a STUN server is a task that even a script-kiddie can easily perform, aiming to expose the IP address of any user who executes the JavaScript code in Listing 6.3. Therefore, if a threat actor installs a STUN server on the "malicious_stun_server.org" domain, they could easily replace lines 2 to 4 of Listing 6.3 with lines in Listing 6.4:

```
var servers = {iceServers:
 [{url:"stun:eve@malicious_stun_server.org",
 "credential":"my_password"}]};
```

Listing 6.4: Choosing the preferred STUN or TURN server

By doing so, the attacker is able to force the caller's browser to use a STUN or TURN server controlled by them. Now, the aggressor could inject the script in the source code of a webpage that they own or have compromised, and as a result any browser visiting this page will fain executing a STUN request to the STUN server controlled by the attacker. For instance, think of a watering hole variant attack scenario or the use of specially selected advertisements displayed to the victim. Accordingly, any webpage containing the JavaScript code listed in Listing 6.3 and the corresponding STUN server can instantly acquire the public IP:port of the peer. Bear in mind that the JavaScript code becomes knowledgeable of all ICE candidates of the peer, and as each one of them is associated with a peer's private interface, the mapping of the user's connection topology is feasible. The script injection mentioned above can be performed using a Cross-Site Scripting (XSS) attack. This is feasible on a web application where user input is directly included in the webpage without previous validation or encoding. If the XSS attack is successful and the script executes any STUN requests, then those requests will not be blocked by extensions such as Adblock [125], given that STUN requests are made outside the normal XMLHttpRequest procedure.

While the main role of a STUN server is informing the peer of their public IP address and port as seen from its viewpoint, there are no completely safe ways for a user to hide their public or local IP address, even if VPN or an anonymity network like Tor is employed. Bear in mind that, when a user connects to a VPN, a virtual network interface is created to handle all the network traffic, redirecting any data inside the VPN tunnel. Linux-based systems use the network tunnel (TUN) or network tap (TAP) [126] to provide packet reception and transmission when VPN is enabled. When a new virtual network interface is created, two or more adapters will be present, and the Operating System (OS) has to decide which one will use in order to send traffic. The selection is based on each adapter properties, which, for example, in MS Windows OS are determined by a feature called Automatic Metric.

After this process, the VPN adapter will be selected as default for achieving secure and confidential communication. However, the previous main network interface, that is the Ethernet or Wi-Fi adapter, is still active. In every OS, the software can choose any active network interface for communication, but usually the default one selected by the OS is used. Therefore, in the context of this adapter selection process, a STUN request may also reveal the end-user's public IP as allocated by the ISP to the JavaScript running on the webpage. Current trials over well-known VPN services, including TunnelBear [127] and Hotspot Shield [128], verify this situation. Although the MS Windows OS is the main victim of such a flaw, Mac and Linux systems may be vulnerable as well, depending on the user's VPN client and how it is configured [129]. In a similar research [130], the authors claim that both Firefox and Chrome are vulnerable regardless of the underlying OS, with iOS being the only exception. They also mention that the only browsers that support WebRTC, but immune to this vulnerability, are Safari on Mac and Microsoft Edge on Windows.

A straightforward solution for protecting users' IP addresses would be the use of a VPN firewall, but still without ensuring full anonymity. Generally, if a VPN connection fails, any data will be sent unsecured. A VPN firewall will ensure that any outgoing traffic will be sent through the established VPN tunnel, forbidding any connection in case the tunnel breaks down for any reason. In addition, a VPN firewall offers more benefits, including tight firewall rules and defeating the shared VPN/Tor server leak bug [131]. However, even this approach will not ensure full anonymity. This is because some browsers can store data from past user activities, like previously opened tabs. In that case, if a tab is opened before the user connects to the VPN, the public IP of the user will be cached in memory. In addition, even if the public IP address is not leaked, the end-user's local IP is still left unprotected, since JavaScript is able to identify it. As a result, the server will retrieve and log all the available information that could be used to identify the user.

Based on the WebRTC API [132], apart from public and private IP addresses, media information can also be retrieved by the "MediaDeviceInfo" interface. Precisely, WebRTC can leak the existence of any microphone and camera that the user may utilize. After obtaining an array of the available "MediaDeviceInfo" objects, one per media device, the browser can collect information for input and output devices though the object's properties. For instance, the "kind" property will return an enumerated value that is either "videoinput", "audioinput" or "audiooutput", while the "label" property will return a DOMString describing the device. By default, the "label" is an empty string; however if one or more media streams are active, or if the user granted persistent permissions to the browser, then the DOMString will contain information about any employed peripheral device, including its name and type, e.g., "External USB Webcam". For example, the enumeration of "MediaDeviceInfo" array may produce an output similar to that in Listing 6.5. Thus, if two audio and one video input were attached to the client's machine, the kind of each input along with its ID would be outputted.

```
videoinput: id = 56430b9613fcb1ac822fd53a6c25
audioinput: id = 321668ae7aebb94d0d2b90bee995
audioinput: id = 0fd5b84ae87e3420486e2e2c4d9f
```

Listing 6.5: Example of MediaDeviceInfo output

On the other hand, each item of Listing 6.5 would contain all the available information in case one or more media streams were active or permissions were granted by the user, as presented in Listing 6.6. That is, for each media input, apart from the device's ID and its type, the "label" describing the device and a "groupID" would also be returned. The "groupID" represents the group of the device, and two devices have the same group identifier if they are part of the same physical device, for example, if both a camera and microphone belong to the same monitor.

```
kind: videoinput
label: Integrated Webcam (1bcf:28b0)
deviceId: 56430b9613fcb1ac822fd53a6c25
groupId: 85632d755cfb1dfb30228124124ec

kind: audioinput
label: Microphone (Realtek Audio)
deviceId: 0fd5b84ae87e3420486e2e2c4d9f
groupId: 40756e2116ee3d4d75183136bd03e

kind: audioinput
label: Headset (HD 4.40BT Hands-Free AG Audio)
deviceId: 321668ae7aebb94d0d2b90bee995
groupId: b0d25385669f2a63bfe9d556fee46
```

LISTING 6.6: MediaDeviceInfo output with granted permissions

6.5 Dealing with IP Leaks

As discussed in Section 6.2, a browser informs the actual WebRTC application about the location of STUN and TURN servers using the RTCPeerConnection object. The peer can later use this object to connect to the STUN server, get informed about its own IP addresses through this connection, and create SDP offers for initializing a call. It is therefore straightforward that a webpage that is not destined to execute WebRTC requests should not use by any means the RTCPeerConnection object. In that case, if a binding request to a STUN server takes place, then the user should be informed that the visited website needs to access private information (the public IP address) in order to use WebRTC.

Taking the latest Android update as an example, a "runtime" permission system would be the ideal approach against such kind of actions. More precisely, a user should be informed that the webpage is trying to access their IP address information to initiate a WebRTC call and the exact time the user is placing the call. A similar type of permission request is used in some browsers, like Chrome, to inform the user that the current webpage is trying to access their location or wishes to send notifications. In such circumstances, if the user approves the request, then the browser will be able to get the appropriate user's information. Contrariwise, if the user rejects the request, the action should get blocked, also meaning they will be unable to make a call, if that is the case.

To address the above issue and detect whether a webpage attempts to execute a STUN or TURN request, we propose two diverse kinds of solutions, which are implemented differently, but, in essence, use the same detection technique. The first approach is a browser extension, which uses a preload mechanism to prevent such JavaScript calls before the actual HTML DOM loading starts. The second uses a gateway, either local or third-party, to inspect the JavaScript code the user is about to download.

6.5.1 Browser Extension

The implemented browser extension can prevent any WebRTC requests before they are executed by a webpage. This is achieved by dereferencing any WebRTC objects that could be used by a webpage for executing STUN or TURN requests. Initially, when a user visits a webpage, the WebRTC requests are blocked, while the rest of the page loads normally without any user interruptions. After the webpage loading is completed, the extension inspects for any suspicious WebRTC requests, and if any is detected, the browser asks for WebRTC permissions from the user. If the user wishes to use WebRTC capabilities, the extension will restore any WebRTC objects that were previously dereferenced, allowing the user to initiate a WebRTC call. As browser extensions are implemented differently on each browser, the drawback of this approach is that a different extension per browser is needed. As detailed in Section 6.7, our extension has been implemented for use with the popular Chrome browser.

6.5.2 Gateway

A more generic solution is to implement a gateway, which examines the webpage source code before it is delivered to the user's browser. The user has the ability to choose the gateway that will be used, and thus, the gateway is considered trusted. This way, a user would again be informed by the trusted gateway if the website that they would like to visit is trying to make a WebRTC request. If the user is unaware about that website actions, they may block the relative JavaScript or blacklist the website by storing the domain on the gateway's list. Then, this list will be checked every time a user visits a website, so the latter will be prevented from loading if its domain already exists in the list. In contrast to the browser extension, any check will be conducted by the gateway before the webpage loads to the user, and thus, an extra delay will be added to the load time of the webpage. Those extra delays are presented later in Section 6.7.

6.6 Implementation

The source code of both the implemented solutions is correspondingly given under the European Union Public License (EUPL) [133] at: `https://github.com/IncredibleMe/WebRTC-IP-Leak-Chrome-Extension` and `https://github.com/IncredibleMe/WebRTC-IP-Leak-Gateways`.

6.6.1 Browser Extension Implementation

As already pointed out in Section 6.4, STUN requests are made outside the normal XMLHttpRequest procedure, so a common extension responsible for ad blocking, for instance, is unable to detect such requests. The implemented extension should prevent the webpage for such possible STUN requests by inspecting any JavaScript code. The main hurdle is that any JavaScript residing on the source code will be immediately executed after the webpage has been downloaded, so blocking is infeasible, and, as a result, a different approach should be followed. While implementing a browser extension, lots of information about it should be declared in the "manifest.json" file, such as the name of the extension, its version and its permissions. In the manifest file, one can also force the extension to be executed as fast as possible, using the "run_at : document_start" option. We also deployed the same “run_at” option, since we need to inject a JavaScript piece of code preventing WebRTC requests in the webpage’s source code before any other DOM is constructed, or any other script is executed.

This option gives us the ability to inject JavaScript code as a script at the top of the website’s HTML document and execute it immediately after the DOMContentLoaded event. Bear in mind that in an HTML document all scripts are treated the same, thus, they are executed in the order they appear in the document. Having our defensive script placed on top of everything else, every malicious action that may appear later in the document can be prevented.

The next step is to create a JavaScript element and backup up the “RTCPeerConnection” object value, along with the compatible objects for Firefox and Chrome, “mozRTCPeer-Connection” and “webkitRTCPeerConnection” respectively, to some temporary values. After that, one can set those object values to null, so any later scripts will be unable to use them. By using this technique, we are able to break any malicious code during execution. In addition, if any use of those objects is detected in the source code, the user will be informed, and if they wish to proceed, the script can revert on-the-fly the initial value to the RTCPeerConnection object.

6.6.2 Gateway Implementation

The gateway can either be a SOCKS [103] or an HTTP proxy, and while a plethora of programming languages can be used to develop such a software, there is no actual limitation about the implementation details. However, the main question that needs to be answered is what is the delay that this process adds to the webpage loading process. With that in mind, a high-performance programming language should be used, making the code-examining process a really quick task. For the sake of this work, we developed two different versions of an HTTP proxy: one written in C++ and one in Golang, two of the most high-performance server programming languages. Using the aforementioned implementations, we measured the time penalty produced by different webpages of various sizes (data volume) and characteristics. To dereference the induced time penalty from any network latencies, the gateway runs in the same machine where the user is located.

No less important, the gateway is able to examine the content of any webpage using either HTTP or HTTPS. Precisely, for HTTPS traffic, the gateway works as a transparent proxy, which acts as a MiTM at the TLS tunnel, therefore it is able to read the traffic even if the content is encrypted with TLS. Typically, a client would connect to a proxy by executing a CONNECT request, for instance "CONNECT example.com:443 HTTP/1.1", and open a P2P connection between the client and the destination web server.

Using a modern programming language, the parsing of an average webpage's source code will be completed in a few milliseconds. Table 6.1 presents a comparison of Golang and C++ when it comes to processing text files of diverse sizes and characteristics. As observed, the results pertaining to both languages are almost identical, and even on much larger file sizes than the average webpage size, which is ≈3 MB [134]. Moreover, the searching of certain keywords can be achieved in less than a second on files even 100 MB large, that is ≈534 ms and ≈513 ms on average for C++ and Golang, correspondingly. We should also mention that on each search the whole text file was examined, reaching to the conclusion that the document is safe.

However, if a malicious JavaScript was included in a document, the result would be obtained before reaching the end of the file, and thus the delay is expected to be even lesser. It is to noted that the results reported in Table 6.1 have been acquired after conducting 500 searches per different text file size and computing the mean value. As observed from the Table 6.1, the average value is almost half a second for files even thirty times as big as the size of the average webpage. This leads us to conclude that most of the delay of this gateway-based approach is expected to be mainly due to the bandwidth of the gateway and the network speed, rather than the actual examination of the webpage's source code.

Filesize (MB)	Delay (ms)	
	Golang	C++
0.5	3	3
1	6	6
5	28	28
10	56	58
50	283	269
100	513	534

Table 6.1: Performance comparison of C++ and Golang in file processing

6.7 Evaluation

6.7.1 Browser Extension Evaluation

As already pointed out in Section 6.6, to assess the performance of the proposed browser extension, we created a proof-of-concept implementation for the Chrome browser running on Ubuntu Linux 18.10 on a laptop machine equipped with an Intel Core i7-6700HQ 3.4 GHz CPU and 16 GB of RAM. Typically, Chrome extensions tend to be a heavy burden to the overall performance of the browser, as they currently work for each active tab. For instance, the popular Adblock extension consumes ≈100 MB of memory in Chrome [135]. We used Chrome's task manager to measure the CPU and memory usage of our extension when visiting a webpage. Regarding CPU usage while the extension loads, it only happens once, and according to our measurements was negligible, i.e., less than 0.1%. Upon the examination of the webpage, the extension consumed ≈0.15% of the available CPU, while the memory footprint reached ≈3 and 5 MB for websites having correspondingly a size of ≈3 and 10 MB. This is due to the fact that the extension initially blocks any STUN or TURN requests before they execute, rather than checking the whole webpage's source code for such requests.

6.7.2 Gateway Evaluation

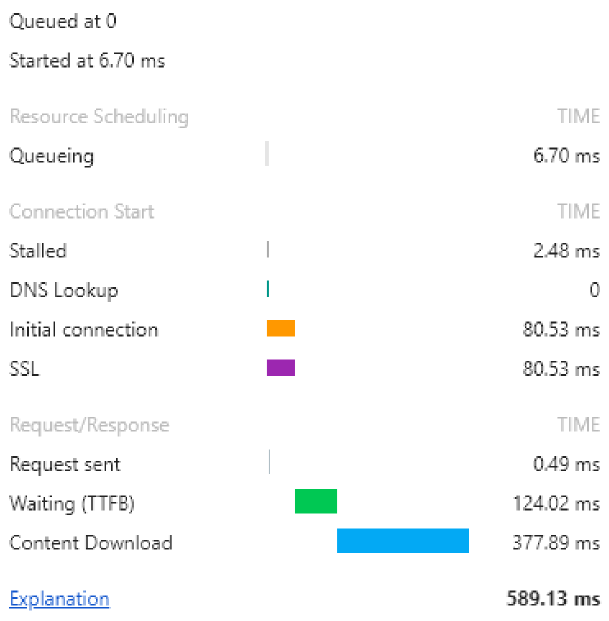
To measure the time penalty induced by the examination of the webpage source code, we used Chrome Developers Tools [136] and made measurements during webpage loading, both with and without the use of the gateway. To approximate the actual time it takes for each HTTP/HTTPS GET request to be examined before it is sent back to the user, we need to identify the different components that contribute to the final webpage load time. To this end, we employed the Resource Timing API [137], which provides a way to obtain detailed network timing data about the resource loading. The interface splits the webpage's loading time in different parts, each one representing a network event, such as DNS lookup delay, response start and end times, as well as the actual content downloading. Modern browsers use Resource Timing API [137], which is usually embedded by default, to offer developers the ability to calculate a web request load and serving time. For instance, in Chrome, a user can observe detailed information about the time each request needs to be fulfilled, using the Network Analysis Tool, which is part of the Chrome Developers Tools, placed in the Network panel. An example of a request's timing breakdown is presented in Figure 6.5.

In the following, we succinctly describe each of the webpage phases for the sake of understanding the whole lifecycle of a web request. Before each request is sent to its final destination it may need to be delayed for several reasons, like other higher priority requests waiting to be served, the allowed number of TCP connections has been exceeded, or the disk cache is full. For each one of the aforementioned reasons, a request may be queued and even stalled. The "DNS Lookup" delay, comprises the time spent for a request to perform a DNS lookup for a certain network domain in a webpage. However, if the same webpage along with the carrying domains has been visited in the past, most or even all of its domains already exist in the DNS cache of the browser, and thus the DNS Lookup delay will be negligible. The connection process with the remote web server is measured by the "Initial connection" metric, which represents the time the connection needs to be initialized, including the TCP handshake and the needed retries, and, if applicable, the negotiation of TLS, that is, the TLS handshake between the client and the server.

Finally, the "SSL" is the time spent until the TLS handshake protocol completes, that is, the "change cipher spec" message, which is used to switch to symmetric key protection. After the connection is established, a number of other phases take place. The "Request Sent" is the actual time spent for the request to be sent, overcoming any network issues, which usually is negligible. The subsequent two phases, are probably the most significant ones when it comes to the delay of the request. The "Waiting" represents the time spent waiting for the initial response to be processed, also known as Time to First Byte (TTFB). This pertains to the number of milliseconds it takes for a browser to accept the first byte of the response after a request has been sent to a web server, that is, the latency of a round trip to the server. Last but not least, the "Content Download" is the actual time spent before the client receives the complete web content from the server, a phase which is obviously the most critical, as most of the web request's time is consumed here.

It is expected that, as the whole webpage source is downloaded and parsed by our proxy, the size of the webpage's source code will probably affect the final delay. However, the browser can also employ several techniques to speed up the content downloading process, including caching. Namely, the browser cache is being used to store website documents like HTML files, CSS style sheets and JavaScript code for avoiding downloading it again in future website visits. This means that if a user revisits a given webpage, the browser will try to download only new content, loading the rest from its cache. The previous action, will naturally lead to insignificant delays when it comes to security checks on a webpage, since the content has already been checked in the (near) past.

As already mentioned, the average webpage size is ≈3 MB, which makes source code parsing a really quick task. It is also worth mentioning that the heavier part of the webpage is being bound by images or other multimedia type objects, while our interest


Queued at 0
Started at 6.70 ms

Resource Scheduling	TIME
Queueing	6.70 ms

Connection Start	TIME
Stalled	2.48 ms
DNS Lookup	0
Initial connection	80.53 ms
SSL	80.53 ms

Request/Response	TIME
Request sent	0.49 ms
Waiting (TTFB)	124.02 ms
Content Download	377.89 ms

Explanation **589.13 ms**

Figure 6.5: Preview a timing breakdown with Chrome

lies solely on the JavaScript source code possibly existing in the webpage. With that in mind, the proxy needs to only deal with the parsing of the source code, whose actual size will be significantly reduced, hence introducing less delay.

The gateways were implemented in C++ and Golang and installed on Ubuntu Linux 18.10 on a laptop machine equipped with an Intel Core i7-6700HQ 3.4 GHz CPU and 16 GB of RAM. The Internet connection used provided a bandwidth of 34 Mbps downstream and 28 Mbps upstream. The laptop machine was connected to the Internet via an IEEE 802.11n wireless access point. We also configured a Coturn server [138], which is an open-source TURN/STUN server for Linux. Additionally, we created a webpage, which executes requests to our STUN server, and thus, we were able to examine the exact log entry at the server side.

We compared the two gateway implementations and logged key metrics that characterize a website visit. The acquired results are summarized in Table 6.2. As observed from the table, we tested our gateway implementations against some of the most popular websites which mandate HTTPS, as well as websites that are plain HTTP.

The time each solution needs to download all the (possibly compressed) contents of the webpage is naturally dependent upon its size. The "content size" column in the table represents the actual size of the resource, while "response size" column represents the number of MBytes transferred on the wire or the wireless medium. As long as most of the modern web servers use compression to the response body before they are sent over the network, the size of the transferred data are significantly reduced. The "load time" column is the total time delay that the webpage consumes for loading all of its contents, including JavaScript and CSS code, images, as well as the main HTML document. In detail, the "load time" is divided into the "DOM content load time" and the webpage rendering. The first represents the time that the webpage needs to build the DOM. Namely, when the browser receives the HTML document from the server, it stores it into the local memory and automatically parses it for creating the DOM tree. During this time, any synchronous scripts will be executed and static assets, such as images saved on the server side, will be downloaded as well. The webpage rendering phase is the amount of time that the browser needs to display the content of the webpage to the end-user.

Table 6.2: Performance comparison of C++ and Golang proxy implementations vis-à-vis to the typical proxyless setting. Content and response current approximate sizes are in MB, while load times are in seconds and represented as a 95% confidence interval over 50 measurements done in different days and times.

URL	Content Size	Response Size	Type of Proxy	Load Time	DOM Content Load Time
https://wikipedia.org	0.22	0.09	C++	3.91 ± 1.36	5.18 ± 1.27
			Golang	4.05 ± 1.37	4.89 ± 1.33
			Proxyless	2.41 ± 1.26	2.54 ± 1.29
https://office.com	1.46	0.77	C++	1.69 ± 0.28	2.43 ± 0.31
			Golang	1.65 ± 0.25	2.31 ± 0.38
			Proxyless	1.13 ± 0.16	1.65 ± 0.25
https://ebay.com	2.66	1.25	C++	2.51 ± 0.26	3.21 ± 0.41
			Golang	2.15 ± 0.22	2.57 ± 0.27
			Proxyless	1.78 ± 0.14	2.41 ± 0.34
https://vk.com	3.27	1.16	C++	4.48 ± 0.80	6.16 ± 0.77
			Golang	4.09 ± 0.94	5.70 ± 0.85
			Proxyless	3.09 ± 0.64	4.32 ± 0.85
https://instagram.com	3.44	1.11	C++	2.39 ± 0.19	3.93 ± 0.34
			Golang	2.14 ± 0.15	3.66 ± 0.30
			Proxyless	1.11 ± 0.16	2.51 ± 0.31
https://aliexpress.com	5.40	3.12	C++ proxy	4.49 ± 0.24	7.45 ± 0.52
			Golang	2.78 ± 0.21	7.40 ± 0.96
			Proxyless	1.92 ± 0.13	6.17 ± 0.75
https://yahoo.com	0.31	0.14	C++	1.57 ± 0.18	2.00 ± 0.23
			Golang	1.78 ± 0.16	2.62 ± 0.22
			Proxyless	0.66 ± 0.14	0.87 ± 0.29
https://netflix.com	5.37	2.24	C++	3.97 ± 0.27	4.74 ± 0.40
			Golang	3.38 ± 0.25	4.26 ± 0.34
			Proxyless	2.47 ± 0.32	2.86 ± 0.52
https://amazon.com	9.84	2.93	C++	2.80 ± 0.24	3.02 ± 0.35
			Golang	3.57 ± 0.22	4.85 ± 0.38
			Proxyless	1.41 ± 0.19	1.93 ± 0.37
https://twitch.tv	22.73	17.46	C++	5.60 ± 0.56	13.09 ± 2.08
			Golang proxy	5.1 ± 0.56	12.45 ± 1.99
			Proxyless	3.76 ± 0.58	10.74 ± 2.14
http://wired.com	20.20	11.61	C++	6.09 ± 1.53	21.18 ± 5.54
			Golang	5.76 ± 1.54	20.77 ± 5.52
			Proxyless	4.83 ± 1.49	19.57 ± 5.44
http://nba.com	23.60	14.50	C++	6.82 ± 1.06	19.10 ± 3.25
			Golang	7.52 ± 0.97	20.59 ± 3.09
			Proxyless	5.85 ± 0.95	17.29 ± 3.22
http://espn.com	1.56	0.55	C++	4.15 ± 0.74	8.64 ± 2.06
			Golang	3.77 ± 0.78	8.28 ± 2.13
			Proxyless	2.98 ± 0.79	7.31 ± 1.89

However, in that stage, no extra delay will be added since the initial code will be sent directly to the client's browser.

As observed from Table 6.2, checking a website's intentions, in terms of JavaScript code, comes with a cost, although most of the time it will not be noticeable. Both C++ and Golang have proven good choices for a proxy implementation, producing negligible delay vis-à-vis a non-proxy scenario. Precisely, neglecting the webpage size factor, in the case of the C++ proxy, the average time across all the websites we tested was 6.25 sec with a standard deviation of 3.41 sec, while for the Golang one the corresponding times were

5.83 and 2.69 sec. These times compared to those of the proxyless configuration (4.18 sec and 1.65 sec) are rather insignificant. However, the delay becomes noticeable on sites almost 300% bigger than the average one, which in our table exceed the size of 8 MB. In addition, no major difference appears between HTTP and HTTPS websites, which means that the client's gateway acting as an "TLS Termination Proxy" adds a negligible delay. All in all, the most critical parameter is the "content size" of each website. The most significant time delay was observed on amazon.com, nba.com and espn.com whose size is among the biggest in Table 6.2. As expected, the larger the size of the source code to be examined, the greater the webpage's loading time.

Chapter 7

Conclusions and Future Directions

7.1 Conclusions

The main aim of the present PhD thesis is to emphasize the importance of ongoing research regarding privacy-enhancing technologies, specifically those pertaining to VoIP and other real-time communication protocols. That is, as these communication technologies continue to advance and proliferate, it is imperative to develop new privacy and security solutions capable of addressing emerging threats in a more holistic manner. In this context, through this study, we aim to build upon and contribute to the past and ongoing efforts towards enhancing the privacy and security of Internet services from an end-user viewpoint.

Altogether, considering the thesis' objectives outlined in Chapter 1, this work provides and evaluates a range of mechanisms aimed at safeguarding users' anonymity and privacy in both SIP and WebRTC protocols. An overview of the contributions of this PhD thesis in comparison to the existing literature is presented in Table 7.1.

Table 7.1: Overview of PhD thesis contributions and related publications

Objective	**Chapter**	**Contribution**	**Publication**
Obj. 1	4	Complete SIP message obfuscation: PrivaSIP over Tor	[52]
Obj. 2	5	OnionSIP: Preserving Privacy in SIP with Onion Routing	[4]
Obj. 3	6	Neither Denied nor Exposed: Fixing WebRTC Privacy Leaks	[139]

With reference to Table 7.1, the key contribution of this PhD thesis hinges upon the lack of privacy protections in SIP. Given that the latter protocol is plaintext by design, it necessitates external (add-on) mechanisms to safeguard end-users' privacy and anonymity. Excluding the proposed solutions by this PhD thesis, to the best of our knowledge, as detailed in Chapter 3, the only practical privacy-preserving solution so far is PrivaSIP. The latter preserves privacy at the application layer by encrypting sensitive SIP fields, and may be utilized in conjunction with layer 3 or 4 protocols, including TLS and IP Secure (IPSec), to offer privacy at the transport or network layer. However, thus far, there is no single scheme able to provide privacy and anonymity in a cross-layer fashion. To investigate this potential, the present thesis came with the idea of harnessing the power of popular anonymity Internet overlays, namely Tor and IP2, to fulfill the previously mentioned goal. To this end, we first evaluated the performance of PrivaSIP-over-Tor and over-I2P. From the derived results in Section 4.3, it is obvious the while such a scheme is powerful in terms of privacy, it introduces significant latency in SIP session establishment.

In the same direction and to mitigate the aforementioned issue, another contribution of this thesis is the design and implementation of a custom-tailored solution, which relies on the onion routing concept. The evaluation results regarding the proposed scheme demonstrated that our solution not only provides a high level of anonymity to the end-users, but also is considerably faster vis-à-vis PrivaSIP-over-Tor and SIP-over-I2P by at least 0.6 secs. In particular, we implemented two discrete solutions for the encryption of all the sensitive fields within a SIP message. The first leverages public-key encryption, which is supported by means of SIP proxy server self-signed or Private CA X.509 certificates, while the latter employs symmetric-key encryption.

SIP is also a key component of WebRTC, a modern VoIP technology that allows browsers to establish direct communication between devices, allowing users to engage in real-time audio and video communication without the need to download plugins or additional thrird-party software. Despite its many benefits, WebRTC raises significant privacy concerns. Namely, JavaScript code embedded in websites can expose users' IP addresses, even if they are utilizing VPNs or anonymization software like Tor. Precisely, to establish a connection, WebRTC includes a feature called "ICE" that gathers information about IP addresses and networks. The deployment of JavaScript code on a website provides malicious or semi-honest parties with the opportunity to exploit ICE, forcing end-users to unknowingly use the ICE protocol, revealing their actual IP address.

Through the above-mentioned prism, the current PhD thesis proposes and meticulously evaluated novel solutions to mitigate the aforementioned privacy invasion threat. Namely, a first solution comes in the form of a browser plugin that detects any STUN or TURN requests originating from the webpage that a user intends to visit. These requests can potentially expose both the user's private and public IP addresses. Therefore, our plugin temporarily deactivates WebRTC functionality, thereby notifying the end-user about the webpage's intentions. Given that the aforementioned solution is lightweight and effective, a distinct custom-tailored implementation is required for each browser. Bearing this in mind, we also proposed an alternative browser-independent approach that employs a gateway, which acts as a MiTM entity and proactively inspects the contents of the webpage, that is, before they are transmitted to the client. Overall, both of the proposed solutions are designed to notify users when a loading webpage attempts to establish a WebRTC connection without their consent. Finally, we evaluated the proposed solutions in terms of the delay penalty they introduce, and the results show that it is negligible, around 6 sec, as detailed in Section 6.7.

7.2 Future directions

This PhD thesis contributed a number of novel methods aimed at safeguarding the privacy and anonymity of VoIP users. We detailed the advantages and drawbacks of each method, with a particular focus on two aspects: the level of anonymity they provide and the introduced penalty in terms of time delay. Nevertheless, several research topics remain open for future investigation.

- A potential future direction involves conducting further experiments that closely approximate real SIP signaling traffic and exploring methods for reducing the $\approx$2 sec overhead caused by anonymization networks like Tor. Precisely, one could examine more complex VoIP infrastructures and scenarios with multiple users, services, and intermediate SIP elements. While the delay introduced by the Tor network can be considered bearable, improvements can be made through optimization, mitigating network limitations and improving user experience. To lower delays presented by Tor network, one could implement an automated way to create circuits using nodes with higher advertised bandwidth chosen by the Tor central directory. Comparable outcomes may be obtained through the utilization of ShorTor [140], an overlay for the Tor network that employs sophisticated routing techniques, claiming to decrease the latency among relays within a circuit.
- Another direction for future research is the incorporation of more anonymity-preserving solutions such as ShadowSocks [141]. The latter is a secure socks5 proxy that is predominantly utilized for evading censorship and overcoming geo-restrictions. It employs a proprietary encryption methodology that enhances its competence to evade detection and blockages compared to traditional proxies. ShadowSocks works by channeling Internet traffic through an encrypted tunnel to a remote server, which then dispatches the traffic to its intended destination. Although both VPN and ShadowSocks encrypt data, ShadowSocks offers a more streamlined approach. Unlike VPN, which employs several layers of slow and resource-heavy encryption protocols to entirely conceal traffic on its servers, ShadowSocks renders data unidentifiable to resemble HTTPS traffic, enabling unrestricted movement. As demonstrated in Chapters 4 and 5, it is feasible to funnel

SIP traffic through a socks5 proxy; hence, the utilization of ShadowSocks is unlikely to pose any issues in this regard. This approach could potentially enable SIP message anonymization, while incurring considerably lower delays than alternative solutions like VPN.

- Another future direction pertains to the security enhancement of the scheme proposed by this PhD thesis in Chapter 5. This can be done by implementing encryption either at each layer of the onion routing scheme or by integrating multi-hop routing to further obscure the origin of the SIP traffic.
- Regarding WebRTC, additional privacy-preserving solutions may be evaluated, including browser extensions that are capable of white-listing STUN and TURN servers. That is, the identification of trusted servers (similarly to the HTTP Strict Transport Security (HSTS) preload list for HTTPS-only websites) could prevent IP logging from non-authorized parties. Moreover, the use of browser fingerprinting techniques has the potential to assist in thwarting user tracking, including that which arises from WebRTC leaks. Certain browsers, like Firefox, have already integrated this mechanism, thereby providing users with a noteworthy degree of protection against fingerprinting. This can be accomplished by impeding third-party requests to domains that are known to partake in fingerprinting, such as malicious STUN/TURN servers.
- With reference to WebRTC, a further line of research may entail exploring additional security concerns that can arise when an unwitting user employs a fraudulent STUN or TURN server, beyond the exposure of the IP:port. These concerns may include session hijacking, MiTM, DoS, and DNS spoofing.

Bibliography

[1] Adam Rowe. 11 voip stats that prove the importance of the business tech, Sep 2022. URL https://tech.co/business-phone-systems/voip-statistics.

[2] Ilsa Godlovitch and Peter Kroon. Copper switch-off: European experience and practical considerations. Technical report, WIK-Consult White paper, 2020.

[3] I2p anonymous network, 2013. URL https://geti2p.net/en/.

[4] Alexandros Fakis, Georgios Karopoulos, and Georgios Kambourakis. Onionsip: Preserving privacy in sip with onion routing. *J. Univers. Comput. Sci.*, 23(10): 969–991, 2017.

[5] WebRTC 1.0: Real-time Communication Between Browsers. https://www.w3.org/TR/webrtc/, 2019. Accessed: 2019-06-21.

[6] Zahraa Sabra and Hassan Artail. Preserving anonymity and quality of service for voip applications over hybrid networks. In *MELECON 2014-2014 17th IEEE Mediterranean Electrotechnical Conference*, pages 421–425. IEEE, 2014.

[7] Mudhakar Srivatsa, Ling Liu, and Arun Iyengar. Preserving caller anonymity in voice-over-ip networks. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pages 50–63. IEEE, 2008.

[8] Maimun Rizal. *A Study of VoIP performance in anonymous network-The onion routing (Tor)*. PhD thesis, Georg-August-Universität Göttingen, 2014.

[9] Giorgos Karopoulos, Georgios Kambourakis, Stefanos Gritzalis, and Elisavet Konstantinou. A framework for identity privacy in SIP. *Journal of Network and Computer Applications*, 33(1):16–28, January 2010. ISSN 1084-8045. doi: 10.1016/j.jnca.2009.07.004. URL http://www.sciencedirect.com/science/article/pii/S1084804509001052.

[10] Giorgos Karopoulos, Georgios Kambourakis, and Stefanos Gritzalis. PrivaSIP: ad-hoc identity privacy in SIP. *Computer Standards & Interfaces*, 33(3):301–314, March 2011. ISSN 0920-5489. doi: 10.1016/j.csi.2010.07.002. URL http://www.sciencedirect.com/science/article/pii/S0920548910000942.

[11] Dimitris Geneiatakis, Tasos Dagiuklas, Georgios Kambourakis, Costas Lambrinoudakis, Stefanos Gritzalis, Sven Ehlert, and Dorgham Sisalem. Survey of security vulnerabilities in session initiation protocol. *IEEE Commun. Surv. Tutorials*, 8(1-4):68–81, 2006. doi: 10.1109/COMST.2006.253270. URL https://doi.org/10.1109/COMST.2006.253270.

[12] Dimitris Geneiatakis, Georgios Kambourakis, Costas Lambrinoudakis, Tasos Dagiuklas, and Stefanos Gritzalis. A framework for protecting a sip-based infrastructure against malformed message attacks. *Comput. Networks*, 51(10):2580–2593, 2007. doi: 10.1016/j.comnet.2006.11.014. URL https://doi.org/10.1016/j.comnet.2006.11.014.

[13] Sven Ehlert, Ge Zhang, Dimitris Geneiatakis, Georgios Kambourakis, Tasos Dagiuklas, Jirí Markl, and Dorgham Sisalem. Two layer denial of service prevention on SIP voip infrastructures. *Comput. Commun.*, 31(10):2443–2456, 2008. doi: 10.1016/j.comcom.2008.03.016. URL https://doi.org/10.1016/j.comcom.2008.03.016.

[14] Dimitris Geneiatakis, Georgios Kambourakis, Tasos Dagiuklas, Costas Lambrinoudakis, and Stefanos Gritzalis. A framework for detecting malformed messages in SIP networks. In *The 14th IEEE Workshop on Local and Metropolitan Area Networks, LANMAN 2005, Chania, Crete, Greece, September 18, 2005*. IEEE, 2005. doi: 10.1109/LANMAN.2005.1541543. URL https://doi.org/10.1109/LANMAN.2005.1541543.

[15] Dimitris Geneiatakis, Georgios Kambourakis, Tasos Dagiuklas, Costas Lambrinoudakis, and Stefanos Gritzalis. Sip security mechanisms: A state-of-the-art review. In *Proceedings of the Fifth International Network Conference (INC 2005)*, pages 147–155, 2005.

[16] Dimitris Geneiatakis, Georgios Kambourakis, C Lambrinoudakis, T Dagiuklas, and S Gritzalis. Sip message tampering: The sql code injection attack. In *Proceedings*

of 13th International Conference on Software, Telecommunications and Computer Networks (SoftCOM 2005), Split, Croatia, 2005.

[17] Dimitris Geneiatakis, Costas Lambrinoudakis, and Georgios Kambourakis. An ontology-based policy for deploying secure sip-based voip services. *Comput. Secur.*, 27(7-8):285–297, 2008. doi: 10.1016/j.cose.2008.07.002. URL https://doi.org/10.1016/j.cose.2008.07.002.

[18] Zisis Tsiatsikas, Dimitris Geneiatakis, Georgios Kambourakis, and Angelos D. Keromytis. An efficient and easily deployable method for dealing with dos in SIP services. *Comput. Commun.*, 57:50–63, 2015. doi: 10.1016/j.comcom.2014.11.002. URL https://doi.org/10.1016/j.comcom.2014.11.002.

[19] Zisis Tsiatsikas, Georgios Kambourakis, Dimitris Geneiatakis, and Hua Wang. The devil is in the detail: Sdp-driven malformed message attacks and mitigation in SIP ecosystems. *IEEE Access*, 7:2401–2417, 2019. doi: 10.1109/ACCESS.2018.2886356. URL https://doi.org/10.1109/ACCESS.2018.2886356.

[20] Dimitris Geneiatakis, Tasos Dagiuklas, Costas Lambrinoudakis, Georgios Kambourakis, and Stefanos Gritzalis. Novel protecting mechanism for sip-based infrastructure against malformed message attacks: Performance evaluation study. In *Proceedings of the 5th International Conference on Communication Systems, Networks and Digital Signal Processing (CSNDSP)*, pages 261–266. Citeseer, 2006.

[21] Georgios Kambourakis, Dimitris Geneiatakis, Stefanos Gritzalis, Costas Lambrinoudakis, Tasos Dagiuklas, Sven Ehlert, and Jens Fiedler. High availability for sip: Solutions and real-time measurement performance evaluation. *International Journal of Disaster Recovery and Business Continuity*, 1(1), 2010.

[22] Jonathan Rosenberg, Henning Schulzrinne, Gonzalo Camarillo, Alan Johnston, Jon Peterson, Robert Sparks, Mark Handley, and Eve Schooler. Rfc3261: Sip: session initiation protocol, 2002.

[23] Henning Schulzrinne, Steven Casner, R Frederick, and Van Jacobson. Rfc3550: Rtp: A transport protocol for real-time applications, 2003.

[24] Mark Baugher, David McGrew, Mats Naslund, Elisabetta Carrara, and Karl Norrman. The secure real-time transport protocol (srtp). Technical report, 2004.

[25] Mark Baugher, David McGrew, Mats Naslund, Elisabetta Carrara, and Karl Norrman. Rfc3711: The secure real-time transport protocol (srtp), 2004.

[26] Mark Handley, Van Jacobson, and Colin Perkins. SDP: session description protocol. *RFC*, 4566:1–49, 2006. doi: 10.17487/RFC4566. URL https://doi.org/10.17487/RFC4566.

[27] Alissa Cooper, Hannes Tschofenig, Bernard Aboba, Jon Peterson, John B. Morris, Marit Hansen, and Rhys Smith. Privacy considerations for internet protocols. *RFC*, 6973:1–36, 2013. doi: 10.17487/RFC6973. URL https://doi.org/10.17487/RFC6973.

[28] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC, 2004.

[29] Rob Jansen, Kevin S Bauer, Nicholas Hopper, and Roger Dingledine. Methodically modeling the tor network. In *CSET*, 2012.

[30] Tor Metrics-Relays. https://metrics.torproject.org/networksize.html, 2023. Accessed: 2023-05-02.

[31] Radhika Ranjan Roy. *Handbook of SDP for Multimedia Session Negotiations: SIP and WebRTC IP Telephony*. CRC Press, 2018.

[32] Web Real-Time Communication Market. https://www.grandviewresearch.com/industry-analysis/web-real-time-communication-market, 2023. Accessed: 2023-05-04.

[33] The Zfone project, 2003. URL http://zfoneproject.com/. online: http://zfoneproject.com/.

[34] Philip Zimmermann, Alan Johnston, and Jon Callas. ZRTP: media path key agreement for unicast secure RTP. *RFC*, 6189:1–115, 2011. doi: 10.17487/RFC6189. URL https://doi.org/10.17487/RFC6189.

[35] Silent Circle. https://silentcircle.com, 2003. Accessed: 2022-11-09.

[36] RedPhone, a secure calling app for android, 2003. URL https://github.com/WhisperSystems/RedPhone/wiki. online: https://github.com/WhisperSystems/RedPhone/wiki.

[37] Jitsi. https://jitsi.org, 2003. Accessed: 2022-11-09.

[38] Ostel. https://ostel.me, 2003. Accessed: 2022-11-09.

[39] Speak Freely. http://www.speakfreely.org, 2003. Accessed: 2022-11-09.

[40] IHU. http://ihu.sourceforge.net, 2003. Accessed: 2022-11-09.

[41] CryptoPhone. http://www.cryptophone.de, 2003. Accessed: 2022-11-09.

[42] Mumble. http://mumble.sourceforge.net, 2003. Accessed: 2022-11-09.

[43] Nautilus Secure Phone. https://github.com/argotel/nautilus, 2003. Accessed: 2023-05-22.

[44] iMule (2003). http://echelon.i2p.xyz/imule/, 2003. Accessed: 2022-11-09.

[45] StealthNet (2007). http://www.stealthnet.de/en_index.php, 2003. Accessed: 2022-11-09.

[46] Parekh (1996). Prospects for remailers. First Monday, 1(2), 1996. Accessed: 2022-11-09.

[47] Michael J Freedman and Robert Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 193–206, 2002.

[48] Marc Rennhard. Morphmix: Peer-to-peer based anonymous internet usage with collusion detection (available at http://www. tik. ee. ethz. ch/˜ rennhard/publications/morphmix. pdf). tik technical report nr. 147. *TIK, ETH Zurich, Zurich, CH*, 2002.

[49] M Brading. Generic, decentralized, unstoppable anonymity: the phantom protocol. *White paper*, 2011.

[50] Chen Chen, Daniele E Asoni, David Barrera, George Danezis, and Adrain Perrig. Hornet: High-speed onion routing at the network layer. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1441–1454, 2015.

[51] Albert Hyukjae Kwon, David Lazar, Srinivas Devadas, and Bryan Ford. Riffle: An efficient communication system with strong anonymity. 2015.

[52] Georgios Karopoulos, Alexandros Fakis, and Georgios Kambourakis. Complete sip message obfuscation: Privasip over tor. In *2014 Ninth International Conference on Availability, Reliability and Security*, pages 217–226. IEEE, 2014.

[53] Ge Zhang and Simone Fischer-Hübner. Peer-to-peer voip communications using anonymisation overlay networks. In *IFIP International Conference on Communications and Multimedia Security*, pages 130–141. Springer, 2010.

[54] Georgios Kambourakis. Anonymity and closely related terms in the cyberspace: An analysis by example. *Journal of information security and applications*, 19(1): 2–17, 2014.

[55] J Peterson. S/mime advanced encryption standard (aes) requirement for the session initiation protocol (sip). Technical report, 2004.

[56] Stevens Le Blond, David Choffnes, William Caldwell, Peter Druschel, and Nicholas Merritt. Herd: A scalable, traffic analysis resistant anonymity network for voip systems. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 639–652, 2015.

[57] Nagendra Modadugu and Eric Rescorla. The design and implementation of datagram tls. In *NDSS*, 2004.

[58] Stephan Heuser, Bradley Reaves, Praveen Kumar Pendyala, Henry Carter, Alexandra Dmitrienko, William Enck, Negar Kiyavash, Ahmad-Reza Sadeghi, and Patrick Traynor. Phonion: Practical protection of metadata in telephony networks. *Proc. Priv. Enhancing Technol.*, 2017(1):170–187, 2017.

[59] TORFone. URL `http://torfone.org/index.html`. online: http://torfone.org/index.html.

[60] Nasser Mohammed Al-Fannah. One leak will sink a ship: Webrtc ip address leaks. In *2017 International Carnahan Conference on Security Technology (ICCST)*, pages 1–5. IEEE, 2017.

[61] Chaoge Liu, Xiang Cui, Zhi Wang, Xiaoxi Wang, Yun Feng, and Xiaoyun Li. Malicescript: a novel browser-based intranet threat. In *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*, pages 219–226. IEEE, 2018.

[62] Mohammadreza Hazhirpasand and Mohammad Ghafari. One leak is enough to expose them all. In *International Symposium on Engineering Secure Software and Systems*, pages 61–76. Springer, 2018.

[63] Andreas Reiter and Alexander Marsalek. Webrtc: your privacy is at risk. In *Proceedings of the Symposium on Applied Computing*, pages 664–669. ACM, 2017.

[64] JSLanScanner. `https://code.google.com/archive/p/jslanscanner/`, 2016. Accessed: 2022-11-09.

[65] Rio Hosoi, Takamichi Saito, Takayuki Ishikawa, Daichi Miyata, and Yongyan Chen. A browser scanner: Collecting intranet information. In *2016 19th International Conference on Network-Based Information Systems (NBiS)*, pages 140–145. IEEE, 2016.

[66] Y Fablet, JD Borst, J Uberti, and Q Wang. Using multicast dns to protect privacy when exposing ice candidates. Technical report, Internet-Draft draft-ietf-rtcweb-mdns-ice-candidates-04, 2021.

[67] Stuart Cheshire and Marc Krochmal. Rfc 6762: Multicast dns, 2013.

[68] EFForg/Privacy Badger. `https://github.com/EFForg/privacybadger`, 2016. Accessed: 2022-11-09.

[69] uBlock Origin. `https://github.com/gorhill/uBlock`, 2016. Accessed: 2022-11-09.

[70] Amit Klein and Benny Pinkas. From {IP}{ID} to device {ID} and {KASLR} bypass. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1063–1080, 2019.

[71] Nasser Mohammed Al-Fannah and Wanpeng Li. Not all browsers are created equal: Comparing web browser fingerprintability. In *International workshop on security*, pages 105–120. Springer, 2017.

[72] Steven Englehardt and Arvind Narayanan. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 1388–1401, 2016.

[73] Xiaofeng Liu, Qixu Liu, Xiaoxi Wang, and Zhaopeng Jia. Fingerprinting web browser for tracing anonymous web attackers. In *2016 IEEE First International Conference on Data Science in Cyberspace (DSC)*, pages 222–229. IEEE, 2016.

[74] Furkan Alaca and Paul C Van Oorschot. Device fingerprinting for augmenting web authentication: classification and analysis of methods. In *Proceedings of the 32nd annual conference on computer security applications*, pages 289–301, 2016.

[75] WebRTC. URL http://www.webrtc.org/. online: http://www.webrtc.org/.

[76] Evangelos Mitakidis, Dimitrios Taketzis, Alexandros Fakis, and Georgios Kambourakis. Snoopybot: An android spyware to bridge the mixes in tor. In *24th International Conference on Software, Telecommunications and Computer Networks, SoftCOM 2016, Split, Croatia, September 22-24, 2016*, pages 1–5. IEEE, 2016. doi: 10.1109/SOFTCOM.2016.7772180. URL https://doi.org/10.1109/SOFTCOM.2016.7772180.

[77] A. Christensen. Practical onion hacking: Find the real address of tor clients, Oct 2016. URL http://www.fortconsult.net/images/pdf/Practical_Onion_Hacking.pdf.

[78] Xiaogang Wang, Junzhou Luo, Ming Yang, and Zhen Ling. A potential http-based application-level attack against tor. *Future Generation Computer Systems*, 27(1): 67–77, 2011.

[79] Zhen Ling, Junzhou Luo, Wei Yu, Xinwen Fu, Dong Xuan, and Weijia Jia. A new cell counter based attack against tor. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 578–589, 2009.

[80] Xinwen Fu, Zhen Ling, J Luo, W Yu, W Jia, and W Zhao. One cell is enough to break tor's anonymity. In *Proceedings of Black Hat Technical Security Conference*, pages 578–589, 2009.

[81] Timothy G Abbott, Katherine J Lai, Michael R Lieberman, and Eric C Price. Browser-based attacks on tor. In *International Workshop on Privacy Enhancing Technologies*, pages 184–199. Springer, 2007.

[82] Polipo - a caching web proxy, 2024. URL https://www.irif.fr/~jch/software/polipo/. Accessed: 2022-11-09.

[83] Privoxy-Developers, 2003. online: https://www.privoxy.org/.

[84] TorStatus - tor network status. URL https://torstatus.rueckgr.at/. Accessed: 2022-11-09.

[85] Georgios Kambourakis. Anonymity and closely related terms in the cyberspace: An analysis by example. *Journal of Information Security and Applications*, 2014. ISSN 2214-2126. doi: http://dx.doi.org/10.1016/j.jisa.2014.04.001. URL http://www.sciencedirect.com/science/article/pii/S2214212614000209.

[86] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), June 2002. URL http://www.ietf.org/rfc/rfc3261.txt. Updated by RFCs 3265, 3853, 4320, 4916, 5393, 5621, 5626, 5630, 5922, 5954, 6026, 6141, 6665, 6878.

[87] J. Peterson. A Privacy Mechanism for the Session Initiation Protocol (SIP). RFC 3323 (Proposed Standard), November 2002. URL http://www.ietf.org/rfc/rfc3323.txt.

[88] Charles Shen and Henning Schulzrinne. A VoIP privacy mechanism and its application in VoIP peering for voice service provider topology and identity hiding. arXiv e-print 0807.1169, July 2008. URL http://arxiv.org/abs/0807.1169.

[89] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-generation Onion Router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 21–21, Berkeley, CA, USA, 2004. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1251375.1251396.

[90] Nicholas Hopper, Eugene Y. Vasserman, and Eric Chan-tin. How much anonymity does network latency leak? In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, pages 82–91. ACM, 2007.

[91] Oliver Berthold, Hannes Federrath, and Marit Köhntopp. Project "anonymity and unobservability in the internet". In *Proceedings of the Tenth Conference on Computers, Freedom and Privacy: Challenging the Assumptions*, CFP '00, pages 57–65, New York, NY, USA, 2000. ACM. doi: 10.1145/332186.332211.

[92] Angelos D Keromytis. A comprehensive survey of voice over ip security research. *IEEE Communications Surveys & Tutorials*, 14(2):514–537, 2011.

[93] George Danezis. The traffic analysis of continuous-time mixes. In *Privacy Enhancing Technologies*, pages 35–50, 2004.

[94] Vitaly Shmatikov and Ming-Hsiu Wang. Timing analysis in low-latency mix networks: Attacks and defenses. In *ESORICS*, pages 18–33, 2006.

[95] Rungrat Wiangsripanawan, Willy Susilo, and Reihaneh Safavi-Naini. Design principles for low latency anonymous network systems secure against timing attacks. In *ACSW Frontiers*, pages 183–191, 2007.

[96] Joan Feigenbaum, Aaron Johnson, and Paul F. Syverson. Preventing active timing attacks in low-latency anonymous communication. In *Privacy Enhancing Technologies*, pages 166–183, 2010.

[97] Marc Rennhard and Bernhard Plattner. Practical anonymity for the masses with morphmix. In *Financial Cryptography*, pages 233–250, 2004.

[98] Michael J. Freedman and Robert Morris. Tarzan: a peer-to-peer anonymizing network layer. In *ACM Conference on Computer and Communications Security*, pages 193–206, 2002.

[99] F. Pereñiguez-Garcia, R. Marin-Lopez, G. Kambourakis, A. Ruiz-Martinez, S. Gritzalis, and A.F. Skarmeta-Gomez. Kamu: providing advanced user privacy in kerberos multi-domain scenarios. *International Journal of Information Security*, pages 1–21, 2013. doi: 10.1007/s10207-013-0201-1.

[100] J Rosenberg. Rfc 5245: Interactive connectivity establishment (ice): A protocol for network address translator (nat) traversal for offer. *Answer Protocols, Feb*, 2010.

[101] Gouda I Salama, M Elemam Shehab, AA Hafez, and M Zaki. Performance analysis of transmitting voice over communication links implementing ipsec. In *Paper in 13th International Conference on Aerospace Sciences and Aviation Technology (ASAT), Military Technical College, Cairo, Egypt*, 2009.

[102] Bassam Zantout and Ramzi Haraty. I2p data communication system. In *Proceedings of ICN*, pages 401–409, 2011.

[103] Marcus Leech, Matt Ganis, Y Lee, Ron Kuris, David Koblas, and L Jones. Rfc1928: Socks protocol version 5, 1996.

[104] Michael J Freedman and Robert Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 193–206. ACM, 2002.

[105] Paul Syverson, Gene Tsudik, Michael Reed, and Carl Landwehr. Towards an analysis of onion routing security. In *Designing Privacy Enhancing Technologies*, pages 96–114. Springer, 2001.

[106] Paul Baran. On distributed communications networks. *IEEE transactions on Communications Systems*, 12(1):1–9, 1964.

[107] Paul F Syverson, David M Goldschlag, and Michael G Reed. Anonymous connections and onion routing. In *Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No. 97CB36097)*, pages 44–54. IEEE, 1997.

[108] Xinwen Fu, Bryan Graham, Riccardo Bettati, and Wei Zhao. Active traffic analysis attacks and countermeasures. In *2003 International Conference on Computer Networks and Mobile Computing, 2003. ICCNMC 2003.*, pages 31–39. IEEE, 2003.

[109] Guardian-project. orbot: Tor for android. URL https://guardianproject.info/apps/orbot/.

[110] Rolf Wendolsky, Dominik Herrmann, and Hannes Federrath. Performance comparison of low-latency anonymisation services from a user perspective. In *International Workshop on Privacy Enhancing Technologies*, pages 233–253. Springer, 2007.

[111] Kamailio – the open source sip server. URL https://www.kamailio.org. online: https://www.kamailio.org.

[112] Twinkle, open source SIP softphone. URL http://www.twinklephone.com. Accessed: 2023-05-21.

[113] csipsimple. URL https://code.google.com/archive/p/csipsimple/. Accessed: 2023-05-23.

[114] sipdroid. URL http://www.twinklephone.com. Accessed: 2023-05-21.

[115] Sam Bowne. How socks5 works. https://samsclass.info/122/proj/how-socks5-works.html, 2009. Accessed: 2016-09-30.

[116] I2P. The invisible internet project. https://geti2p.net/en/, 2013. Accessed: 2016-09-30.

[117] ProxyChains. Tcp and dns through proxy server. http://proxychains.sourceforge.net/, 2002. Accessed: 2016-09-30.

[118] Zisis Tsiatsikas, Marios Anagnostopoulos, Georgios Kambourakis, Sozon Lambrou, and Dimitris Geneiatakis. Hidden in plain sight. sdp-based covert channel for botnet communication. In *International Conference on Trust and Privacy in Digital Business*, pages 48–59. Springer, 2015.

[119] The WebRTC Wave Is Now Unstoppable. https://www.nojitter.com/webrtc-wave-now-unstoppable, 2018. Accessed: 2019-01-03.

[120] Rohan Mahy, Philippe Matthews, and Jonathan Rosenberg. Rfc 5766: Traversal using relays around nat (turn): relay extensions to session traversal utilities for nat (stun). *Internet Engineering Task Force*, 2010.

[121] R Mahy RFC5766, J Rosenberg, and C Huitema. Turn: traversal using relay nat. Technical report, Internet draft, Internet Engineering Task Force, 2004.

[122] J Uberti, C Jennings, and E Rescorla. Javascript session establishment protocol, draft-ietf-rtcweb-jsep-25, 2018.

[123] Alan B Johnston and Daniel C Burnett. *WebRTC: APIs and RTCWEB protocols of the HTML5 real-time web*. Digital Codex LLC, 2012.

[124] E Rescorla. Rfc 8826 security considerations for webrtc. 2021.

[125] AdBlock. Adblock plus. https://github.com/adblockplus/adblockplus, 2017.

[126] Maxim Krasnyansky and M Yevmenkin. Virtual point-to-point (tun) and ethernet (tap) devices, 2006.

[127] Tunnelbear llc. tunnelbear. URL https://www.tunnelbear.com/. Accessed: 2023-05-23.

[128] Anchorfree, hotspot shield. URL https://www.hotspotshield.com/. Accessed: 2023-05-23.

[129] Christer Jakobsson. Peer-to-peer communication in web browsers using webrtc a detailed overview of webrtc and what security and network concerns exists, 2015.

[130] Hazhirpasand Mohammadreza and Ghafari Mohammad. One leak is enought to expose them all. In *Engineering Secure Software and Systems: 10th International Symposium*, pages 664–669. ACM, 2018.

[131] Patrick Schleizer. Fix shared vpn/tor server leak bug. https://github.com/adrelanos/vpn-firewall/issues/12, 2016.

[132] Media Capture and Streams. https://dev.w3.org/2011/webrtc/editor/archives/20140619/getusermedia.html, 2014. Accessed: 2019-06-19.

[133] European union public licence. URL https://ec.europa.eu/info/european-union-publiclicence_en. Accessed: 2023-05-23.

[134] The average web page is 3MB. How much should we care? https://speedcurve.com/blog/web-performance-page-bloat, 2016. Accessed: 2019-10-09.

[135] 10 Ad Blocking Extensions Tested for Best Performance. https://www.raymond.cc/blog/10-ad-blocking-extensions-tested-for-best-performance/3/, 2017. Accessed: 2020-03-20.

[136] Chrome developer tools, 2023. URL https://developer.chrome.com/docs/devtools/. Accessed: 2023-05-23.

[137] A Primer for Web Performance Timing APIs.
urlhttps://w3c.github.io/perf-timing-primer/, 2019.

[138] Oleg Moskalenko. coturn turn server project. https://https://github.com/coturn/coturn, 2016.

[139] Alexandros Fakis, Georgios Karopoulos, and Georgios Kambourakis. Neither denied nor exposed: Fixing webrtc privacy leaks. *Future Internet*, 12(5):92, 2020.

[140] Kyle Hogan, Sacha Servan-Schreiber, Zachary Newman, Ben Weintraub, Cristina Nita-Rotaru, and Srinivas Devadas. Shortor: Improving tor network latency via multi-hop overlay routing. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1933–1952. IEEE, 2022.

[141] Shadowsocks. URL https://shadowsocks.org. Accessed: 2023-05-23.