Onion Privacy Guard

MOHAMMAD MAHAD SIDDIQUI
01-135221-027
FAIZAN JAWAD AHMED
01-135221-097

Supervisor: Dr. Arif Ur Rahman

Bachelor of Science in Information Technology

Department of Computer Science
Bahria University, Islamabad

December 2025

Certificate

We accept the work contained in the report titled "ONION PRIVACY GUARD", written by Mr. Mohammad Mahad Siddiqui AND Mr. Faizan Jawad Ahmed as a confirmation to the required standard for the partial fulfillment of the degree of Bachelor of Science in Information Technology.

Approved by . . . :

Supervisor: Dr. Arif Ur Rahman (Professor)

_____

Internal Examiner: Name of the Internal Examiner (Title)

_____

External Examiner: Name of the External Examiner (Title)

_____

Project Coordinator: Dr. Kabeer Ahmed Bhatti (Fyp Coordinator IT Department)

_____

Head of the Department: Dr. Muhammad Khurram Ehsan (Head Of Department Computer Science)

_____

December 31st, 2025

Abstract

Onion Routing (OR) is a method that makes the communication private and secure via the internet. This project will focus on creating a system where people can send anonymity of data via a set of special computers known as onion routers. By enveloping messages in more than one layers of encryption, OR does not allow any one server to know full path to the destination server.

Some of the major characteristics that will be incorporated in the system include the directory authority to locate available routers,hybrid encryption, secure connections via key exchange, and multi-hop encryption to secure messages at every step. OR improves the privacy of users compared to VPNs that can reveal their identities at the end server, hiding traffic on each stage.

As the use of mass monitoring, traffic analysis, and cyber-dangers grow, it is necessary to ensure the issue of online anonymity has become a critical issue to individuals, activists, journalists, and cybersecurity professionals. The Onion Privacy Guard (OPG) project tackles the challenges through an added layer of security infrastructure that incorporates onion routing approach in order to secure traffic routing, DNS leak prevention with DNS-over-TLS and DNS-over-HTTPS, firewall-based enforcement of anonymity, application sandboxing, and firejail user profile isolation.

The existing project will involve the development of a functional OR system, the encryption techniques analysis, and so on evaluating its success in avoiding surveillance and tracking. The proposed system will cater to as an effective basis of safe and unidentified internet communications.

Acknowledgments

We would like to take this opportunity to thank our supervisor, Dr. Arif Ur Rahman, by his priceless advice, help, and experience during the creation of this project. His feedbacks and suggestions have played a significant role in directing and determining the quality of our work.

We would like to thank the Department of Computer Science to Bahria University, Islamabad, to give us the means and the surroundings under which to do our research. We also owe its debt to the open-source community, especially Tor, ProxyChains, and developers and other privacy-enhancing technologies, the work of which has given us inspiration and information on our project.

We would like to thank our fellow instructors, students and friends who were of moral assistance and provided constructive evaluation at the difficult stages of the project. Lastly we would like to appreciate everyone who contributed directly or indirectly to our project's success through their advice, assistance, or encouragement. Your contributions have been vital in completing this milestone in our academic journey

MOHAMMAD MAHAD SIDDIQUI AND FAIZAN JAWAD AHMED
Islamabad, Pakistan

November 2025

Quote

"Privacy is not something that I'm merely entitled to, it's an absolute prerequisite for freedom."

— Edward Snowden

Contents

Abstract i

1 Introduction 1
1.1 Project Background . . . 2
1.1.1 Problem Description . . . 2
1.1.2 Project Objectives . . . 3
1.1.3 Project Scope . . . 3

2 Literature Review 5
2.1 Onion Routing and Its Applications . . . 6
2.1.1 Python-Based Security Enhancements for Onion Routing . . . 6
2.1.2 Secure DNS and Firewall Protection . . . 7
2.1.3 Traffic Obfuscation and Detection Arms Race . . . 7
2.1.4 Traffic Analysis and Routing Attacks . . . 8
2.1.5 Path Selection and Guard Placement . . . 8
2.1.6 Directory Authorities and Protocol Security . . . 8
2.2 Comparative Study . . . 9

3 Requirement Specifications 14
3.1 Existing System . . . 15
3.1.1 Onion Routing and the Tor Network . . . 15
3.1.2 Virtual Private Networks (VPNs) . . . 16
3.1.3 Application Sandboxing with Firejail . . . 16
3.1.4 Encrypted DNS and DNS Leak Prevention . . . 17
3.2 Requirements Specifications . . . 18
3.3 Functional Requirements . . . 18
3.3.1 FR-01: Core Privacy Features . . . 18
3.3.2 FR-02: System Management . . . 19
3.3.3 FR-03: Security Features . . . 19
3.4 Non-Functional Requirements . . . 20
3.4.1 Performance . . . 20
3.4.2 Reliability . . . 20
3.4.3 Security . . . 20
3.4.4 Usability . . . 20
3.5 Proposed System . . . 21
3.5.1 The Proposed System Overcomes Existing System Limitations . . . 21
3.6 Use Cases Diagram . . . 22
3.7 Comparison of Privacy Techniques . . . 36

4 Design 37
4.1 System Architecture . . . 37
4.2 Design Constraints . . . 40

4.2.1 Target Platform . . . 40
4.2.2 Security and Privacy Standards . . . 40
4.2.3 Performance . . . 40
4.2.4 Usability . . . 40
4.2.5 Integration with Third-Party Tools . . . 40
4.3 Design Methodology . . . 41
4.3.1 Waterfall Model . . . 41
4.3.2 Activity Diagram . . . 42
4.4 High Level Design . . . 44
4.4.1 Component Diagram . . . 44
4.4.2 Deployment Diagram . . . 46
4.4.3 Sequence Diagram . . . 47
4.5 Low Level Design . . . 51
4.6 GUI Design . . . 53

5 System Implementation 60
5.1 System Architecture . . . 60
5.1.1 System Internal Components . . . 60
5.1.2 Functionality of Components . . . 61
5.1.3 Communication Between Components . . . 61
5.2 Tools and Technology Used . . . 62
5.3 Development Environment / Layout . . . 62
5.4 Processing Logic / Algorithms . . . 62
5.4.1 Path Selection (DA) . . . 62
5.4.2 Onion Packet Construction (Client) . . . 62
5.4.3 Node Processing . . . 63
5.4.4 SOCKS5 Proxy and Tunnels . . . 63
5.4.5 Circuit Rotation and Continuity . . . 63
5.4.6 DNS Leak Prevention . . . 63
5.5 Application Access Security . . . 63
5.5.1 Security Zones and Firewalls . . . 63
5.5.2 Encryption . . . 64
5.5.3 Authentication . . . 64
5.5.4 Authorization and Roles . . . 64
5.5.5 Auditing / Access Logging . . . 64
5.5.6 Profiling Resistance (Browser) . . . 64
5.6 Safe Data Storage ("Database" Security) . . . 65
5.7 Deployment . . . 65

6 System Testing and Evaluation 66
6.1 Testing Methodology . . . 66
6.2 Test Environment . . . 67
6.3 Functional Testing . . . 67
6.4 Non-Functional Testing . . . 73
6.4.1 Performance Testing . . . 73
6.4.2 Usability Testing . . . 74

6.4.3 Compatibility Testing . . . 75
6.4.4 Security Testing . . . 77
6.5 Evaluation Metrics and Analysis . . . 78
6.5.1 Anonymity Metrics . . . 78
6.5.2 Security Metrics . . . 78
6.5.3 Performance Metrics . . . 78
6.6 Strengths and Weaknesses . . . 79
6.6.1 Strengths . . . 79
6.6.2 Weaknesses and Limitations . . . 79
6.7 Comparison with Existing Systems . . . 80
6.8 Summary of Evaluation Results . . . 80

7 Conclusions **82**
7.1 Key Conclusions and Lessons Learned . . . 82
7.2 Current Limitations . . . 83
7.3 Future Work and Extensions . . . 83
7.3.1 Network and Cryptography . . . 83
7.3.2 Security and Hardening . . . 84
7.3.3 Operations and Tooling . . . 84
7.3.4 User Experience . . . 84
7.4 Closing Statement . . . 84

A User Manual **85**
A.1 System Requirements . . . 85
A.1.1 Hardware . . . 85
A.1.2 Software . . . 85
A.2 Installation . . . 85
A.2.1 Using the provided installer (openSUSE Tumbleweed) . . . 85
A.2.2 Manual install (any Linux) . . . 86
A.2.3 SSH setup for key sharing (required) . . . 86
A.3 Project Layout . . . 87
A.4 Running OPG . . . 87
A.4.1 Start the GUI . . . 87
A.4.2 SOCKS5 Proxy . . . 87
A.4.3 Client CLI (non-GUI) . . . 87
A.4.4 DNS Leak Fix Helper . . . 88
A.4.5 Directory Authority and Nodes (developer/demo) . . . 88
A.4.6 Deploy node files to remote hosts . . . 88
A.4.7 Node installation scripts . . . 88
A.4.8 Connectivity and port forwarding . . . 89
A.5 Basic Usage . . . 89
A.5.1 Browse via SOCKS5 . . . 89
A.5.2 Firewall Controls . . . 89
A.5.3 DNS Leak Tests . . . 89
A.6 Troubleshooting . . . 89
A.7 Command Reference . . . 90

References 91

turnitin Page 11 of 106 - AI Writing Submission Submission ID trn:oid:::1:3445230174

List of Figures

3.1 Comparison of Privacy Techniques 18
3.2 OPG Use Case Diagram 22

4.1 OPG System Architecture 38
4.2 High Level Context Diagram for Onion Privacy Guard (OPG) 39
4.3 Waterfall Model Methodology for OPG Development Lifecycle 42
4.4 Activity Diagram: Primary Tab Controls 43
4.5 OPG High-Level Component Diagram 45
4.6 OPG Deployment Diagram 46
4.7 OPG Tor Service Activation Sequence Diagram 47
4.8 OPG Service Activation and Rotation Sequence Diagram 48
4.9 OPG DNS Leak Test Sequence Diagram 49
4.10 OPG User Browsing Flow (SOCKS5 request) Sequence Diagram 50
4.11 Node Registration Via Directory Authority 51
4.12 OPG UML Frontend Diagram 52
4.13 OPG UML backend Diagram 53
4.14 OPG Onion Routing Proxy Control Panel 54
4.15 Node Configuration Interface 55
4.16 Security and Encryption Configuration 56
4.17 Network and Performance Configuration 57
4.18 Firewall Management Interface 58
4.19 OPG Security Tools Interface 59

List of Tables

2.1 Comparative Study 10

3.1 Build or Rotate Circuit (Client) 23
3.2 Start SOCKS5 Proxy (Client) 24
3.3 Apply DNS Leak Prevention (Route DNS via SOCKS5) 25
3.4 Enable Firewall Egress Control 26
3.5 Maintain Active SOCKS5 Streams During Circuit Rotation 27
3.6 Test DNS Leak Protection 28
3.7 Multi-Client Concurrent Access to Onion Network 29
3.8 Configure Extended Circuit Hop Count (up to 8 Hops) 30
3.9 Launch Sandboxed Browser via OPG Client 31
3.10 Check Public IP Address via OPG Circuit 32
3.11 Configure and Validate Node Settings 33
3.12 Configure TLS Obfuscation for Client-Entry Link 34
3.13 Save, Load, and Reset Configuration 35
3.14 Comparison of Existing Privacy Techniques with OPG 36

6.1 Test Case 1: Directory Authority Registration and Consensus 68
6.2 Test Case 2: Circuit Construction and Path Selection 69
6.3 Test Case 3: Onion Packet Encryption and Decryption 70
6.4 Test Case 4: SOCKS5 Proxy Functionality 71
6.5 Test Case 5: Circuit Rotation and Stream Persistence 71
6.6 Test Case 6: DNS Leak Prevention 72
6.7 Test Case 7: Firewall Integration 72
6.8 Test Case 8: Firejail Browser Sandboxing 73
6.9 Test Case 9: Circuit Construction Latency 74
6.10 Test Case 10: Packet Forwarding Throughput 74
6.11 Test Case 11: GUI Responsiveness and Ease of Use 75
6.12 Test Case 12: Multi-Client Concurrent Access 76
6.13 Test Case 13: Multi-Distribution Installation 77
6.14 Test Case 14: Encryption Integrity Under MITM Attack 77
6.15 Performance Summary 78
6.16 OPG vs. Tor Network Comparison 80

A.1 OPG Local Commands 90

Acronyms and Abbreviations

AES	Advanced Encryption Standard
API	Application Programming Interface
APT	Advanced Persistent Threat
AS	Autonomous System
CBC	Cipher Block Chaining
DA	Directory Authority
DAC	Discretionary Access Control
DH	Diffie-Hellman
DNS	Domain Name System
DoH	DNS-over-HTTPS
DoT	DNS-over-TLS
GCM	Galois/Counter Mode
GUI	Graphical User Interface
HMAC	Hash-based Message Authentication Code
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IP	Internet Protocol
ISP	Internet Service Provider
JSON	JavaScript Object Notation
MAC	Mandatory Access Control
MitM	Man-in-the-Middle
OAEP	Optimal Asymmetric Encryption Padding
OPG	Onion Privacy Guard
OR	Onion Routing
PFS	Perfect Forward Secrecy
PGP	Pretty Good Privacy
RSA	Rivest–Shamir–Adleman (public-key cryptosystem)
SOCKS5	Socket Secure version 5
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
UFW	Uncomplicated Firewall
UI	User Interface
UX	User Experience
VM	Virtual Machine
VPN	Virtual Private Network

Chapter 1

Introduction

Recent developments in the OPG project have shifted the focus from integrating with the public Tor network to implementing a private, Tor-inspired onion routing system with significant advantages over traditional implementations. In this design, a set of lightweight virtual machines (VMs) operate as onion routers with explicit roles (entry, middle1, middle2, exit), supporting configurable 4-8 hop circuits compared to Tor's fixed 3-hop limitation. A central Directory Authority (DA) maintains a registry of active nodes and returns diversity-aware paths to clients with geographic and subnet diversity enforcement. The client constructs multi-layer "onion" packets using configurable encryption algorithms (AES-128/256-CBC, AES-128/256-GCM) per hop with HMAC-SHA256 for integrity, and wraps per-hop AES keys with RSA-2048 (OAEP) using each node's public key. Each node peels a single layer and forwards the remaining ciphertext to the next hop until the exit delivers the final payload. The client provides a PyQt6-based graphical interface with real-time monitoring and exposes a local SOCKS5 proxy so applications can transparently route traffic through the circuit. The system has Firejail sandboxing to provide a better isolation of applications and is optimized in Linux systems. In this chapter, the author presents the background, purpose, and goals of the Onion Privacy Guard (OPG) project. In a time when mass surveillance, cyber attacks and data breaches are growing more threatening to digital privacy, the necessity of effective anonymity solutions has never been as high as it is today. The OPG project will be able to overcome these challenges by incorporating advanced privacy-saving technologies and user-friendly interfaces. This chapter provides the history of the project, the issues which it is intended to address and the objectives which should be followed in the development of the project. It also preconditions the following chapters, which will involve the technical, methodological, and evaluative features of the system.

In the recent developments of the OPG project, the emphasis has been redirected to making it integration with the public Tor network to an implementation of a private, Tor-like onion routing system. This design uses a system of lightweight virtual machines (VMs) acting as onion routers that explicitly have entry, middle, exit roles. One central Dir authority (DA) keeps a table of active nodes and provides diversity-conscious paths to clients. The client packets are assembled in layers of packets multiplexed with the AES-256-CBC protocol and an integrity probe packet (HMAC-SHA258) assigned to each hop, and the AES keys (per hop) are encrypted

by the RSA-2048 (OAEP) protocol using the public keys of each node. Every node removes one layer of the ciphertext and transmits the rest of the ciphertext to the next hop until the exit is providing the final payload. The client also gives a local SOCKS5 proxy whereby applications may transparently send traffic over the circuit.

Guaranteeing anonymity on the internet is not only a technical issue but also a social necessity. People, reporters and even organizations are increasingly exposed to dangers as their activities on the internet are being tracked and scrutinized by different agencies. Historically used tools of privacy are practical, but usually put all the trust in one place, or expose metadata.

1.1 Project Background

As mass surveillance, traffic analysis, and cyber threats continue to grow in popularity, online anonymity has become a key issue to many individuals, activists, journalists, and cybersecurity experts. The existing privacy protection schemes, including Virtual Private Networks (VPNs) will offer certain anonymity, yet they will not fully defend users against traffic correlation attacks, DNS leaks, and compromised exit nodes.

Originally created to support anonymous communication, Onion Routing (OR) offers a multi-layered encryption strategy, which contributes to the anonymity of users and alleviation of the risk of being monitored. OR works through multiple levels of message encryption and transmitting it through a chain of nodes (entry, middle, exit). The nodes strip off a single level of encryption and only learns the next hop, such that no one node is aware of the sender and the ultimate recipient. The resulting layered encryption model has been shown to dramatically improve the privacy and anonymity and forms the basis of systems like Tor. [1].

1.1.1 Problem Description

In spite of benefits, OR-based systems have practical issues. Public networks are vulnerable to dangerous exits, traffic, metadata, and changing circuits may sabotage unlinkability, long-lived sessions can be disrupted by frequent circuit changes. In informal implementations such as OPG, simplifying assumptions, such as single- Directory Authority, variable-size JSON packets and RSA-based key transport without perfect forward secrecy, add new risks that have to be acknowledged and addressed.

Existing privacy solutions have the following common problems:

- **Exit Observation and Abuse:** Exit relays can inspect or manipulate plaintext application data; lack of exit policies can enable misuse.
- **Instability in Circuit:** Long downloads or long running sessions interrupt the route causing degradation of usability.
- **Bypass Risks:** The applications can be incorrectly configured and bypass the anonymity path.
- **Complexity in Operations:** Expertise and good tooling are expected when it comes to safe default and proper routing.

1.1.2 Project Objectives

Onion Privacy Guard (OPG) is a project, which seeks to solve these issues by a Tor-inspired private prototype, which centers on transparency and control. The objectives are:

- **Implement a Private Onion Network:** Deploy Entry, Middle, and Exit nodes on VMs; expose a single Directory Authority for node registration and path selection.
- **Circuit Construction and Rotation:** Build 3-4 hop circuits with elementary subnet/IP diversity and periodic rotation so that there is less correlation between circuits.
- **Layered Cryptography:** AES-256-CBC HMAC-SHA256 integrity per hop; wrap (per-hop) AES keys with transporting RSA- 2048 (OAEP).
- **Client Integration through SOCKS5:** Have a local SOCKS5 proxy for application compatibility, which will allow the application to run on the circuit and be able to tunnel applications.
- **Mitigating Common Leaks:** Mitigate In-circuit DoH/DoT and a sandboxed browser workflow (Firefox started under Firejail with temporary Firefox profile) so cookies, browser cache, and history are deleted at the end of a session, to limit the profiling of users and to limit fingerprinting of application-layers.
- **Operational Controls:** Offer configurable parameters (hop count, rotation, transport obfuscation) and visibility into active nodes/consensus.

1.1.3 Project Scope

The OPG project will aim at the following key components:

- **Directory Authority and Consensus:** Active node registry maintenance, providing consensus, and making a selection based on simple diversity.
- **Adding node Roles and Forwarding:** Implementing entry/middle/exit behavior to peel of layers and packets forwarding with length-prefixed TCP.
- **Client and Proxy:** Build layered onions, circuits/key management and exposing a local SOCKS5 proxy to applications.
- **Security and Transport:** Enforcing per-hop HMAC, anti-replay windows, and optional TLS-based transport obfuscation.
- **Operational Tooling:** Offering script/service deployment, key distribution, monitoring in a controlled testbed, and a Firefox launch supported by Firejail to create a new temporary profile each session in order to avoid cookies that Last Forever and the like.

The integration of these elements will allow OPG to enhance user anonymity, minimize typical leak vectors, and offer a stable platform to users to experiment and communicate safely. Although it is not yet fully executing the perfect forward secrecy, fixed-size cells, and exit policies, its design is modular, and a good way to add such features is to iterate.

Chapter 2

Literature Review

This chapter is an in-depth literature review of privacy enhancing technology which is present in current literature, particularly Onion Routing (OR) and other devices of this nature. The aim of the literature review is to provide the Onion Privacy Guard (OPG) project with a good background in which the project is situated. It talks about the development of anonymity networks, merits and demerits of the solutions available, as well as deficiencies that users that are interested in maintaining their privacy online continue to face.

A close examination of the research conducted over the years will enable one to realize the critical role of the usage of technologies, including Tor and ProxyChains, in facilitating the industry of anonymous communication. However, the literature also reveals the perennial weaknesses that consist of DNS leaks, rogue exit nodes, and usability that make it rather difficult to adopt widely. These problems when examined are used in the determination of the gaps that the OPG project is expected to fill.

The review has been organized in ways that the historical development of the privacy tools has been covered, the different approaches and their comparison and how the regulatory and social factors have influenced the anonymity solutions. Through its discussion, the readers can know about the forces that drove the OPG project and the privacy studies in general.

The privacy-enhancing technologies are interdisciplinary and most other related disciplines (including computer science, information security, law, and sociology) have a vast literature on this topic. The first research conducted was directed at the calculation of cryptographic algorithms and channels of communication which resulted into the modern anonymity networks. As time passed, the focus was to apply these thoughts to the ground level thus coming up with a system like Tor which has become the new standard of conducting the anonymous browsing of the internet.

2.1 Onion Routing and Its Applications

The activists, journalists and individuals under oppressive regimes also find OR popular because they need to have an assurance that they would not be spied on. It also can be used to malware analysis and penetration testing where hackers must remain anonymous in order to research a threat.

Choudhary and Bhagat. [1] have described OR to be highly anonymous through encryption of traffic at different levels and transmitting through a parallel scheme of nodes. The approach is therefore appropriate in preventing the tracking of IP and location hence making it more difficult to monitor what the adversary is doing.

There are three significant nodes in the OR network:

- **Entry Node** - It is the initial node where the encrypted request of the user is redirected to
- **Relay Node/Middle Node** - These are the nodes that intermediarily hold the encrypted message and pass the encrypted message, but are not aware of the destination of this message.
- **Exit Node** - This node decrypts request and forwards this request to the target destination of the circuit.

Each node in the network possesses a key pair in the cryptographic process. Tor Ntor (Curve25519) is forward secret, and experimental implementations can make use of RSA-2048 to transport keys and per-hop AES to transport payloads. Metadata about relays is stored in directory services (public authorities in Tor; a single DA in prototypes such as OPG), and help clients construct anonymous circuits.

2.1.1 Python-Based Security Enhancements for Onion Routing

Rhodes and Goerzen, in a similar article [2], sheds light on network security, proxy support, DNS leak protection, and firewall security. Their Python socket protocol can be used to implement SOCKS5 interposition to make applications use anonymity systems.

Conventional or traditional network requests will have the actual IP address of the user circulated and it would be required to institute SOCKS5 proxy routing on outbound connections. Through the use of the socket and socks libraries, applications can be limited to the use of the client local SOCKS5 port which avoids direct connections which may jeopardize privacy.

They also talk of dynamic management of the SOCKS proxies and HTTP. Using multi-hop relay forwarding and a client-side SOCKS5 proxy it is possible in onion-routing prototypes to enable layered anonymity and still be compatible with existing applications.

2.1.2 Secure DNS and Firewall Protection

Zhu et al. [3] emphasize the weaknesses of the conventional DNS protocol and share T-DNS (Transport-layer DNS) as the solution. The UDP DNS that is used is unencrypted, and thus, the ISPs and governments, as well as attackers, can intercept DNS queries, launch attacks of spoofing and responses.

T-DNS provides encryption via TCP and TLS that eliminates eavesdropping and spoofing attacks. It also mitigates the DNS amplification attacks, which necessitate TCP handshakes verification, to provide a more airtight and privacy-vindicated DNS system.

In the case of the OPG project, DNS leak prevention is highly essential since it reveals the true IP address of a user. It is possible to accomplish this by configuring all DNS queries to resolve in the onion circuit using the SOCKS5 proxy running on the client and using DNS-over-HTTPS (DoH) or DNS-over-TLS (DoT) to encrypt queries and stop interception. The DefecTor line of work demonstrates how DNS side channels can degrade anonymity and thus motivate in-circuit DNS handling and padding [4]. Moreover, previous efforts endorse the separation of browser state; OPG uses Firefox-under-Firejail design, where the application starts with a short-term profile, meaning cookies, cache and history are killed at application termination, and long-term linkability is lowered.

2.1.3 Traffic Obfuscation and Detection Arms Race

Hostility is progressively using machine learning to obfuscate obfuscated Tor traffic [5], while Sanjalawe et al. use bidirectional GANs and vision transformers to enhance classification [6], whereas Xu and Zou [5] suggest sliding-window features to detect and learn obfuscated flows, which advance the classification as they is proposed in the article by Xu and Zou, in turn, tries to solve it presenting the bidirectional GANs as the solution to the classification problem, thus proving it at the same time as their counterpart does, only a bit differently i.e. by using bidirectional GANs, instead of vision transformers, as the solver in the classification. Adversarial methods to obfuscate against these detectors are researched by Sheffey [7] and they are more effective at obfuscating if the adversary is a stateful obfuscating attacker, otherwise the detection rate stays constant regardless of the method's robustness in one-way obfuscation (no longer obfuscated); and Lapshichyov studies specific characteristics of Tor TLS certificates that

could facilitate traffic detection or blocking [8]. These findings encourage pluggable transports, padding and protocol mimicry of privacy systems.

2.1.4 Traffic Analysis and Routing Attacks

Analysis of traffic is a key threat. Platzer et al. examine critical traffic analysis on the Tor network [9]. Buccafurri et al. examine sender anonymity on a global passive adversary and talk about the defenses [10]. Buccafurri et al. also address the issue of anonymity against a global passive enemy with implications on directory trust application [10]. Sun et al. introduce RAPTOR that disrupts anonymization of its users with the help of routing attacks based on BGP dynamics exploitation knowledge [11]. Subsequently, Sun et al. constructed Counter-RAPTOR based on the idea of countermeasures against active routing attacks by using countermeasures methods like relay fingerprinting and bandwidth-based detection [12]. These works inform OPG design choices for path diversity, circuit rotation, and link-layer defenses.

2.1.5 Path Selection and Guard Placement

The algorithm of path selection of critical relevance to the anonymity. Rochet et al. propose a more effective path selection method in Tor, CLAPS (Client-Location-Aware Path Selection), that lends a bit more attention to geographic and network proximity to shorten the latency and protect the security of this system at the same time [13]. Nevertheless, Wan et al. have shown that, in some cases path selection algorithms are sensitive to guard placement attack, the attackers can place malicious guards strategically to help them high chances of accessing circuits compromised by them [14]. Wails et al. use the Tempest to expose elliptical behavior of anonymity systems over time, and how these trends of user behavior and circuit lifetime can be used to generate information as time passes, highlighting the critical role of circuit rotation policy in the context of an anonymity system [15]. These principles and findings help OPG determine the way to build a circuit, frequency of rotation, constraints in the diversity of paths, between the factors of performance, security and resistance to long-term traffic analysis.

2.1.6 Directory Authorities and Protocol Security

Circuit construction is based on directory services. In addition to the classical design of Tor, last of all, hardening and attack surfaces are explored by previous work. Luo et al. focus on attacks and enhancement of the Tor directory protocol providing the information about the strong design of the authority design of a robust Tor authority design [16]. These researches or studies

enlighten the Directory Authority of OPG on the integrity of the consensus and relay metadata validation.

Van Vugt [17] discusses the need to have a properly set firewall in order to maintain online anonymity. Netfilter (iptables) offers packet filtering at the knee and UFW (Uncomplicated Firewall), is an easier-to-manage firewall. There are very few risks but with the rigorous application of the firewall rules, OPG can guarantee that only Tor traffic can be allowed but not the non-Tor traffic as well as avoid unauthorized outbound connection and register rejected packets as a method of tracking the possible leaks.

Napetvaridze and Gonzalez [18] etress the importance of application sandboxing with the help of some applications such as Firejail, which allows improving security measures by offering process isolation, resource privacy, and discretionary access control (DAC) to reduce risks of potential attacks. Firejail can be utilized by OPG to sandbox applications like browsers and file downloaders eliminating the execution of malicious code and unauthorized network access.

2.2 Comparative Study

The present section of the document presents the review and the comparative analysis of the available scholarly literature, privacy- noticeable tools, and commercial applications, which are applicable in our project. A critical analysis reveals the major developments in onion routing, resistance in traffic analysis and protection of privacy. The parallelism reveals the benefits and drawbacks of the existing strategies and highlights the gaps and opportunities that are going to be filled by the proposed Onion Privacy Guard (OPG) system. Such weaknesses are: the private onion routing testbeds, support of multi-client architecture and all round DNS leak protection, and embedded firewall administration.

Table 2.1: Comparative Study

Ref	Title	Key Insight	Project-Relevant Gap
[1]	Onion Routing: A Comprehensive Survey [1]	Research on anonymous network services has confirmed that multi-hop circuits implemented with entry/middle/exit nodes offer high levels of anonymity; layered encryption ensures traffic circulation is not correlated; circuit rotation prevents timing attacks	Research is restricted to the open Tor network, not to deployable private testbed or multiclient deployability, or to controlled operation of the DA, which are relevant to OPG.
[2]	Python Based Security Enhancements [2]	Python socket interposition techniques SOCKS5 interposition; dynamic HTTP/SOCKS proxy control; client-side SOCKS5 proxy multi-hop relay forwarding	Limited to general proxy techniques; no integration with onion routing threat, proxy auto-management, or multi-client interactions
[3]	T-DNS: Transport-layer DNS Solution [3]	TCP and TLS-based DNS encryption eliminates eavesdropping; mitigates DNS amplification attacks; needs to verify the TCP handshake	In-circuit DNS routing; not carried out practically and automated DNS leak prevention; or integrated with onion routing systems
[4]	DefecTor: DNS Impact on Anonymity [4]	DNS side channels penalize anonymity; DNS leak impact detection on privacy full in-circuit DNS processing motivation	DNS protection Limited to DNS leak analysis; no real-world example of automated DNS leak prevention, a private network DNS routing or DNS protection research.
[5]	Obfuscated Tor Traffic Identification [5]	Sliding-window features to detect obfuscated flows; Obfuscated Tor traffic fingerprinting and fingerprinting with machine learning: Better classification accuracy,	Limited to traffic detection; Nnot research-friendly to analyse traffic but Sliding-window features: no custom transport layer or features that are research-friendly, not scalable to custom transport layer flows and not scalable to obfuscated flows;

Table 2.1 – continued from previous page

Ref	Title	Key Insight	Project-Relevant Gap
[6]	Tor Traffic Detection with GANs [6]	Aiding classification with Bidirectional GANs and vision transformers; machine learning gain on fingerprinting traffic, high detection accuracy	Concentrates on detection algorithms;lacks countermeasures for private networks, custom obfuscation techniques, or research-oriented traffic protection
[7]	Adversarial Traffic Obfuscation [7]	Obfuscation is stronger with adversarial techniques; resistance to machine learning; better protective measures of traffic	Theoretical obfuscation methods and not implemented practically; do not exist in a deployed obfuscation system or as practical tool of transport protocol; custom transport protocol or obfuscation methods friendly to research
[8]	LS Certificates in Tor Network [8]	Described Unique features of TLS certificates in the Tor network support identification of traffic; detection by blocking on certificate validation; Techniques of TLS fingerprint methodology	Limited to public Tor analysis; ddoes not cover the organization of TLS-based obfuscation methods to private network TLS configuration, custom certificate use or research-oriented TLS obfuscation.
[9]	ritical Traffic Analysis on Tor [9]	High-level traffic analysis, ritical vulnerabilities in Tor network; analysis techniques and solutions	Focuses on public Tor network; omits resistance to analysis, path customization, and analysis friendly to research analysis mitigation
[10]	Sender Anonymity Against Global Adversary [10]	Anonymity analysis with global passive adversary; the implication of directory trust; sender protection defense	Theoretical anonymity analysis; does not have any practical implementation of anonymity protection to private network, multi-client architecture, or anonymity protection in a research context.

Table 2.1 – continued from previous page

Ref	Title	Key Insight	Project-Relevant Gap
[11]	RAPTOR: Routing Attacks on Privacy [11]	BGP dynamics exploitation to user deanonymization; routing attacks on privacy; methods of active attack	Limited to attack analysis, does not protect privateness of the network, custom routing protection and attack mitigation through research
[12]	Counter-RAPTOR Defense solutions [12]	Active routing attacks defense mechanism; relay fingerprinting and detecting bandwidth; countermeasures to RAPTOR attacks	Prioritizes public net defense, but not the private network defense, own path choice algorithm and research friendly attack tolerance
[13]	CLAPS: Client-Location-Aware Path Selection [13]	Geographic and network proximity consideration; latency reduction at the cost of security; improved path selection algorithms	Is restricted to network path exploitation, not based on research-constrained diversity or path optimization.
[14]	Guard Placement attacks [14]	Path selection algorithm vulnerabilities; strategic placement of guards to compromise the circuit; vulnerabilities of the analysis of the probability of attack	Focuses on public network attacks and Does not address the issue of network privacy guard selection, custom guard placement, and network-oriented guard management.
[15]	Tempest: Temporal Dynamics Analysis [15]	Tempest assists user behavior patterns and circuit lifetime leak information; temporal dynamics of anonymity systems; significance of circuit rotation policy	Lacks privateness to temporal protection, custom rotation policy, or research orientated temporal defense.
[16]	Tor Directory Protocol Attacks [16]	Attacks and improvements of the Tor directory protocol; sound design of authority that consent integrity analysis	Focuses on public network directory services; and they do not discuss autonomous network DA design, custom consensus mechanisms, and research oriented directory management

Table 2.1 – continued from previous page

Ref	Title	Key Insight	Project-Relevant Gap
[17]	Firewall setup to support anonymity usage [17]	Firewall packet filtering at the kernel level (Netfilter (iptables)) A simple administration tool (UFW) with strict rules to reject non-tor packets and record rejected packets with logging enabled	No facilities to work with onion routing systems or generate rules automatically or save privacy focused packet management of private networks
[18]	Application Sandboxing with Firejail [18]	Process isolation and resource restriction reduce security risk; DAC prevents unauthorized access to unauthorized users; sandboxing denies the execution of malicious code	The focus is on generic application security; does not cooperate with onion routing technology; does not support automated launching of browsers and set-up SOCKS5 proxies as proxy resources only
[19]	Comparative Analysis of OR and VPNs [19]	OR provides anonymity to both parties, responder and the intermediary nodes; only node knows about immediate neighbors; messages are encrypted in layers after layers with peeling mechanism	It is only restricted to theoretical analysis, the actual implementation is not yet made in the context of private onion routing using custom path selection and using multi clients.
[20]	Canonical analysis of malicious exit relays [20]	Canonical analysis of malicious exit relays found 65 malicious exit relays by systematic scanning; traffic interception, DNS poisoning, SSL-based MitM attacks found; HoneyConnector infrastructure to detect attacks	Focus on threats to a public Tor network; is silent about security of a private network, controlled exit policy, experiment-oriented exit node system.

As the comparative analysis in Table 2.1 hows, the current research covers the aspects of privacy and anonymity individually, but there is a huge gap in the overall solutions that unite the privacy onion testbeds with multiple clients, automated DNS leak prevention, and firewall administration. The OPG project covers these gaps by offering a single platform to capitulate all these elements into an environment that can support a research friendly and controllable environment.

Chapter 3

Requirement Specifications

The success of any privacy protection software, including the ones suggested in table 3.14, which offers a comprehensive service comparison between the OPG Onion Routing System and the Traditional Tor Network , where the OPG is seen to be more effective in network architecture, circuit building, encryption algorithms, directory services, user interface, DNS protection, Temporary User Profiling and Isolation using Firejail, browser integration, firewall integration, transport obfuscation, and platform support. It has been compared to provide a private controlled environment with 4 hop default circuits (expandable to 8 hops), configurable encryption algorithms, PyQt6 GUI and real time monitoring, mandatory DNS redirection, Firejail sandboxing, autopilot iptables management and optimally tuned to OpenSUSE tumbleweed systems. Onion Privacy Guard (OPG), relies on the presence of a remarkable number of requirements. System requirements are particular specifications, characteristics, and limitations to which OPG needs to conform to operate properly, effectively and without vulnerability. When these requirements are not obtained and applied correctly, the outcome may include vulnerabilities, bottlenecks in their performance, or even a failure of the system, which is particularly essential when it comes to privacy and security solutions.

Requirements specification is not a mere technicality to OPG, rather it is the basis through which all the project is constructed. OPG would offer system-wide, strong anonymity and privacy protection using a onion routing system that is privately implemented, Tor-inspired, using a Directory Authority (DA), entry, middle, exit node roles, client with SOCKS5 proxy, encrypted DNS and firewall management. In order to accomplish this, the requirements should not only cover the built-in functionalities, such as multi-hop path selection, rotating circuits, DNS leak protection, and easy to manage restraints but also the non-functional features, such as performance, reliability, user-ease and privacy congruency.

An effective requirements specification of OPG fulfills a number of functions:

- It helps to create a common vision between the developers, stakeholders, and end-users as to the objectives and extent of the project.
- t informs the process of design and implementation because the entire setup is made to work perfectly to ensure full privacy protection is guaranteed.

- It offers quantifiable testing and validation measures towards assuring that OPG achieves its security and usability goals.
- It ensures effective communication and decision making during the lifecycle of the project and minimizes the chances of scope creep or misunderstandings.

The requirements gathering in the framework of OPG implies close cooperation with privacy advocates, technical specialists, and the real users in order to define the specific challenges in the real-life setting. This involves an examination of the drawbacks of the current privacy tools, knowledge of the changing threat environment, and anticipation of the potential demand of users who do not necessarily possess technical expertise.

3.1 Existing System

3.1.1 Onion Routing and the Tor Network

One of the most popular methods of anonymous communication is Onion Routing. Tor network is a real-life practical example of Onion Routing: user traffic is encrypted several times and transmitted to a random number of nodes (entry, relay/middle, exit nodes) to conceal its sender. This method is very effective in resisting the correlation and eavesdropping of traffic.

- **Strengths**
  a) **Strong Anonymity:** The Anonymity will be strongly achieved by encrypting the traffic at multiple layers.
  b) **Limited Tracking:** Eliminates direct IP tracking and location detection or recognition.
  c) **Node Anonimity:** Provides protection against traffic analysis by making sure that no single node is aware of the recipient and the sender.
- **Limitations**
  a) **Malicious Exit Nodes:** Traffic leaves Tor unencrypted, and therefore attackers can intercept traffic, alter it or inject it at the exit node.
  b) **DNS Leaks:** When a system of the user queries DNS servers not belonging to Tor, they reveal their actual IP address.
  c) **Slow Performance:** Tor may experience high latency, because of multi-hop relaying, it is not optimal when its use is needed in making fast data transfers.

3.1.2 Virtual Private Networks (VPNs)

VPNs establish an encrypted tunnel between the device and a remote server covering the IP address of the user. The VPNs are encrypted with no surveillance by ISP, but it does not give the equal anonymity as Onion Routing offers.

- **Strengths**
  a) **Encryption:** Encrypts the internet traffic, and it can no longer be tracked by the ISP.
  b) **Quicker Browsing:** It provides a quicker and faster experience in browsing than Tor.
  c) **Unrestricted Bypass:** Provides effective use in overcoming geo-restrictions and censorship.
- **Limitations**
  a) **Centralized Trust Model:** VPN is dependent on the user relying on the trust of the VPN provider that can log and monitor their usage.
  b) **Not Resistant to Traffic Analysis:** VPNs (unlike Tor) do not support multi-hop routing and allow users to become victims of metadata analysis.
  c) **Can Be Blocked:** VPN traffic is blocked by several sites and services actively block VPN traffic.

3.1.3 Application Sandboxing with Firejail

Firejail is a tool within the Linux operating system which offers application sandboxing via process isolation, resource limitation as well as discretionary access control (DAC). It provides distinct environments to applications, and they cannot access system resources, network interfaces, or file system paths that they are not authorized to access. Firejail is also a useful tool in sandboxing browsers and any other network facing application to reduce the attack surface and execution of malicious code.

- **Strengths**
  a) **Process Isolation:** Offers process isolation and resource restriction, reducing security threats of compromised applications.
  b) **Access Control:** blocks unauthorized use of system resources by Discretionary Access Control (DAC).

c) **Ephemeral Profiles:** Browser support temporary ephemeral loadable profiles, where cookies, cache and browsing history can be deleted at the conclusion of the session.

d) **Reduced Profiling:** Less long term user profiling, reduces profiling of users of a system over a long term, by isolating the browser state and avoiding lingering tracking information.

- **Limitations**

a) **Platform-Specific:** Firejail is mainly used on Linux based systems, which does not allow cross platform implementation.

b) **Configuration Complexity:** Sandboxing also needs profiling configuration to be properly obtained between security and functionality.

c) **Performance Overhead:** Sandboxing may induce a performance overhead due to its isolation mechanisms

d) **Root Privileges Necessary:** There are certain Firejail capabilities which cannot be fully used without SUID bits or without root access.

3.1.4 Encrypted DNS and DNS Leak Prevention

Traditional DNS queries are not encrypted, which means this gives the ISPs and attackers a chance to monitor or control the DNS queries. DNS-over-TLS (DoT) and DNS-over-HTTPS (DoH) offer encryption to offer security of DNS queries.

- **Strengths**

a) **Anonimity From ISPs:** There is no one to monitor the DNS queries made by the ISPs.

b) **Defends against DNS Spoofing attacks:** Defends aagainst DNS spoofing attacks and cache poisoning attacks.

c) **Encrypting DNS:** Privatizing DNS resolution by encrypting the DNS resolution

- **Limitations**

a) **DNS Resolver probable logging:** In the event that a user is reliant on a third party encrypted DNS service then his queries would still end up logged.

b) **Not Fully Integrated with Tor by Default:** Some applications may still bypass encrypted DNS settings.

c) **Performance Overhead:** Encrypting DNS requests introduces slight latency.

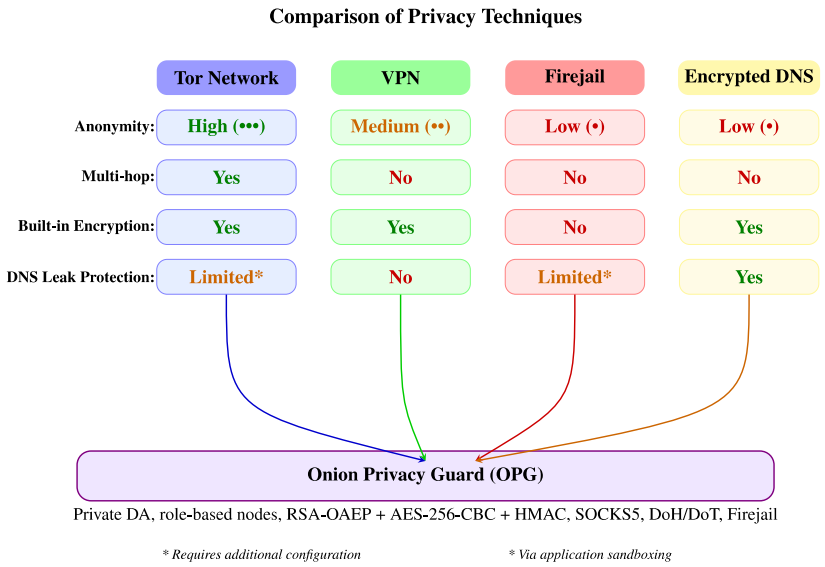

Figure 3.1: Comparison of Privacy Techniques

3.2 Requirements Specifications

The purpose of requirement definition is to clearly outline and convey the decisions on the functional and non-functional requirements of a software project or system. It is a working plan which helps the team to learn what has to be done and how each of the parts should interact with each other. Requirement definition ensures that the completed product meets the purpose and expectations which makes the project to succeed as it ensures that there is a common understanding of those involved in the project, managing them, as well as aiding in the quality assurance effort.

3.3 Functional Requirements

3.3.1 FR-01: Core Privacy Features

- Deployment and operation of a private onion routing network by means of lightweight Virtual Machines as role-based nodes (entry, middle, exit).
- Each node can have one or more IP addresses; paths should not have duplicate IP addresses and subnets where possible.

- Length-prefixed TCP, AES-256-CBC and AES-256-GCM per hop and page integrity over a lengthy chain of user traffic relaying and forwarding.
- rotational scheme by creating circuits; then tearing down; then reconfiguring through OPG client and controller tooling.
- customization of circuit lengths and simple diversity (subnet/IP) in path selection.
- DNS leak prevention and encrypted DNS resolution (DoH/DoT) within or via the circuit.
- Automatic rotation of the circuit and session management with a local SOCKS5 proxy for application.
- Support Deployment of relay nodes as cloud-hosted virtual machines (e.g. DigitalOcean, AWS and similar providers) in different geographic locations and subnets so that multi-hop paths can leverage IP and /24 diversity.

3.3.2 FR-02: System Management

- User-friendly graphical interface allowing to control relay nodes, ip pools, and circuit status.
- Monitoring of Tunnel health, node status, and active circuits in real-time.
- Logging and debugging capabilities for tunnel setup and traffic flow.
- System resource management of lightweight VMs.

3.3.3 FR-03: Security Features

- Firewall integration and management to allow node only communication with authorized peers.
- Application sandboxing and relay node processes.
- safe and secure file management and download pathway using the private onion network.
- Regular security updates of VMs and OPG components.

3.4 Non-Functional Requirements

3.4.1 Performance

- Efficient resource optimization when running multiple VMs.
- The onion routing with the low latency and rotation of circuits in a private testbed.
- Support of multiple simultaneous connections and circuit reconfiguration.
- Quick startup and teardown of relay nodes and circuits.

3.4.2 Reliability

- Availability of relay nodes and circuit paths is high.
- Circuit failover and automatic error recovery.
- Node configurations Backup and restore.
- Node/tunnel failure crash reporting and analysis.

3.4.3 Security

- VM image and OPG script's to have security audits regularly.
- Secured update mechanism of everything (all components).
- security and protection against unauthorized entry to relay nodes and tunnels.
- Secure storage and management of configuration.

3.4.4 Usability

- Intuitive user friendly interface to manage nodes and circuits.
- Comprehensive documentation to set up and operate.
- Context-sensitive contextual assistance and troubleshooting and guide.
- User-friendliness to different user requirements.

3.5 Proposed System

Onion Privacy Guard (OPG) system resolves the shortcomings of other privacy systems in that it implements their own private onion routing network. Only a single Directory Authority (DA) has a list of active nodes; and provides diversity-conscious paths to clients. Nodes are defined with known roles- Entry, Middle, exit and execute a length fixed TCP which removes a single encryption layer, and sends the remaining payload to the next hop. The client compiles multi-hop AES-256-CBC or AES-256-GCM with HMAC-SHA256 integrity with the symmetric keys and carries the keys along by wrapping them with the RSA-2048 public key of each node (OAEP). A local SOCKS5 proxy can permit the unmodified applications to connect using the built circuit. Firewall controls and encrypted DNS (DoH/DoT) supplement the network in order to minimize metadata leakage as well as policy.

3.5.1 The Proposed System Overcomes Existing System Limitations

- **Increased Anonymity and Security:** OPG offers multi-hop cryptography with layered anonymity over a multi-hop circuit (entry $\rightarrow$ middle-1 $\rightarrow$ middle-2 $\rightarrow$ exit) chosen by a DA, and over per-hop circuits with per-hop cryptography. Each hop authenticates integrity (HMAC-SHA256), and removes one AES-256-CBC or AES-256-GCM block, only learns the following hop, and redirects the inner encrypted block. RSA-2048 (OAEP) is used to deliver per-hop AES keys, in which only the intended node would be able to unwrap its session key.
- **Improved Performance and Reliability:** OPG maintains operational control over path selection and circuit rotation, enabling predictable performance in a private testbed. The client periodically refreshes circuits, implements simple subnet/IP diversity and reveals a local SOCKS5 proxy to be able to use in applications without any per-application settings.
- **Comprehensive DNS Protection:** OPG uses DNS leaks by the name resolution inside the circuit through the SOCKS5 proxy attached to the client and combining DNS-over-HTTPS (DoH) or DNS-over-TLS (DoT) to encryption of queries. This method limits the exposure to non-anonymity resolvable information and eliminates correlation of local networks.
- **User-Friendly Interface:** OPG provides user-facing tooling (e.g., local SOCKS5 proxy and status endpoints) that simplifies configuration and monitoring of circuits and nodes, making the system more accessible to users with varying technical expertise. It also recommends a hardened browsing workflow: Firefox launched under Firejail with an ephemeral profile so cookies and tracking artifacts are discarded when the session ends, limiting long-term profiling.

3.6 Use Cases Diagram

The Use Case Diagrams will show how the user manages the VM nodes and will create new circuits, choose IPs, monitor the status, and also send traffic across the private onion network.

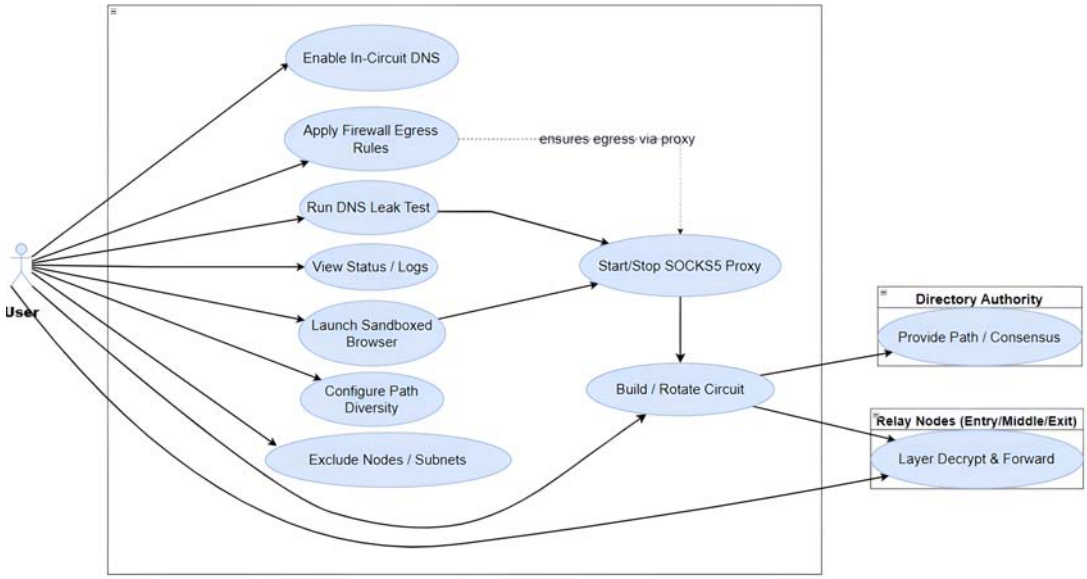

Figure 3.2: OPG Use Case Diagram

The use case diagram in Figure 3.2 which reflects all the functionality of the OPG system in the view of the user. The major player, **the User**, works with several use cases that support privacy-enhanced networking and browsing. The main functions involve constructing and rotating circuits via the Directory Authority that offers path determination and consent data. Path diversity is also configurable and allows the user to block certain nodes or subnets so that their anonymity paths can be customized. The SOCKS5 proxy in the system is the key component and there are a variety of applications which make use of this proxy: launching a sandboxed browser, running DNS leak tests and applying firewall egress rules which guarantee that all traffic goes through the proxy. Other applications like performance of in-circuit DNS, system status and logs and firewall management give the user a full control of his privacy provision. External interactions are also depicted in the diagram: the path and consensus information is given by the **Directory Authority** and the onion packets are decrypted and forwarded by the layer-by-layer by the **Relay Nodes (Entry/Middle/Exit).** This use case model illustrates the implementation of several privacy-enhancing features into a single user-friendly interface by OPG, preserving both the security and anonymity attributes of the underlying onion routing architecture.

Table 3.1: Build or Rotate Circuit (Client)

Use Case ID:	UC-1.1
Use Case Name:	**Build or Rotate Circuit (Client)**
Actor(s):	User
Pre-Conditions:	OPG client has been installed; Directory Authority (DA) is accessible; at least one Entry node and at least one Middle node and Exit node are registered and active.
Priority:	High
Basic Flow:	1. The circuit controls of OPG client are opened by the user. 2. This client asks the DA to provide a path (IP/subnet diversity basic). 3. The client uses onion-building and per-hop keys of AES-256 to construct the onion. 4. The new circuit and session keys are saved by the client. (rotation timer commences) 5. The system checks accessibility of DA and live nodes. 6. Active circuit (roles, IPs, ports) and rotation time are displayed in the system.
Actor Actions	**System Response**
User initiates circuit build/rotate.	The client fetches a DA-selected path, builds keys/onion, updates the active circuit, and displays status.
Alternative Course of Action (if any)	
Actor Action	**System Response**
User forces rotation while DA is unavailable.	The client falls back to local config to build a circuit and updates the display.

Table 3.2: Start SOCKS5 Proxy (Client)

Use Case ID:	UC-1.2
Use Case Name:	**Start SOCKS5 Proxy (Client)**
Actor(s):	User
Pre-Conditions:	OPG client is installed and is operational; it has an active circuit or may be created.
Priority:	High
Basic Flow:	1. The primary tab is opened by the user. 2. The SOCKS5 control is found by the user. 3. The user opens clicking on the SOCKS5. 4. The client guarantees a good circuit (connects/turns around where necessary). 5. The client initiates a local SOCKS5 proxy (e.g. 127.0.0.1:9051). 6. proxy status and circuit information are shown in the system.
Actor Actions	**System Response**
1. User clicks on the SOCKS5 button in the primary tab.	The client starts the SOCKS5 listener, validates/creates a circuit, and shows "SOCKS5 Active" with the listening address and current circuit.
Alternative Course of Action (if any)	
Actor Action	**System Response**
User clicks on an already active SOCKS5 button.	The client stops the listener. The button changes to inactive, and status shows "SOCKS5 Stopped".

Table 3.3: Apply DNS Leak Prevention (Route DNS via SOCKS5)

Use Case ID:	UC-1.3
Use Case Name:	**Apply DNS Leak Prevention (Route DNS via SOCKS5)**
Actor(s):	User
Pre-Conditions:	Operation of OPG client is established and operational; SOCKS 5 listener is present (e.g. 127.0.0.1:9051); user is authorized to execute the OPG DNS leak fix script.
Priority:	High
Basic Flow:	1. The user opts to use DNS leak prevention (through either GUI prompt or terminal). 2. The system/script sets the resolver system to 127.0.0.1 and launches a local DNS proxy. 3. The DNS proxy will redirect the queries by the local SOCKS5 proxy (the active circuit). 4. The system is adding redirect rule of ip tables/UFW to redirect all DNS to local proxy port. 5. The system shows protection status in DNS and the local proxy port in status /log panel.
Actor Actions	**System Response**
1. User confirms "Apply DNS Leak Prevention".	The system runs the OPG DNS leak fix (resolver set to 127.0.0.1, local proxy started, redirect rules applied) and shows active status.
Alternative Course of Action (if any)	
Actor Action	**System Response**
User attempts to re-apply DNS leak prevention while already active.	The system notifies the user that DNS leak prevention is already active and shows the current status.

Table 3.4: Enable Firewall Egress Control

Use Case ID:	UC-1.4
Use Case Name:	**Enable Firewall Egress Control**
Actor(s):	User
Pre-Conditions:	OPG client has been installed; firewall elements are in place; there is SOCKS5 listener or circuit.
Priority:	High
Basic Flow:	1. The primary tab is opened by the user. 2.The Firewall Protection button is found at the primary tab by the user. 3. The user clicks the button Firewall Protection. 4. The system sets the iptables/UFW to block the non-circuit egress and allow the connection of SOCKS5 or node only. 5. The system changes the Firewall Protection button to green color with the active status. 6. The system shows the status of the firewall in the status panel.
Actor Actions	**System Response**
1. User clicks on the Firewall Egress button in the primary tab.	The system activates egress control rules, allowing only circuit-bound traffic; status shows rule counts and permitted endpoints (SOCKS5, DA, nodes).
Alternative Course of Action (if any)	
Actor Action	**System Response**
User clicks on an already active Firewall Protection button.	The system disables firewall protection. The button changes color to red indicating inactive status. The status panel updates to show "Firewall Protection Disabled".

Table 3.5: Maintain Active SOCKS5 Streams During Circuit Rotation

Use Case ID:	UC-2.1
Use Case Name:	**Maintain Active SOCKS5 Streams During Circuit Rotation**
Actor(s):	User
Pre-Conditions:	Client server OPG has been installed and is running; SOCKS5 running and an application stream is passing through the circuit.
Priority:	Medium
Basic Flow:	1. An application is connecting a TCP stream via the SOCKS5 proxy in the area. 2. A circuit rotation takes place (forced or automatic). 3. Existing streams proceed across the established connections where possible. 4. New connections are made with the new circuit that has been rotated. 5. The rotation is logged in the system and the continuity status displayed.
Actor Actions	**System Response**
User monitors a stream during circuit rotation.	The system rotates circuits in the background; existing SOCKS5 streams typically continue; status logs reflect continuity.
Alternative Course of Action (if any)	
Actor Action	**System Response**
User manually triggers rotation during an active stream.	The system rotates circuits; existing streams are preserved when possible; new connections use the new circuit.

Table 3.6: Test DNS Leak Protection

Use Case ID:	UC-3.1
Use Case Name:	**Test DNS Leak Protection**
Actor(s):	User
Pre-Conditions:	OPG software is installed and running. DNS leak protection is applied (if run previously). User navigates to the DNS leak test section.
Priority:	Medium
Basic Flow:	1. The DNS Settings section is opened by the user. 2. The user will find the button of "Test DNS Leak Protection. 3. The user clicks on the button. 4.A sequence of DNS leak tests are done by the system. 5. The system interrogates test servers using various means. 6.The system compares the results in order to identify any possible leakage. 7. The system presents detailed results of the tests to the user.
Actor Actions	**System Response**
User clicks "Test DNS Leak Protection".	The system performs DNS leak tests, analyzes the results, and displays comprehensive test results to the user.
Alternative Course of Action (if any)	
Actor Action	**System Response**
Test detects a DNS leak.	The system highlights the detected leak with details about the cause. The system provides recommendations to fix the leak. The user can apply the recommended fixes directly or manually address the issue.

Table 3.7: Multi-Client Concurrent Access to Onion Network

Use Case ID:	UC-4.1
Use Case Name:	**Multi-Client Concurrent Access to Onion Network**
Actor(s):	Multiple Users (Client 1, Client 2, ..., Client N)
Pre-Conditions:	Entry, Middle, Exit of the OPG network running and registered with DA; Key synchronization script has been run so that each client should have a unique client ID generated automatically and client-specific RSA keys and TLS certificates should be distributed to them.
Priority:	High
Basic Flow:	1. Client 1 initiates OPG client and creates/ loads client ID (e.g., `e8d5eb4ccf07`). 2. Client 1 loads over client-specific directories (`keys/<node>/<client_id>/rsa_public.pem`). 3. Client 1 connects SOCKS5 proxy on the following address `127.0.0.1:9051` and begins routing traffic. 4. Client 2 initiates OPG client on another client machine and creates/loads its own client ID (e.g., `a3f9c2b1d8e4`). 5. Client 2 retrieves its own client-specific RSA public keys out of isolated directories. 6. Client 2 initiates SOCKS5 proxy on its own local port `127.0.0.1:9051` and is also routing traffic by itself. 7. The onion network nodes construct independent circuits in both clients without any conflicts over the key. 8. Nodes decode packets with client-specific RSA private key and TLS certificates matching to the IDs of each client. 9. Several clients utilise the network simultaneously and not interfering with one another.
Actor Actions	**System Response**
Client 1 connects to SOCKS5 proxy.	System generates client ID, loads client-specific keys, builds circuit, and routes traffic.
Client 2 connects to SOCKS5 proxy.	System generates separate client ID, loads independent keys, builds separate circuit, and routes traffic without affecting Client 1.
Alternative Course of Action (if any)	
Actor Action	**System Response**
Client attempts to use non-existent client-specific keys.	System automatically copies keys from fallback paths to client-specific directories and proceeds with circuit construction.
Multiple clients attempt simultaneous circuit rotation.	Each client rotates its circuit independently; node handles concurrent encrypted packets from different clients without conflicts.

Table 3.8: Configure Extended Circuit Hop Count (up to 8 Hops)

Use Case ID:	UC-4.2
Use Case Name:	**Configure Extended Circuit Hop Count (up to 8 Hops)**
Actor(s):	User
Pre-Conditions:	OPG client and GUI are installed and running; Directory Authority is reachable; at least one Entry node, one Exit node, and sufficient Middle nodes are registered (hosted as cloud VMs such as DigitalOcean instances in distinct locations/subnets).
Priority:	Medium
Basic Flow:	1. The user opens the Network/Performance or Circuit Configuration tab in the GUI. 2. The user locates the setting for "Circuit Hops" (number of hops per circuit). 3. The user increases the hop count from the default value (e.g., 4) towards a higher value, up to the supported maximum of 8 hops. 4. The client persists the desired hop count into `configuration.json` and updates in-memory settings. 5. On the next circuit build or rotation, the client requests a path of the configured length from the Directory Authority. 6. The Directory Authority selects a path that still enforces IP and /24 subnet diversity, mapping the configured hop count to 1 Entry, up to 6 Middles, and 1 Exit where available. 7. The active circuit view in the GUI shows the new hop count and the list of node roles and IPs.
Actor Actions	**System Response**
User increases hop count to a higher value (e.g., from 4 to 6 or 8).	The client validates the value (clamped to the 2–8 hop range), saves it to configuration, and future circuits are built with more middle hops using the available cloud-hosted relay nodes.
Alternative Course of Action (if any)	
Actor Action	**System Response**
User attempts to configure more hops than available nodes can support (e.g., requests 8 hops but only one middle node is registered).	The Directory Authority still enforces role and diversity constraints, resulting in a shorter path (e.g., 3–4 hops). The client logs the effective hop count and displays the actual circuit length in the GUI.
User reduces hop count back to a smaller value (e.g., 3 or 4).	The client updates the configuration accordingly; subsequent circuits use fewer middle hops, trading some anonymity for lower latency.

Table 3.9: Launch Sandboxed Browser via OPG Client

Use Case ID:	UC-4.3
Use Case Name:	**Launch Sandboxed Browser via OPG Client**
Actor(s):	User
Pre-Conditions:	OPG client is installed and running; SOCKS5 proxy is active (e.g. `127.0.0.1:9051`); Firejail and Firefox (or equivalent browser) are installed on the client system.
Priority:	High
Basic Flow:	1. The user navigates to the Security Tools tab in the OPG GUI. 2. The user verifies that Firejail is installed by clicking the "Check Firejail" button; the GUI displays Firejail status as available. 3. The user clicks the **"Launch Sandboxed Browser"** button. 4. The client verifies that the SOCKS5 proxy is active and obtains its listening address. 5. The client starts a Firejail sandbox with an ephemeral Firefox profile bound to the SOCKS5 proxy. 6. The browser window opens and all HTTP/HTTPS traffic is routed through the OPG circuit. 7. The GUI updates the browser status to show the sandboxed session is active. 8. On browser exit, the sandbox and temporary profile are destroyed, erasing cookies, cache, and browsing history.
Actor Actions	**System Response**
User clicks "Launch Sandboxed Browser".	The client checks SOCKS5 status, verifies Firejail availability, launches Firejail with an ephemeral browser profile configured to use the OPG SOCKS5 proxy, and updates the status panel with launch confirmation.
Alternative Course of Action (if any)	
Actor Action	**System Response**
User attempts to launch the browser while SOCKS5 is inactive.	The client refuses to launch the browser, shows an error message, and prompts the user to start the SOCKS5 proxy first.
Firejail is not installed on the system.	The GUI displays a warning that Firejail is not found and provides instructions for installing Firejail or an alternative sandboxing mechanism.
User closes the sandboxed browser window.	Firejail tears down the sandbox and deletes the temporary browser profile; the OPG client status view reflects that the sandboxed session has ended.

Table 3.10: Check Public IP Address via OPG Circuit

Use Case ID:	UC-5.1
Use Case Name:	**Check Public IP Address via OPG Circuit**
Actor(s):	User
Pre-Conditions:	OPG client is running; SOCKS5 proxy is active with an established circuit to the exit node.
Priority:	Medium
Basic Flow:	1. The user opens the Security Tools tab in the OPG GUI. 2. The user clicks the **"Check My IP Address"** button. 3. The client sends an HTTPS request to an IP-checking service (e.g., `ifconfig.me` or `api.ipify.org`) via the active SOCKS5 proxy and OPG circuit. 4. The exit node forwards the request to the external IP service. 5. The service responds with the public IP address as seen from the Internet. 6. The client receives the response and displays the exit node's public IP in the GUI. 7. The user verifies that the displayed IP differs from their true local IP, confirming anonymity.
Actor Actions	**System Response**
User clicks "Check My IP Address".	The client queries an IP-checking service through the OPG circuit and displays the public IP address of the exit node in the GUI with a timestamp.
Alternative Course of Action (if any)	
Actor Action	**System Response**
User attempts IP check while SOCKS5 proxy is inactive or circuit is down.	The client shows an error: "No active circuit. Please start the SOCKS5 proxy and establish a circuit first."
IP-checking service is unreachable or times out.	The client retries with a fallback IP-checking service and logs the timeout; if all services fail, displays an error with network diagnostics.

Table 3.11: Configure and Validate Node Settings

Use Case ID:	UC-5.2
Use Case Name:	**Configure and Validate Node Settings**
Actor(s):	User (System Administrator or Advanced User)
Pre-Conditions:	OPG client and GUI are installed; the user has access to the Node Configuration tab; at least one node (Entry, Middle, or Exit) is available for configuration.
Priority:	High
Basic Flow:	1. The user opens the Node Configuration tab in the OPG GUI. 2. The user enters or modifies the Kali host IP address used for local coordination. 3. For each node type (Entry, Middle1, Middle2, Exit), the user adds or updates IP addresses, ports, and usernames in the configuration table. 4. The user clicks **"Validate Configuration"** to check the syntax and reachability of all configured nodes. 5. The client validates IP formats, port ranges, and attempts test connections to each node. 6. The GUI displays validation results, highlighting any errors (e.g., invalid IP, unreachable host, port conflicts) or warnings. 7. If validation passes, the user clicks **"Save Configuration"** to persist settings to `configuration.json`. 8. The system confirms that the configuration is saved and ready for circuit building.
Actor Actions	**System Response**
User clicks "Validate Configuration".	The client validates all node IP addresses, port numbers, and connectivity; displays a summary with pass/fail status for each node and any detected errors or warnings.
User clicks "Save Configuration" after successful validation.	The client writes the updated node configuration to `configuration.json`, logs the save action, and updates in-memory settings for immediate use in circuit construction.
Alternative Course of Action (if any)	
Actor Action	**System Response**
Validation detects errors (invalid IP, unreachable node, duplicate ports).	The client highlights the problematic fields in red, displays detailed error messages, and prevents saving until issues are resolved.
User clicks "Refresh from Consensus" to fetch node list from Directory Authority.	The client queries the DA for the current consensus, parses node information, and populates the configuration table with live node data; user reviews and saves if acceptable.
User clicks "Test Connections" to verify node reachability.	The client attempts to establish TCP connections to each configured node and reports success or failure with latency metrics in the GUI.

Table 3.12: Configure TLS Obfuscation for Client-Entry Link

Use Case ID:	UC-5.3
Use Case Name:	**Configure TLS Obfuscation for Client-Entry Link**
Actor(s):	User (System Administrator or Privacy-Conscious User)
Pre-Conditions:	OPG client is installed; TLS certificates and keys are available; the user has access to the Security & Encryption Configuration tab.
Priority:	Medium
Basic Flow:	1. The user opens the Security & Encryption Configuration tab in the OPG GUI. 2. The user locates the Transport Security section. 3. The user checks the **"Enable TLS Obfuscation"** checkbox to wrap the client-to-entry connection in a TLS tunnel. 4. The user specifies or browses to the TLS certificate file path (e.g., `/path/to/certs/server.pem`). 5. The user specifies or browses to the TLS private key file path (e.g., `/path/to/certs/server.key`). 6. The user verifies the Replay Window setting (default: 120 seconds) for replay attack protection. 7. The user clicks **"Save Configuration"** to persist the TLS obfuscation settings. 8. On the next circuit build, the client establishes a TLS-wrapped connection to the entry node, making the onion traffic appear as standard HTTPS to network observers.
Actor Actions	**System Response**
User enables TLS obfuscation and provides certificate/key paths.	The client validates that the certificate and key files exist and are readable, then saves the TLS configuration; future circuits will use TLS for the client-entry link.
Alternative Course of Action (if any)	
Actor Action	**System Response**
User enables TLS obfuscation but certificate or key file is missing or invalid.	The client displays an error: “Certificate or key file not found or invalid. Please provide valid TLS credentials.” Configuration is not saved until the issue is resolved.
User disables TLS obfuscation after it was previously enabled.	The client removes the TLS wrapping; subsequent circuits use plaintext length-prefixed TCP between client and entry node, saving overhead but reducing obfuscation.

Table 3.13: Save, Load, and Reset Configuration

Use Case ID:	UC-6.1
Use Case Name:	**Save, Load, and Reset Configuration**
Actor(s):	User
Pre-Conditions:	OPG client and GUI are running; user has made changes to node, security, network, or firewall settings in the GUI.
Priority:	High
Basic Flow:	1. The user modifies configuration settings in any tab (Node Config, Security, Network, Firewall). 2. The user clicks the **"Save Configuration"** button (visible at the bottom of each configuration tab). 3. The client validates the modified settings for correctness (IP formats, port ranges, file paths). 4. If validation passes, the client writes the updated configuration to `configuration.json` and displays a success confirmation. 5. The user can optionally click **"Export Config"** to save the current configuration to a custom file location (e.g., `my_config_backup.json`). 6. Later, the user can click **"Load Configuration"** to open a file dialog, select a previously saved configuration file, and restore those settings. 7. If the user wants to revert all changes, they click **"Reset to Defaults"**, which restores factory default settings for all parameters.
Actor Actions	**System Response**
User clicks "Save Configuration".	The client validates and writes settings to `configuration.json`; GUI displays "Configuration saved successfully" with a timestamp.
User clicks "Export Config" and chooses a file path.	The client exports the current configuration as a JSON file to the specified location; confirms export success.
User clicks "Load Configuration" and selects a JSON file.	The client reads and validates the imported configuration, updates all GUI fields to match, and logs the import action; user must click Save to persist.
User clicks "Reset to Defaults".	The client prompts for confirmation ("Are you sure? This will erase all custom settings."), then restores default values for all configuration parameters and updates the GUI.
Alternative Course of Action (if any)	
Actor Action	**System Response**
User attempts to save invalid configuration (e.g., malformed IP, missing required fields).	The client displays validation errors, highlights problematic fields, and prevents saving until corrected.
User loads a configuration file with syntax errors or incompatible schema.	The client displays an error: "Invalid configuration file. Please check the format and try again." and does not apply the changes.
User clicks "Validate All" before saving.	The client runs comprehensive validation checks across all tabs and reports a summary of issues (errors and warnings) without saving; user can then fix issues before saving.

3.7 Comparison of Privacy Techniques

Table 3.14 provides a comparative analysis of existing privacy techniques, including VPNs, Tor, and Encrypted DNS alongside the proposed Onion Privacy Guard (OPG) system. Each technique is evaluated based on encryption, anonymity level, traffic obfuscation, resistance to analysis, and ease of use. The comparison shows that while existing tools address individual privacy aspects, OPG offers a private, DA-driven onion network with multi-layer encryption, in-circuit DNS protections, firewall isolation, and application sandboxing. This approach mitigates common vulnerabilities like DNS leaks, proxy misconfigurations, and long-term user profiling, ensuring consistent, enforced privacy across applications routed via the client's SOCKS5 interface.

Table 3.14: Comparison of Existing Privacy Techniques with OPG

Technique	**Strengths**	**Limitations**
Tor Network	Strong anonymity, multi-hop routing, prevents IP tracking.	Malicious exit nodes, DNS leaks, slow performance.
VPNs	Encrypts traffic, prevents ISP tracking, faster than Tor.	Centralized trust, no multi-hop routing, can be blocked.
Encrypted DNS	Prevents ISP tracking, protects against spoofing.	Potential logging by DNS resolvers, not fully integrated with Tor.
Onion Privacy Guard (OPG)	Private DA with role-based nodes; RSA-OAEP + AES-256-CBC + HMAC; SOCKS5 client; DoH/DoT; basic path diversity.	Single DA (no consensus), no PFS, variable-size JSON packets, no exit policies yet.

The techniques are all assessed in terms of encryption, level of anonymity, resistance to analysis obfuscation of traffic, level of difficulty of use. The comparison raises that, whereas the existing tools are focused on the individual privacy issues, OPG provides the private onion network (DA-oriented) with multi-layer encryption, in-circuit DNS security, firewall isolation and application sandboxing. It applies to minimize the typical weaknesses such as DNS leakage, proxy misconfigurations, and user profiling over the long term and guarantees uniform, enforced privacy between the applications directed via the client SOCKS5 tunnel.

Although there are personal and individual level advantages of these specific security techniques, none of these methods offers a complete safeguard against all the privacy concerns. This highlights the necessity of a unified method, like the Onion Privacy Guard (OPG) which is a group of privacy enhancing technologies used together to overcome the deficiencies of each method. The following figure presented below gives a graphical overview of this comparison showing the way, OPG combines the strengths of Tor, VPNs, Firejail, and Encrypted DNS in one solution.

Chapter 4

Design

System design is an important stage of development of any software project because it is the point of transition between the requirements acquired in the previous stages and the very implementation, which will take place. In the case of Onion Privacy Guard (OPG), design will be particularly important because the privacy, security, usability, and performance requirements are complex. This chapter discusses how OPG designed their own private onion routing system, which was Tor-inspired and how this system combines to create a practical anonymity environment in a research friendly, controlled setting, with a Directory Authority (DA), role-based relay nodes (entry, middle, exit), and a client with a local SOCKS5 proxy.

The design cycle of OPG starts by initially analyzing the system and user requirement, and converting them into detailed blueprint through which the development team follows them. The network architecture (DA, nodes, client), core modules, interfaces, and data flows (length-prefixed TCP, onion layers and control APIs) are mentioned in this blueprint. Defining these items clearly during the design phase will make all the stakeholders have a common vision of how the system will be and how it will behave, eliminate ambiguity thus minimise the risks of making changes that might be costly to the system later in the process.

A unique characteristic of the OPG approach of designing is that both the top-down and bottom-up methodologies are incorporated. Top-down perspective is the one that determines significant subsystems (Directory Authority, Node Services, Client/Proxy, DNS/Firewall controls) and their interactions. The bottom-up perspective is based on the strong per-hop cryptography (RSA-OAEP key wrapping, AES-256-CBC or AES-256-GCM, HMAC-SHA256), replay protection, circuit management, and transport concerns (length-prefixing, optional TLS obfuscation). The combination of such ideas brings about architectural harmony and firm implementation.

4.1 System Architecture

The system architecture of the OPG has been presented in non-technical language to give a good overview to the entire stakeholders. The OPG has the **end user** interacting with the

OPG application via a graphical user interface (GUI) on their desktop or laptop and carry out administrations and updates respectively.

A variety of privacy-enhancing features are incorporated into OPG, coordinated to its private onion network: a Directory Authority to do consensus and path selection, onion relay nodes (entry/middle/exit) to do layered forwarding, a client with SOCKS5 proxy to support application compatibility, in-circuit encrypted DNS, and firewall control to provide egress control. To be a safer browsing, OPG suggests that Firefox be started under Firejail and a temporary profile created so that the cookies, cache and history are deleted upon closing it. The user traffic goes around the circuit selected by the DA; status of the system and the views of consensus can be viewed and configuration and logs are kept locally.

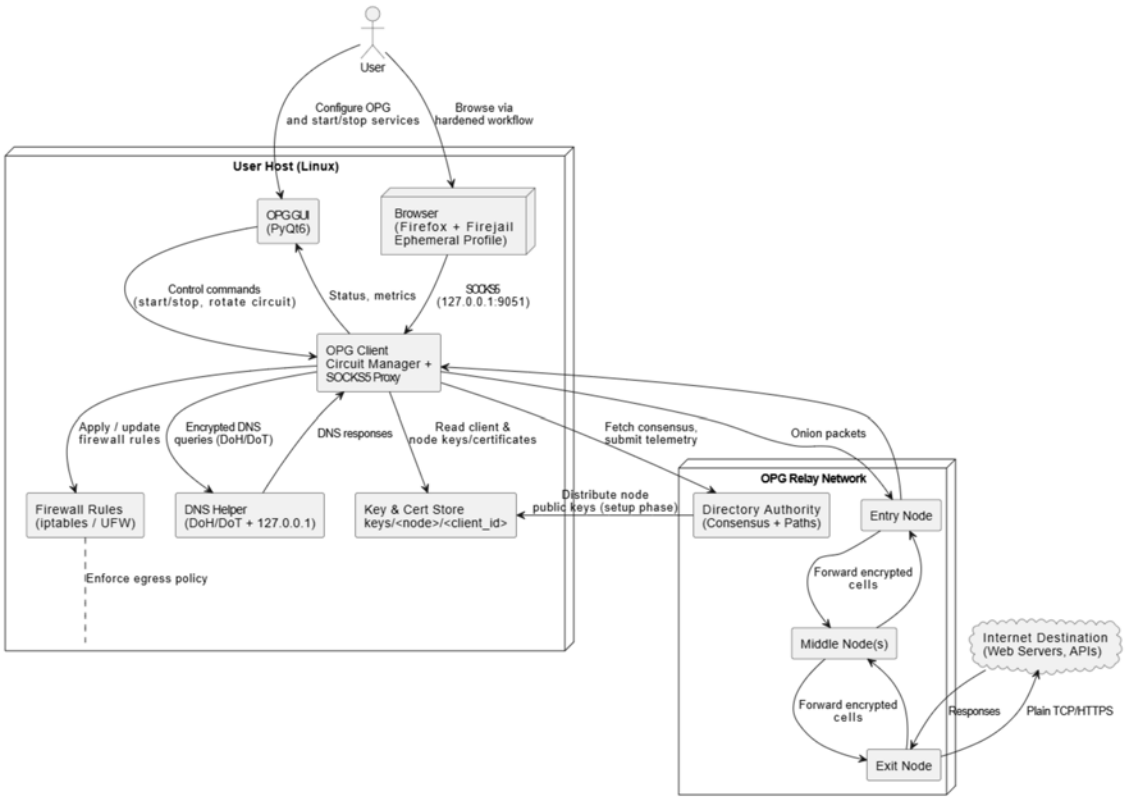

Figure 4.1: OPG System Architecture

System Architecture Overview: This figure shows the communication between the OPG client who runs on a Linux host and the private OPG relay network. At the host user interface level, the PyQt6 OPG GUI triggers commands to the OPG Client Circuit Manager + SOCKS5 Proxy (e.g., start, stop, rotate circuit), whereas hardened applications, e.g. Firefox in a Firejail ephemeral profile set the traffic through the local SOCKS5 socks endpoint, 127.0.0.1:9051. The client collaborates with three local services: a DNS helper that sends encrypted DNS queries (DoH/DoT through 127.0.0.1), client and key store that has

node and client credentials, and to maintain a strict egress policy, the firewall rules (iptables/UFW) are used to ensure that the traffic only goes out over the anonymity circuit. The Distribution of node public keys and consensus information, on the network side, is provided by the Directory Authority to allow all the client to build paths through the OPG relay network composed of entry, middle, and exit nodes. Packets are onion encrypted and transmitted hop-by-hop over a special circuit of entry nodes, middle nodes, and exit nodes. The exit node does a DNS resolution and connects to external Internet resources using plain TCP/HTTPS and routes answers back to the client using the same circuit. This architecture provides anonymity to users and restricts all the DNS and transport traffic within the OPG network, which helps to avoid metadata and DNS leakage.

The high level context diagram in Figure 4.2 depicts the interactions between the OPG system and the main actors. The context diagram visually shows the external entities and their relations to OPG giving us an overview of the system boundaries and external relationships.

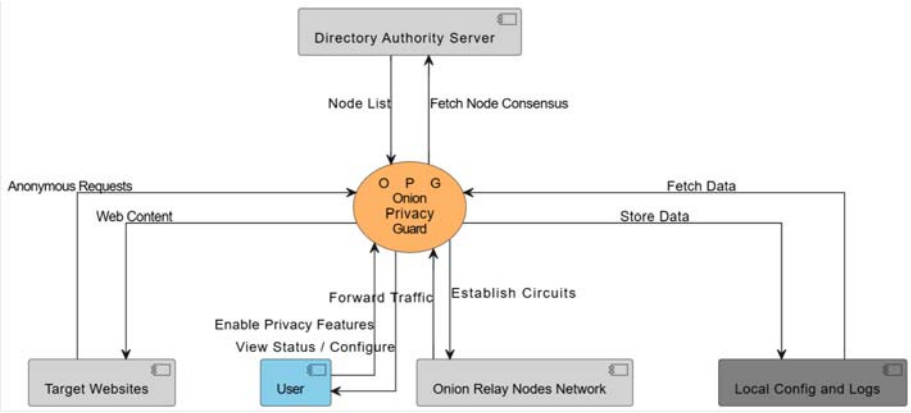

Figure 4.2: High Level Context Diagram for Onion Privacy Guard (OPG)

The above context diagram shows external components and how they are related to the Onion Privacy Guard (OPG) system. In the centre, OPG is an intermediate between a final consumer, the server of the Directory Authority and the network of onion relay nodes, Internet destination sites and the local configuration and logs. Privacy features and status are enabled by the user in the OPG interface, the Directory Authority provides node lists and consensus, which is then used to create circuits and forward traffic over the relay network, the user issues an anonymous request to the target websites and receives web content back, and writes or writes data to local configuration and log files. These interactions, when combined, bring out the system boundary of OPG and its coordination of external components to provide the user anonymous browsing and observable system status

4.2 Design Constraints

The design constraints are limitations, regulations or conditions that affect the design of OPG. These are the major restrictions to achieving privacy and security goals of the system. The constraints that apply to the OPG system include the following:

4.2.1 Target Platform

OPG is designed and configured for Linux machines. Ensuring consistent functionality on mainstream Linux distributions and toolchains (systemd, iptables/nftables, OpenSSL) is a key constraint.

4.2.2 Security and Privacy Standards

Each of the components should comply with the industry best practices in security and privacy such as safe configuration file storage and periodic security patching and also providing authentication token to nodes, so no local or anonymous node get registered to the Directory Authority (DA). Only authorized nodes get registered having the right authentication token.

4.2.3 Performance

The system should not only have low network routing latency but also have short response time to user actions even when the privacy layers add overhead to the system, which is good for private organizations as the number of clients are less as compared to that of a traditional tor network.

4.2.4 Usability

The GUI should be user friendly and easy to use by users with different levels of technical skills and be complex enough to support the sophisticated functions with simplicity.

4.2.5 Integration with Third-Party Tools

OPG is dependent on external (Firejail, DNS resolvers, firewall utilities). A design constraint of these tools is to ensure smooth integration and accommodate changes or updates.

4.3 Design Methodology

Onion Privacy Guard (OPG) is a project based on the Waterfall Model, which is a linear and organized model of software development. Every stage such as requirements, design, implementation, testing, deployment and maintenance is done and then followed by the next one and then there is proper documentation and sustainability is maintained. The approach is highly appropriate to the project since it enables us to clearly state privacy and security requirements initially, transform them into a solid system architecture, and test their implementation at each phase in a well-defined manner. The sequential nature of the Waterfall Model also gives us an opportunity to chart the technical difficulties, e.g., multi-hop circuit routing, DNS leak prevention, and system integration, prior to any code being written.

To make the operations flow of OPG even clearer, the activity diagram is provided as well, depicting the chain of the main activities, the points of decisions, and the interaction of the system parts. This graphical representation assists the developers and the stakeholders to have an insight into the logic of the system, user interaction, and movement between main processes of the OPG workflow.

4.3.1 Waterfall Model

To implement Onion Privacy Guard (OPG) system, we are adhering to the waterfall model as our software development methodology. The use of this method organizes the project into sequential steps; which are requirements analysis, design, implementation, testing, and deployment, each of which has to be finished before the next. Through the waterfall model, we are guaranteed of a clear and well organized workflow and proper documents at each phase of the work and therefore less wastage of work is done and the final product is well implemented as per the specifications and the requirements of the users.

- **Requirements Analysis:** This is done by collecting and specifying user and system requirements, on which all other phases are based.
- **Analysis & Design:** Converting requirements into a system architecture and design specifications, including detailed system design and component specifications.
- **Implementation:** Coding the system segments according to the design specifications, developing all modules and components of the OPG system.
- **Testing & Verification:** Thorough testing includes unit testing of each component, integration testing of interactions, system testing of the entire solution, and acceptance testing to ensure that it is ready and satisfies the users.
- **Deployment & Maintenance:** Deploying the OPG system to the target environment and providing ongoing maintenance, updates, and support to ensure continued functionality and security.

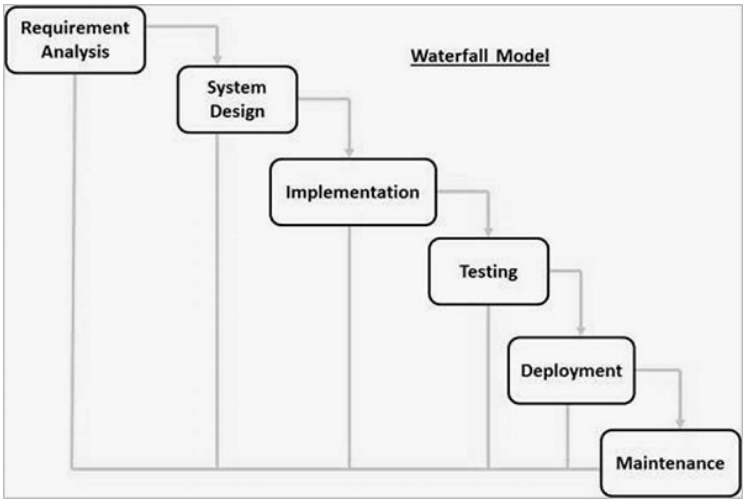

Figure 4.3: Waterfall Model Methodology for OPG Development Lifecycle

This Waterfall Model diagram represents the linear and sequential approach adopted in the OPG project. It illustrates how each development phase flows into the next, beginning from requirements analysis and progressing through design, implementation, testing, deployment, and maintenance. The Waterfall Model emphasizes completing each stage fully before moving to the next, ensuring structured documentation, clear project milestones, and disciplined progress tracking.

4.3.2 Activity Diagram

Further, an Activity Diagram is given to demonstrate how the user will interact with the main tab controls to set privacy features.

The activity diagram provides the visual representation of the primary processes and workflows in the Onion Privacy Guard (OPG) system. It draws the flow of activities, decision points, and user system interaction path for the first user input to final outputs. The flow of activities explains the way the core elements in the system like circuit creation, encrypted routing, DNS leak prevention and user status monitoring are implemented in the system as demonstrated by the diagram. Such visualization can help developers and stake holders comprehend the logic of operations and main transitions in OPG, as well as the processes are clear and transparent under the system.

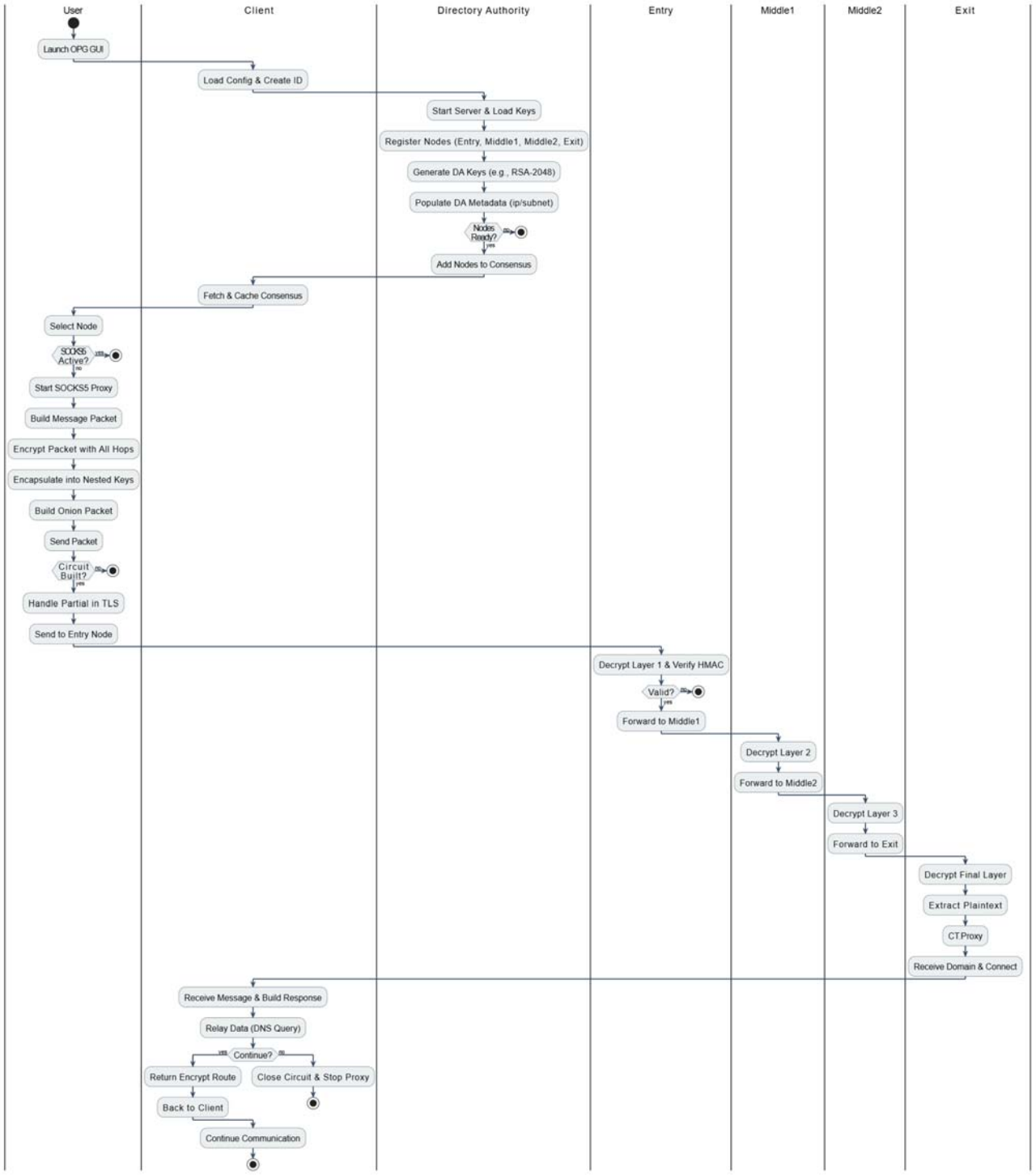

Figure 4.4: Activity Diagram: Primary Tab Controls

The activity diagram is a description of the entire workflow of the OPG Onion Privacy Gateway showing how the user interface interacts with the system components that make up the whole system to offer secure and anonymous communication. The flow starts with the user initiating the OPG application, which makes the client retrieve the configuration of the application and download the current consensus of active nodes provided by the Directory Authority. There are also major decision points highlighted in the diagram such as the operating mode chosen (SOCKS5 proxy or a direct messaging), the use of TLS obfuscation (enabled or disabled) and whether the session would continue or terminate. It also demonstrates the necessary internal operations, including creation of circuits, a layered encryption, exchange of cryptographic keys, sending packets to the Next node through the Entry, Middle and Exit node. In general, the diagram effectively describes the integration of user activities and system logic to provide privacy, routing and data relay capabilities during a default session in the OPG environment.

4.4 High Level Design

This section presents the overview of the major components and their interactions within OPG.

4.4.1 Component Diagram

The component diagram is a graphical and visual representation of the key modules of OPG and how they are connected to each other. Major components include:

- **User Interface (UI):** Handles user interactions and displays system status through the PyQt6 GUI (`opg_gui.py`, `run_gui.py`, `run_opg_gui.sh`).
- **OnionClient (SOCKS5 / Circuit Manager):** Controls Circuit routing connections and circuits of the Tor-like type with the help of the client module (`client.py`), SOCKS5 proxy is provided through client.py.
- **System Integration Modules:** Controls the firewall rules and the DNS leak prevention using iptables/UFW scripts and `fix_dns_leaks_opg.sh` to isolate traffic.
- **Firejail Sandboxed Browser:** Safer browsing because Firefox may be secured with an ephemeral profile and sandboxed with Firejail.
- **Relay Node Software (Entry/Middle/Exit):** Implements the onion routing relay functionality with `node.py` and configuration files of entry node, middle node, and exit node.
- **Directory Authority (Consensus & Path Selection):** Directory Authority supports path selection and provides network consensus through `directory_authority.py` and startup scripts.

- **Configuration & Keys:** tores and manages system configurations, consensus data and cryptographic keys through `configuration.json`, `consensus.json`, and `key_cert.py` with associated certificates and keys.
- **Monitoring & Logs:** Diagnostic, error and event logs of the system are registered in the Logger module to debug and audit the system.
- **Deployment & Helper Scripts:** Offers deployment automation and assistance in operation using startup scripts and utility scripts.
- **Miscellaneous Artifacts:** This contains certificates, keys and Python bytecode caches (`pycache`) used by different system components.

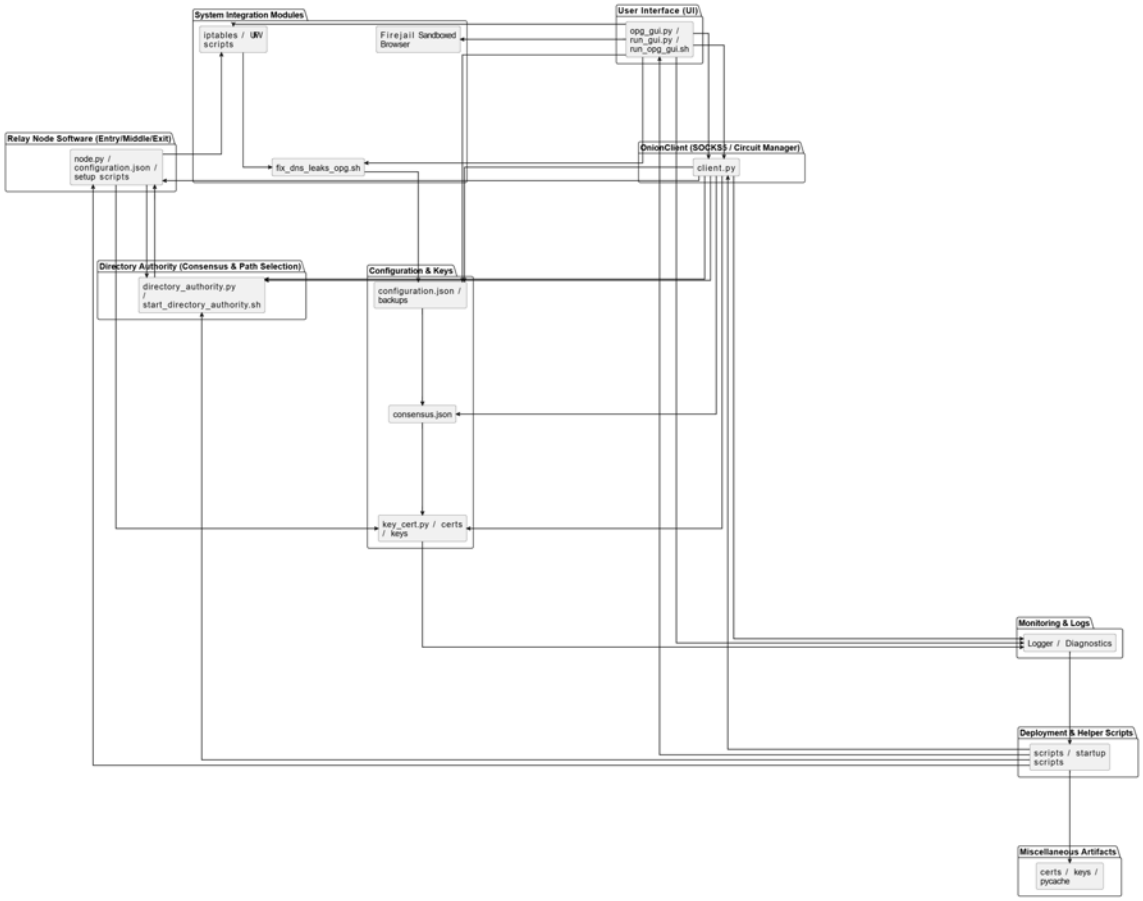

Figure 4.5: OPG High-Level Component Diagram

This component diagram shows the major architecture and connectivity of key components of the OPG system. The figure will display the interaction between various modules with each other that will include GUI interface, Directory Authority, onion routing modules, DNS management, firewall controls and

security modules. It shows the modular approach to design which has a clear separation of concerns and a well-defined interface between the components making it maintainable, scalable and easy to test.

4.4.2 Deployment Diagram

OPG is implemented on Linux machines. The deployment diagram indicates that the application is executing on a Linux host, and communicating with the Directory Authority, onion relay nodes, DNS resolvers (through DoH/DoT within the circuit) and storing configuration/logs locally.

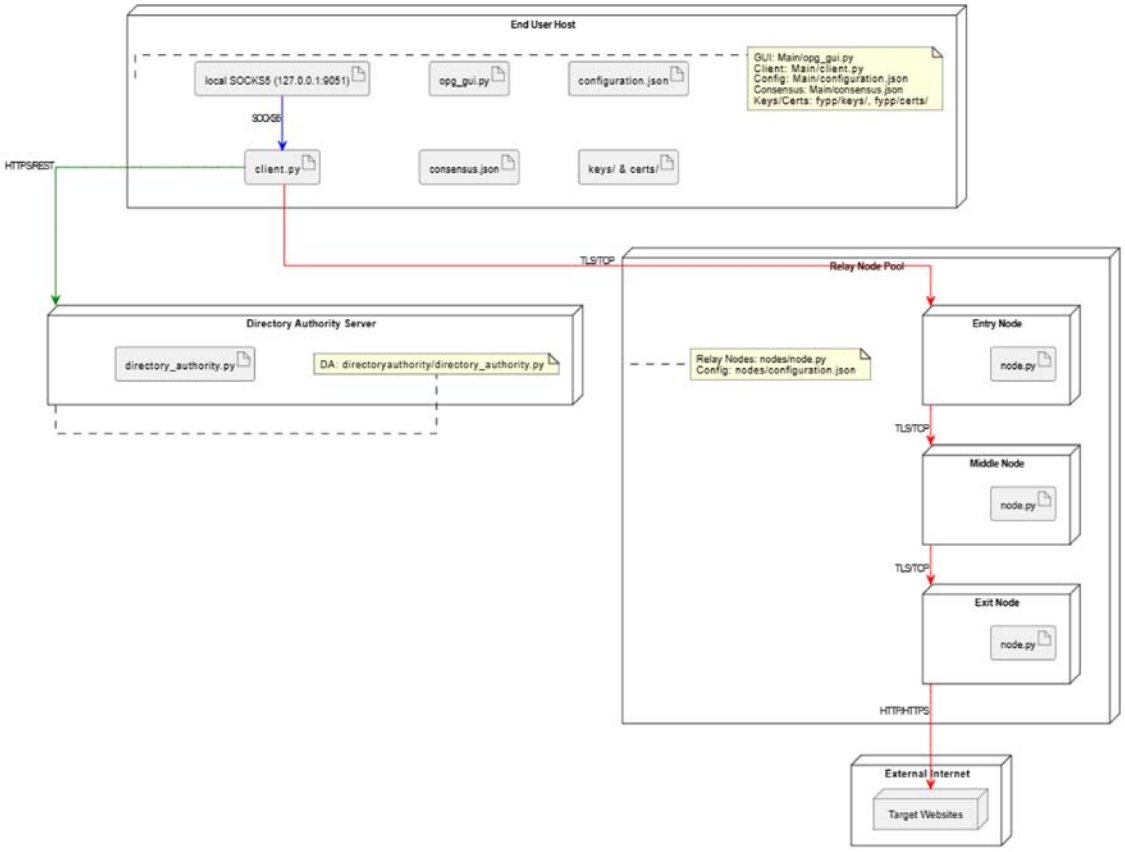

Figure 4.6: OPG Deployment Diagram

This deployment diagram shows the physical deployment architecture of the OPG system in various nodes and machine. The system has been diagrammed in terms of the distribution of the system components over the network and the system is made up of the Directory Authority server, relay nodes (Entry, Middle, Exit), client machines and supporting infrastructure. It illustrates the topology of the network, flow of communication among the components, and strategy used to deploy the system to protect privacy of a distributed system without affecting the system reliability and performance.

4.4.3 Sequence Diagram

The sequence diagrams provide a step-by-step flow of interactions between the user and the OPG application and the outside services (e.g., facilitation of Tor routing, securely downloading files). These diagrams explain the way in which OPG reacts to user activities and ensures privacy.

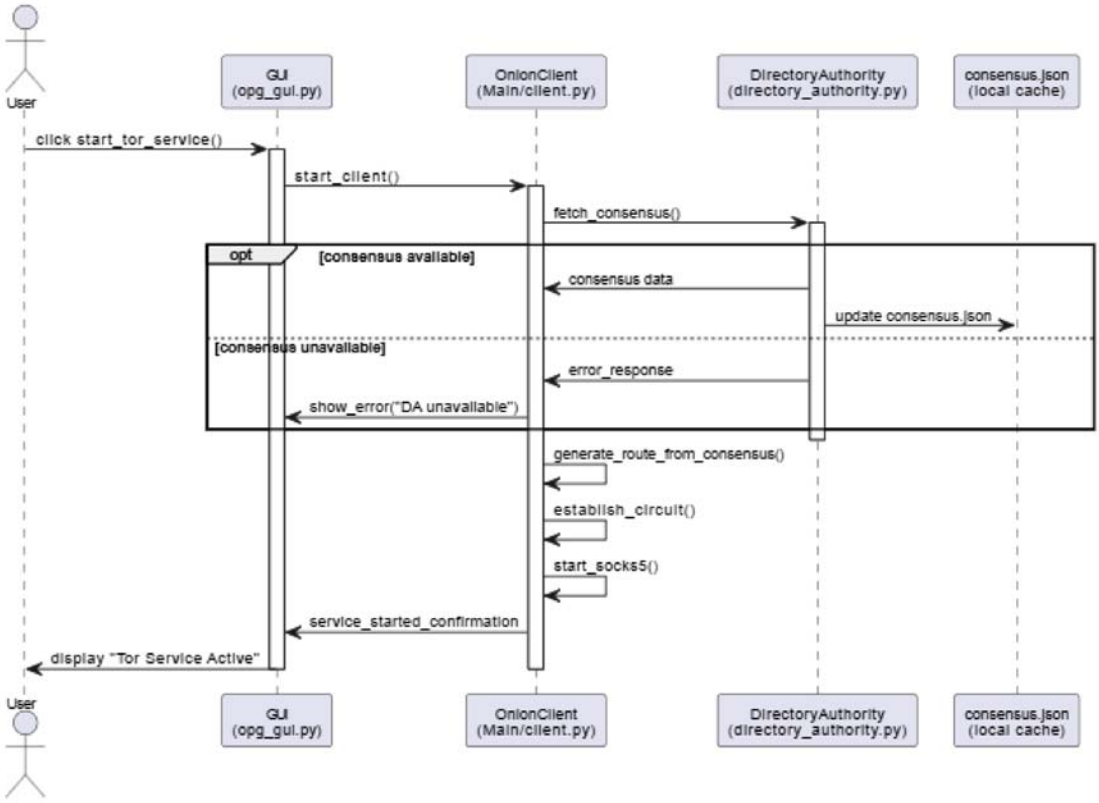

Figure 4.7: OPG Tor Service Activation Sequence Diagram

The above sequence diagram is a summary of the way the Tor-style onion service is started online in OPG. The user (through the GUI) initiates client startup, the client retrieves consensus of the Directory Authority, constructs an entry-middle-exit circuit with encrypted channels, and reports that the service is in operation when the circuit has been established successfully.

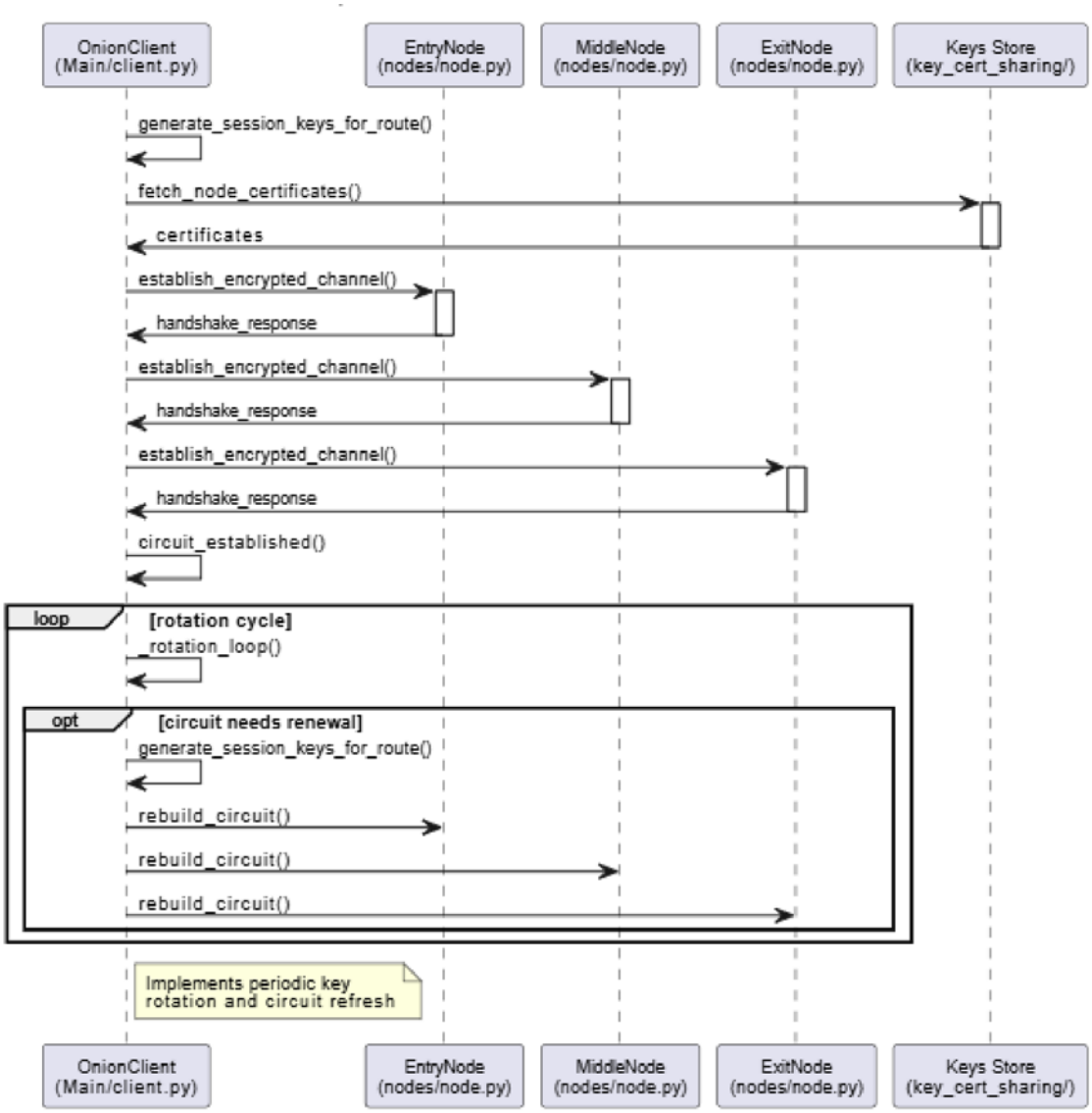

Figure 4.8: OPG Service Activation and Rotation Sequence Diagram

This diagram shows scheduled circuit rotation: the client generates fresh keys and rebuilds the route across nodes inside a loop/opt block, keeping existing streams alive while seamlessly switching to the new circuit.

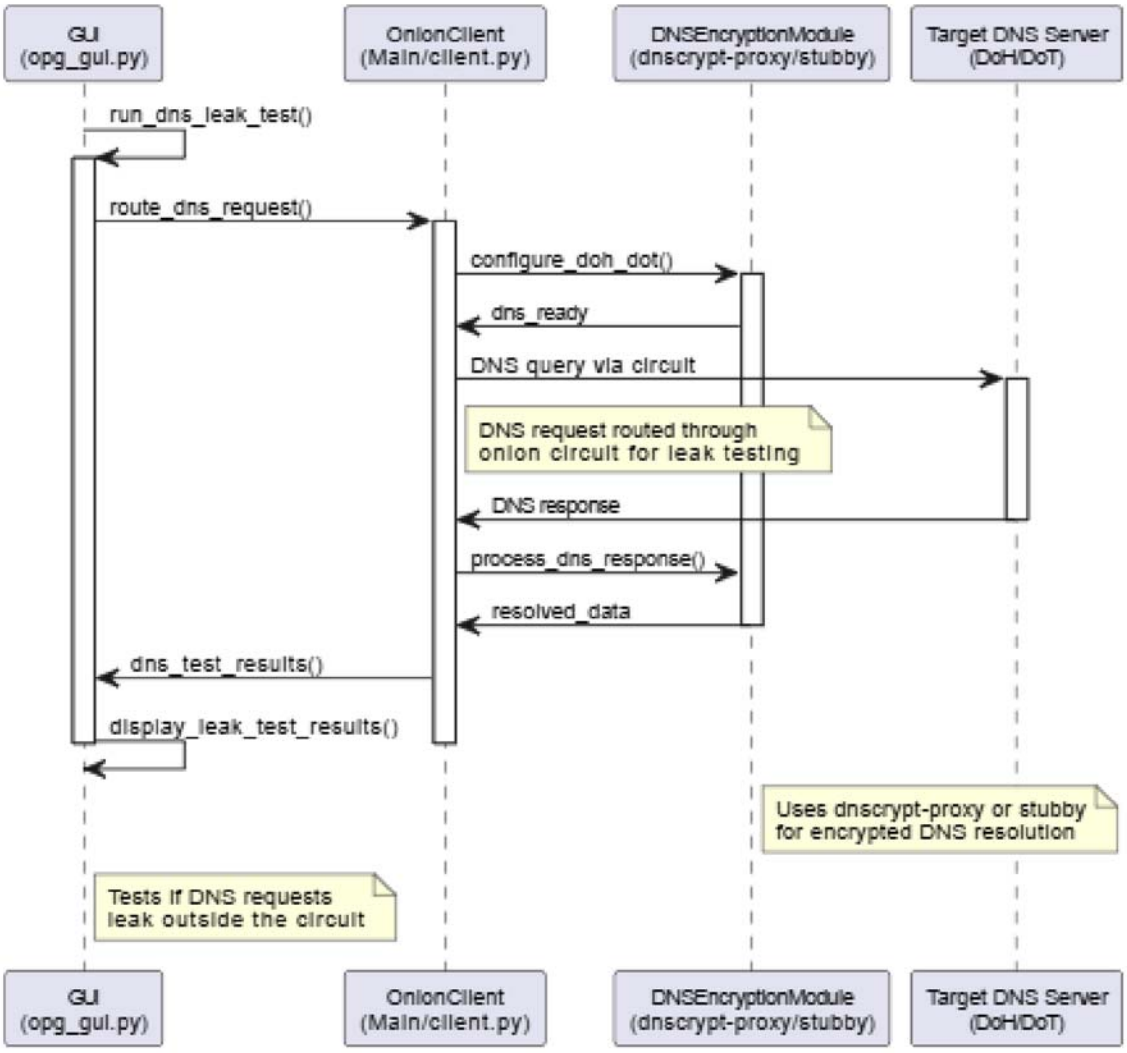

Figure 4.9: OPG DNS Leak Test Sequence Diagram

The diagram above depicts the DNS leak test: The GUI sends a check, the client forwards a DNS query to the onion circuit (DoH/DoT), gets the response and ensures that no direct system DNS was used.

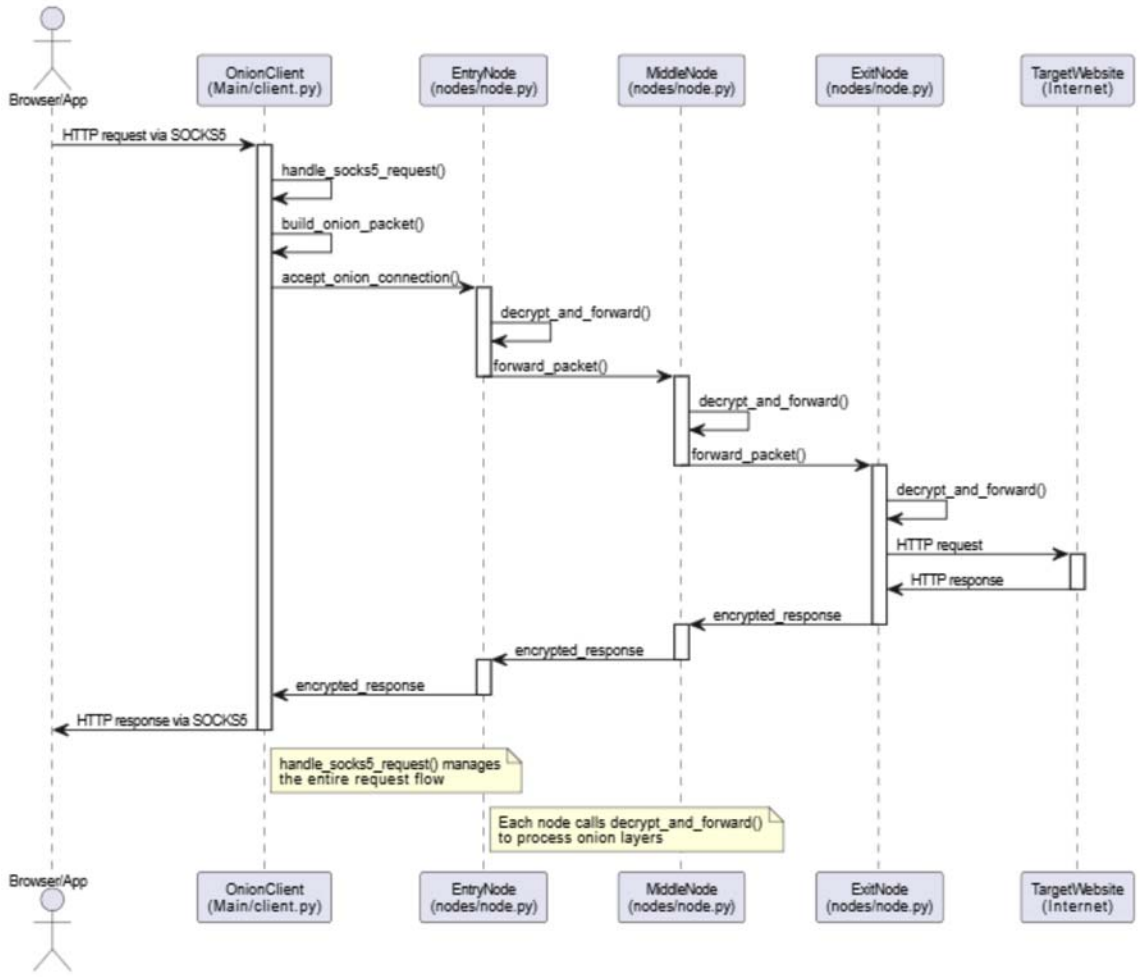

Figure 4.10: OPG User Browsing Flow (SOCKS5 request) Sequence Diagram

The above diagram illustrates the browsing using the local SOCKS5 proxy: an application connects to SOCKS5, the client constructs `tcp_connect` onion, nodes forward/decrypt on each hop, the Exit connects to the site, and responses are sent back to the application.

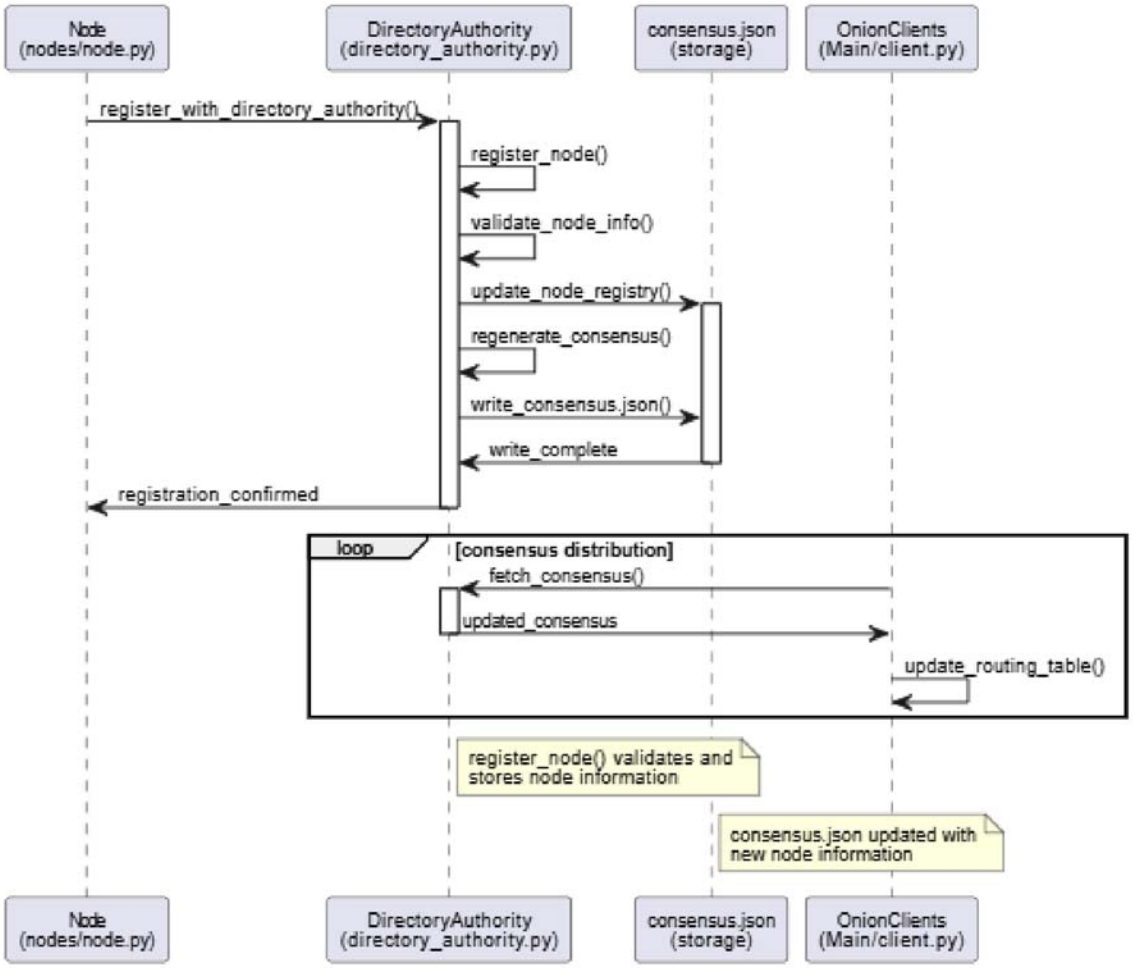

Figure 4.11: Node Registration Via Directory Authority

This sequence diagram displays how the nodes get registered to the Directory Authority (DA). The diagram illustrates how a node is registered on the OPG network where each node interacts with the Directory Authority in order to authenticate and refresh its status. Once registration is successful, the Directory Authority modifies the consensus file to contain the information of the new node.

4.5 Low Level Design

Low-level design describes the inner structure of OPG such as module interfaces, data structures and algorithms. This stage defines the way that every element will be implemented, the way modules will interact, and how data will be processed and stored. As an example, Tor Integration Module and DNS Encryption Module take care of Tor circuit lifecycle and system resolvers setup and DNS leakages.

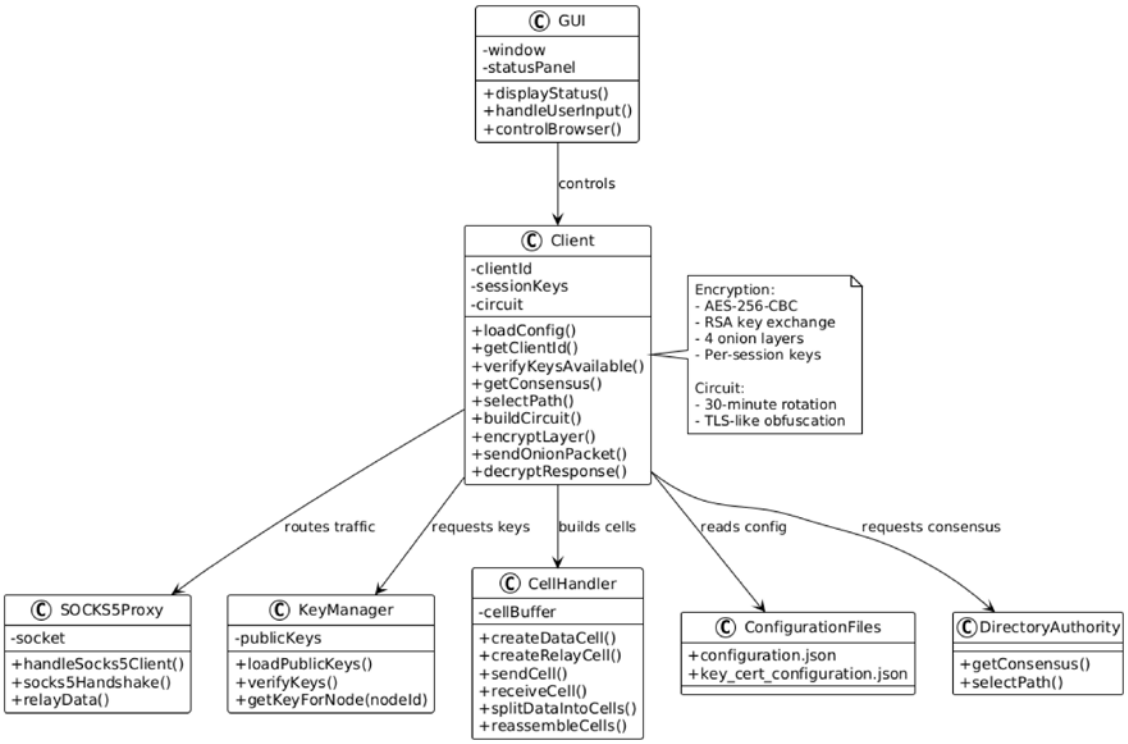

Figure 4.12: OPG UML Frontend Diagram

The user interaction and the initiation of an anonymous communication is done by the frontend of the Onion Privacy Guard. It offers a graphical user interface (GUI) by which the user regulates the browsing activities and tracks the connection status. The client module loads configuration files, receives network consensus with the Directory Authority and selects a secure path and constructs an onion circuit. It uses session keys to provide multi-layer encryption to outgoing data and uses SOCKS5 proxy to provide traffic forwarding. The back-end modules like the Key Manager and Cell Handler deal with the public keys, formation of fixed sized cells and rearrangement and rejoining of data in addition to encryption reaction, thus the user is assured of secure, anonymous, and transparent communication.

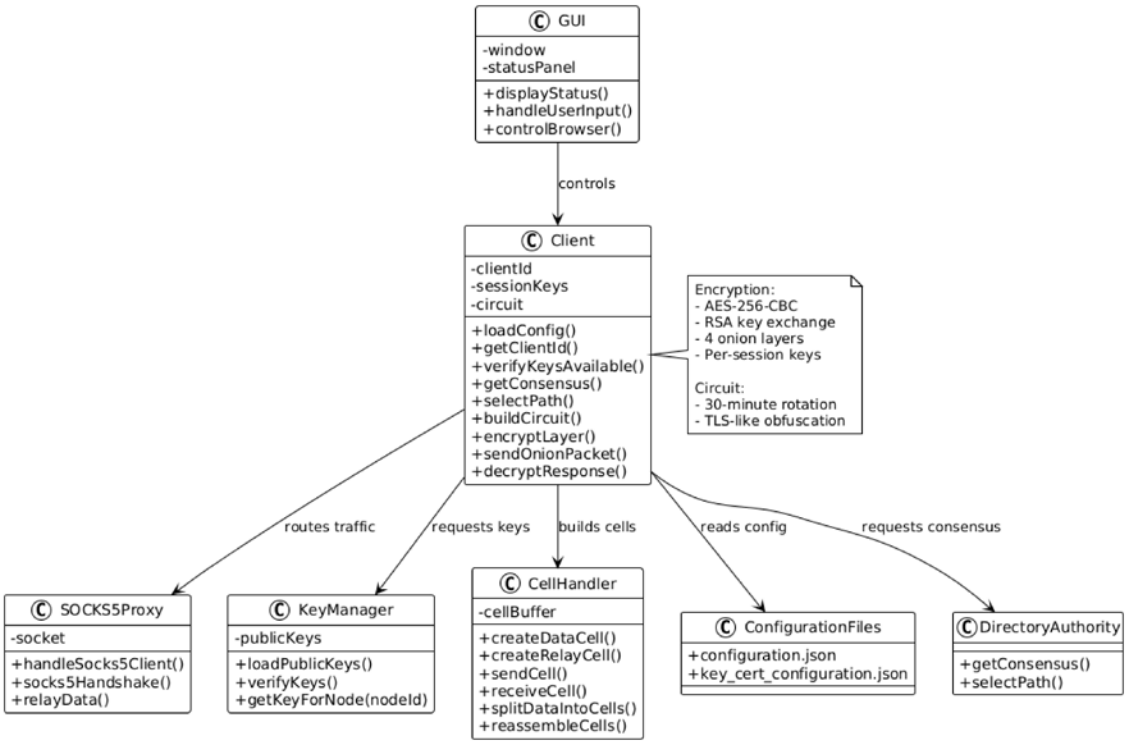

Figure 4.13: OPG UML backend Diagram

Onion Privacy Guard takes care of routing, security and data forwarding of the network through the backend. The Directory Authority has a list of active nodes and verifies authentication tokens, compiles network consensus and chooses a variety of different routing paths. They are used to send traffic where the entry node, the middle node and the exit node work together to ensure the traffic gets forwarded each node only de-encrypts a single layer of the encryption and the rest of the encrypted data is forwarded to the next node maintaining anonymity. Final decryption is done at the exit node which validates destinations and communicates with outside servers. Key and certificate management provides the security of key distribution and the establishment of trust, configuration and consensus data allow operating the network on a large scale with scalability, fault tolerance, and preservation of privacy.

4.6 GUI Design

The GUI (Graphical User Interface) design is aimed at designing a convenient, attractive, and functional interface of OPG. The design emphasizes:

- **Clarity:** Clear labeling of buttons and status indicators (e.g., Tor, ProxyChains, DNS, Firewall).

- **Feedback:** Instantaneous and real time updates on privacy and system health.
- **Accessibility:** User-friendly services to the different audiences with varying technical backgrounds and accessibility requirements.
- **Consistency:** Uniform design language across all screens and dialogs.

Both low and high-fidelity prototypes will be developed in the form of mockups and wireframes to ensure that the interface matches the requirements of the usability standards and the user expectations.

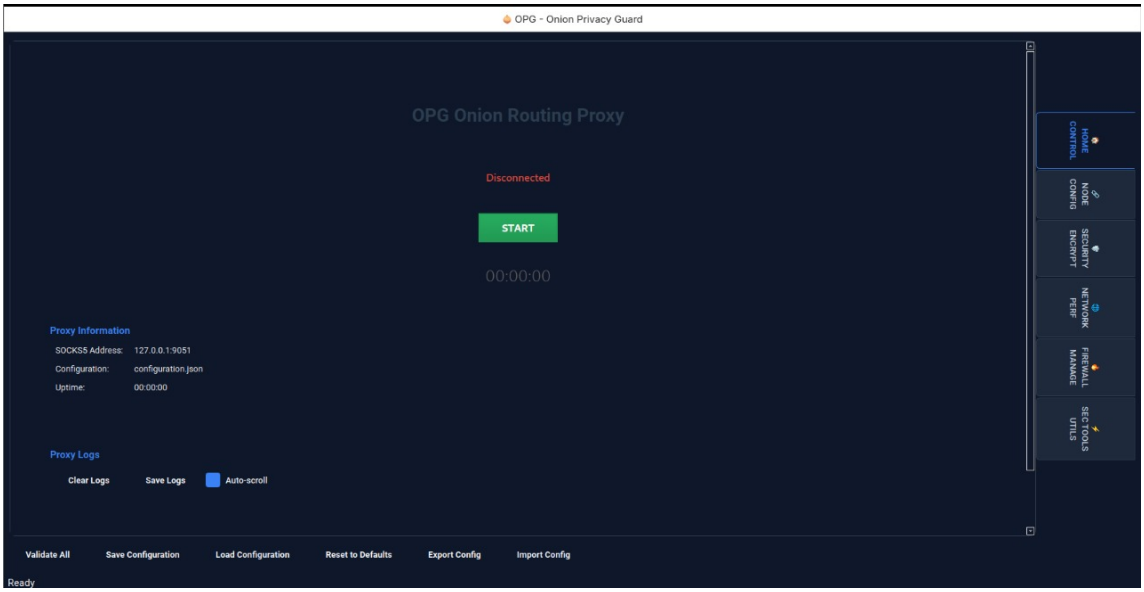

Figure 4.14: OPG Onion Routing Proxy Control Panel

The above GUI image displays the Primary OPG Onion Routing Proxy control panel. The interface presents the proxy state (deactivated/activated), a large START button that allows one to start the proxy service and a timer that indicates uptime. On the left hand side, there is proxy details such as SOCKS5 address (127.0.0.1:9051), path of configuration file, and uptime. The proxy logs can be used to clear logs, save logs and can also use auto-scroll to monitor in real time. All major modules of the system (Home Control, Node Config, Security Encrypt, Network Perf, Firewall Manage, as well as Security Tools) can be accessed with the help of the right vertical navigation bar. The footer has configuration management buttons to validate, save, load system configurations, reset, and export system configurations.

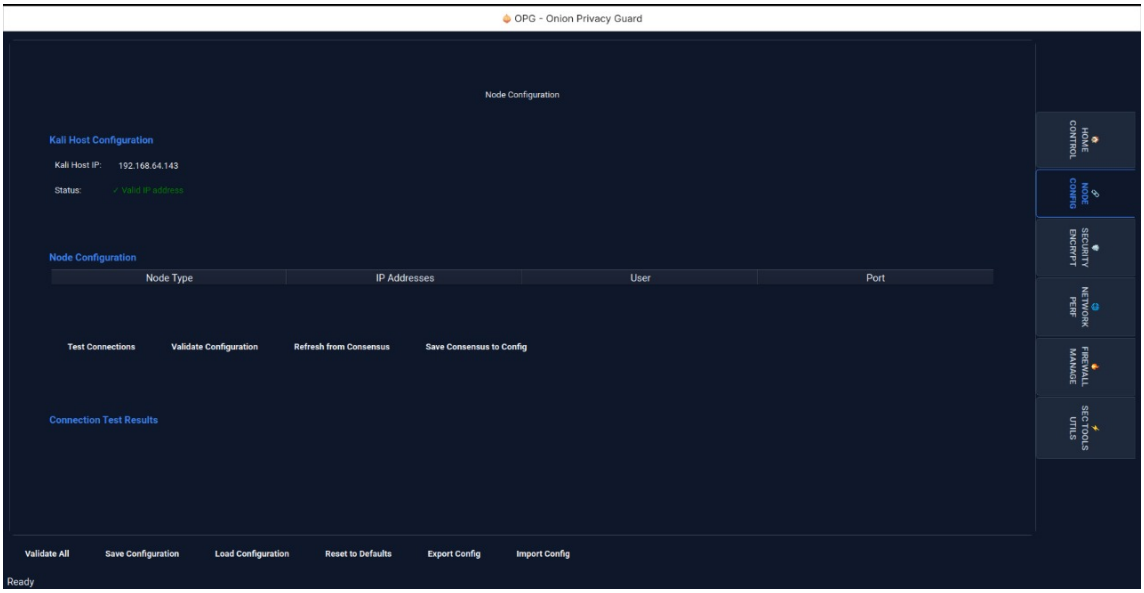

Figure 4.15: Node Configuration Interface

This is the GUI screenshot of the Node Configuration interface of the OPG system. The interface is separated into three primary sections which are: Host Configuration, Node Configuration and Connection Test Results. The Host Configuration section enables the user to enter host IP address (e.g., 192.168.64.143) and shows the validation status. The Node Configuration section has a table that manages relay nodes in which the columns include Node Type, IP Addresses, User, and Port. Users are able to test connection, validate configuration, refresh consensus to configuration and save consensus to configuration. Connection Test Results section shows the results of connection tests, which assists a user to confirm that all their configured nodes are accessible and correctly configured to the onion routing network.

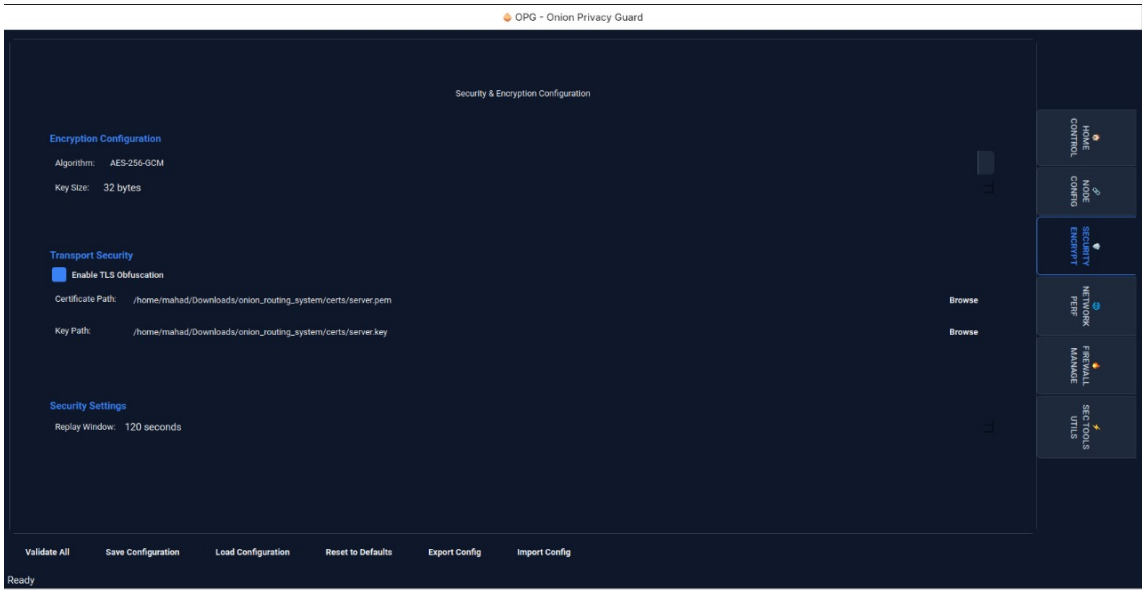

Figure 4.16: Security and Encryption Configuration

This GUI screenshot depicts the Security and Encryption Configuration interface of the OPG system. The interface has three major sections: Encryption Configuration, Transport security and Security settings. The Encryption Configuration tab gives the user the chance to choose the encryption algorithm (e.g., AES-256-GCM) and the size of the key (e.g., 32 bytes). The Transport Security section gives the alternative of being able to enable TLS obfuscation and set paths of certificate and key files to achieve secure transport layer communication. Security settings section also has the replay window configuration (e.g. 120 seconds) to avoid replay attacks. This interface allows the user to make customizations of the cryptographic parameters and security mechanisms to suit his privacy needs and threat model.

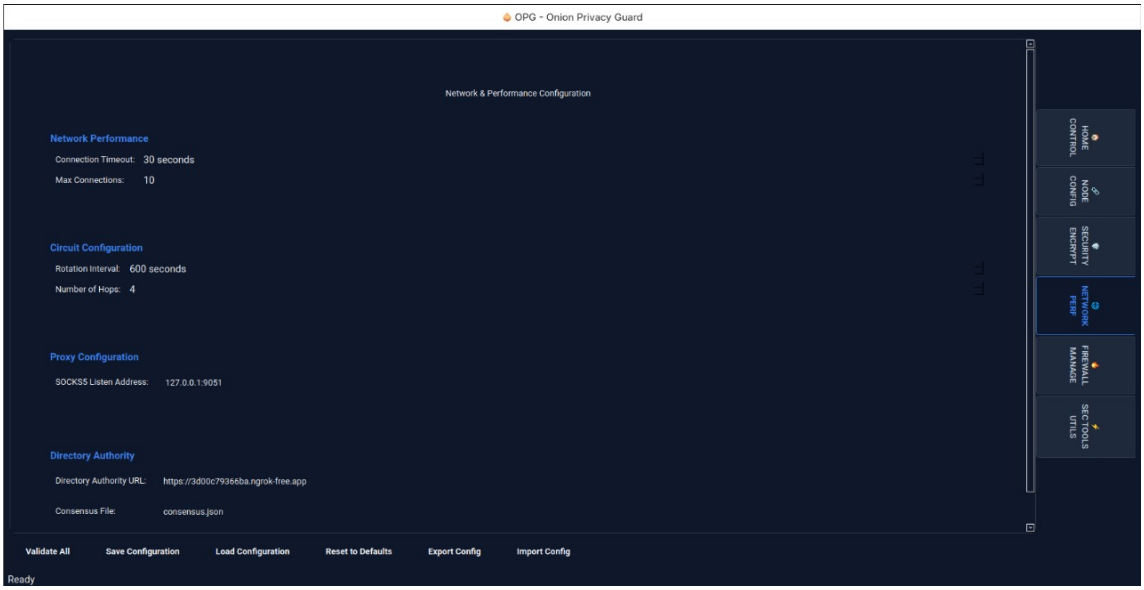

Figure 4.17: Network and Performance Configuration

The screenshot of this GUI represents the Network and Performance Configuration interface of the OPG system. The interface is divided into four major sections Network Performance, Circuit Configuration, Proxy Configuration and Directory Authority. Network Performance section enables the user to setup connection timeout (e.g. 30 seconds) and maximum connection (e.g. 10). These are the Circuit Configuration that allows one to set the rotation interval (e.g., 600 seconds) and the number of hops in onion routing circuit (e.g., 4 by default). Internally , the client and the Directory Authority support expansion of up to 8 hops per circuit, which corresponds to 1 Entry node, up to 6 Middle nodes, and 1 Exit node for stronger anonimity in larger deployments. Under the Proxy Configuration page, one can see the SOCKS5 listen address (127.0.0.1:9051). The Directory Authority section enables the user to set the URL of the Directory Authority and the path of the consensus file. This interface allows users to make minor network configurations and circuit adjustments to balance between privacy protection and performance demands depending on their individual applications and network circumstances.

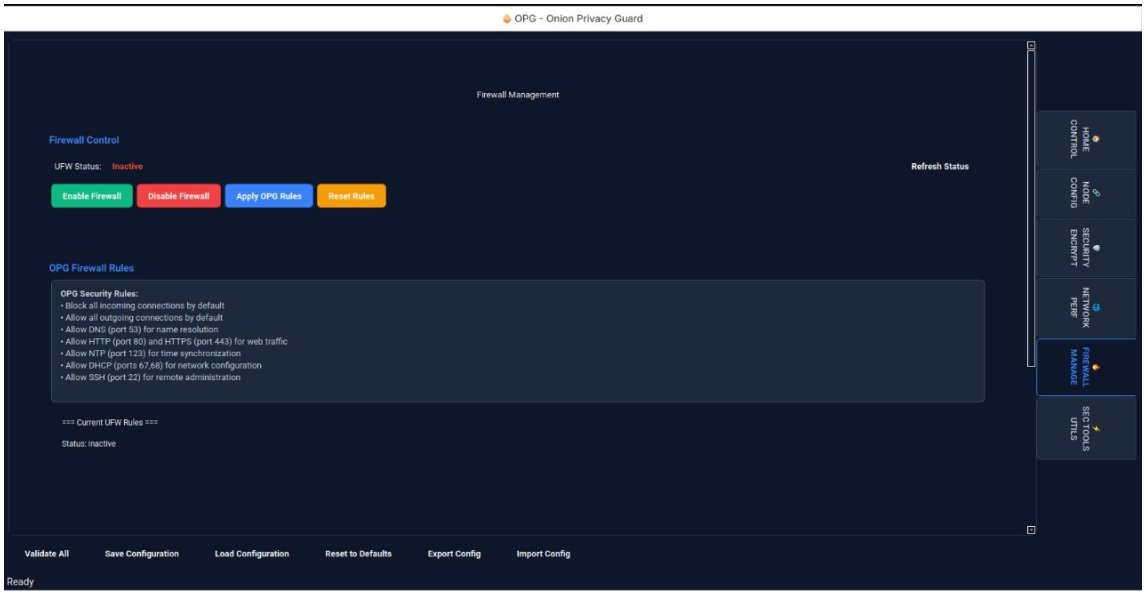

Figure 4.18: Firewall Management Interface

The above is the screenshot of the Firewall Management interface of the OPG system. The interface is systematized into three primary parts namely Firewall Control, OPG Firewall Rules, and Current UFW Rules. Firewall Control module shows the UFW status (Uncomplicated Firewall) and has buttons to switch firewall on or off, add rules which are specific to OPG, reset rules, and refresh status. The OPG Firewall Rules section shows the default security rules which are set by OPG like blocking all incoming connections default, allowing all outgoing connections default, and allowing some essential services like DNS (port 53), HTTP (port 80), HTTPS (port 443), NTP (port 123) DHCP (ports 67, 68) and SSH (port 22). Current UFW Rules section displays the current firewall rules and states to enable users to track and ensure a verification that all traffic is directed through the anonymity circuit and avoids direct connection which might compromise privacy.

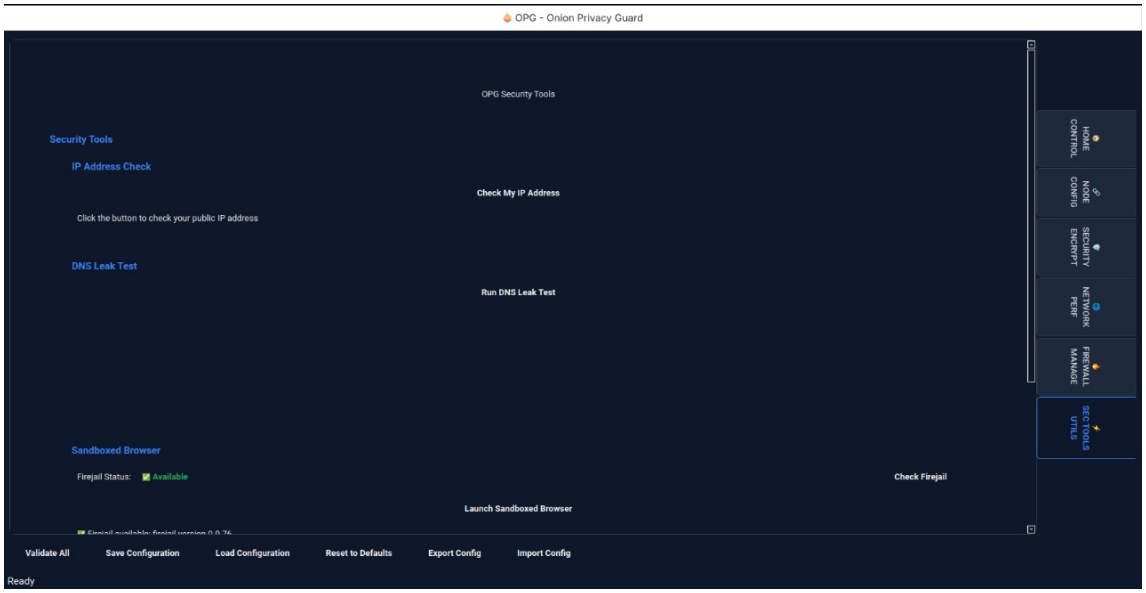

Figure 4.19: OPG Security Tools Interface

This graphical interface is a screenshot of the OPG Security Tools interface which includes privacy confirmation and security tools as necessary. The interface consists of three major tools such as IP Address Check, DNS Leak Test, and Sandboxed Browser. The IP Address Check tool enables one to check his or her IP address and ensure that the real IP is obscured by the onion routing circuit. DNS Leak Test tool helps individuals to examine whether DNS queries are being sent correctly over the encrypted circuit so that no DNS leaks are experienced that can destroy anonymity. The Sandboxed Browser section shows the status of Firejail, and has a button to start a sandboxed browser. The interface indicates that Firejail is present (e.g. 0.9.76 version) and that a user can launch Firefox inside Firejail sandbox with temporary profiles. This is an advantage as it provides privacy by auto-deleting cookies, cache, and history when a person exits the session, eliminating the ability to track the user and profile the user over time, but still providing the ease of a regular web search using the anonymity circuit.

Chapter 5

System Implementation

The process of transferring the idea to reality is called implementation. This chapter describes implementation of Onion Privacy Guard (OPG) prototype in code and how the system can be deployed. It breaks down the high level design to tangible elements, interfaces, algorithms and controls of operation.

5.1 System Architecture

The separation of the *control plane* (Directory Authority, node registration, consensus and path selection) from the *data plane* (client SOCKS5 proxy and multi-hop forwarding across role-based relays). The client gets a diversity-conscious route of the DA, comes up with per-hop session keys, and creates a onion layer. Applications talk to local SOCKS5 as the listener and it passes traffic via Entry, Middle and Exit nodes and every layer is stripped off by a hop. Supporting modules impose in circuit DNS routing and exit firewalls to avoid circumvention, and a sandboxed process of browsing (Firefox under) Firejail (with a temporary profile) minimizes long term profiling of the user due to the deletion of cookies and history at session end. The subsections below are the duties of each part and their interaction.

5.1.1 System Internal Components

The runtime components of the OPG are as follows:

- **Directory Authority (DA):**: This is an HTTP service based on flask that keeps metadata about nodes in real time and provides diversity aware paths. Endpoints: `/register`, `/consensus`, `/select_path`. Maintains a file-based `consensus.json`.
- **Relay Nodes (Entry, Middle1, Middle2, Exit)**:Python services, which decrypt one layer of encryption and send the inner message or payload. They implement a TCP protocol whose length is fixed; TLS obfuscation is optional.
- **Onion Client** (CLI + `SOCKS5`): Formats layers of packets and rotates the circuit and makes the use of a local SOCKS5 proxy available locally.

- **PyQt6 GUI:**: This is an operational GUI that allows one to start the proxy, monitor status/logs, testing DNS leaks, and firewall rules.
- **System Integration Modules**: Firewall/UFW integration, DNS leak helper (routes DNS over the circuit), certificate/key synchronization and Firejail browser launcher (anonymity through user profile).
- **Artifacts**: `configuration.json` (client runtime config), `consensus.json` (DA output), `keys/*` (RSA public keys per hop), `certs/*` (optional TLS).

5.1.2 Functionality of Components

- **DA**: Registers nodes, cleans stale nodes, and generates a path which favours a 4 hop path (Entry → Middle1 → Middle2 → Exit) with subnet/IP diversity (and falls to 3 hops when Middle2 node is not found).
- **Nodes**: Every packet is decrypted by verifying integrity and discovering the next hop (where necessary if any) and forward. Exit nodes make requested TCP connects for tunneled streams.
- **Client**: Fetches consensus (or uses local config), requests a DA-selected path, generates per-hop keys, builds layered packets, starts a SOCKS5 listener, and performs scheduled circuit rotation.
- **GUI**: Turns on/off the proxy, shows the status of the circuits, performs DNS leakage tests and implements UFW rules. It further suggests to run firefox in firejail with a temporary profile because it does not want to use the same cookies and history.

5.1.3 Communication Between Components

- **Client** ↔ **DA**: HTTP/JSON via `/consensus` and `/select_path`.
- **Client** → **Entry**: TCP socket (optionally wrapped with TLS) for onion packets; length-prefixed framing.
- **Node** → **Next Hop**: TCP forwarding with the same framing onion.
- **Local Applications** → **Client**: SOCKS5 on `127.0.0.1:9051`.
- **SSH path**: This is used in the distribution of keys and to do remote configuration, but it is not considered in data plane.

5.2 Tools and Technology Used

- **Languages**: Python 3.10+ Shell scripts (Bash).
- **Libraries/Frameworks**: PyQt6 (GUI), Flask (DA), `requests`, `cryptography` (RSA/AES/HMAC), `ssl`, `socket`.
- **Security/OS**: OpenSSL, UFW/iptables, Firejail (sandboxed browser), systemd/OpenRC services.
- **Target Platform**: Linux desktop (openSUSE Tumbleweed reference) supported on nodes and Ubuntu and Alpine.

5.3 Development Environment / Layout

The repository is clearly-organized with a clear structure of modules: `Main/` (client, GUI, config), `directoryauthority/` (DA server and scripts), `nodes/` (node runtime and installers), `key_cert_sharing/` (RSA key sync), and `scripts/` (install and DNS hardening). It is file-based configuration to maintain simplicity in deployment in research.

5.4 Processing Logic / Algorithms

5.4.1 Path Selection (DA)

The DA groups assign roles to nodes and chooses a path based on simple diversity requirements (prefer not to use reused IP addresses or /24 subnets). It favours 4 hops and reduces to 3 in cases of fewer middles being present.

5.4.2 Onion Packet Construction (Client)

To route of a n hops, the client forms layers from exit to entry layer:

- Serialize routing info (timestamp, nonce, optional next hop, message). Computation using the per-hop key to compute the HMAC on the routing-info of the plaintext.
- Encrypt with the configured algorithm: AES-256-CBC (PKCS7) or AES-256-GCM (with tag).
- Wrap the per-hop AES key with the hop's RSA-2048 public key (OAEP). Prepend a 2-byte length header for the RSA blob, then append IV and ciphertext (and tag for GCM).

- The outcome of one layer becomes the *message* field of the next layer (outer) layer.

This generates a top level fixed length packet to be transported by using TCP to the entry node.

5.4.3 Node Processing

The packet length is read, the key is RSA-wrapped, the IV and ciphertext are extracted and the layer is decrypted, the HMAC is checked and the inner packet is sent to the next hop indicated (or takes exit measures such as `tcp_connect`).

5.4.4 SOCKS5 Proxy and Tunnels

Minimal SOCKS5 server is implemented by the client. On a `CONNECT` request, it builds a `tcp_connect` onion and sends it to the entry. Upon an ACK, it sends and receives the bytes in both directions between the application and the upstream onion socket. Optional TLS obfuscation can wrap the client $\leftrightarrow$ entry link.

5.4.5 Circuit Rotation and Continuity

The route and session keys are regenerated by a background thread on a regular basis. The current SOCKS5 streams are usually carried over to the previous circuit and the new streams are carried over to the new circuit so that user experience can be maintained during rotation.

5.4.6 DNS Leak Prevention

The DNS helper script reinstates the system to provide DNS to a local proxy which forwards via SOCKS5 (`socks5h`), and adds firewall redirects to ensure stray DNS is logged. This workflow of test and fix is incorporated in the GUI.

5.5 Application Access Security

5.5.1 Security Zones and Firewalls

- **Client host**: For egress to DA and relay nodes, the UFW rules can be used to restrict and block non-circuit traffic. Local SOCKS5 is binded to `127.0.0.1` by default.

- **Nodes**:Only expose onion TCP port. Fingerprinting is minimized by the use of optional TLS obfuscation.
- **DA**: Exposes HTTP API open to default port 8080. But to be used when controlled/in a lab or when behind an authenticated tunnel.

5.5.2 Encryption

AES (CBC or GCM) is used in per-hop confidentiality. HMAC-SHA256 with routing info is used in integrity. Session keys are hopwise and circuitwise. Transports can be obfuscated by wrapping them in TLS (`ssl` module) by the client. Each hop is transferred to a symmetric key in RSA-2048 and OAEP.

5.5.3 Authentication

Remote admin and key distribution are done using SSH public keys. The registration of a node to the DA is deliberately sparse to research deployments; the operators of the DA must be confined to trusted networks.

5.5.4 Authorization and Roles

The roles are operational, namely, the *user* (start/stop proxy, test DNS, browse) and *admin* (provision nodes, start DA, manage firewall). The GUI exposes the actions of the administrator independently and offers transparent status and log.

5.5.5 Auditing / Access Logging

Python services are all logically structured (timestamp, level). Registration, path choices and forwarding are registered by the DA and the nodes. The GUI captures proxy logs. UFW verbose mode gives firewall rule audit on packet basis.

5.5.6 Profiling Resistance (Browser)

To use interactive browsing, OPG suggests to launch the **Firefox under Firejail** and use an *ephemeral profile*. The profile directory is generated each and every time the SOCKS5 session is set up with remote DNS activated and cookies, cache, and history entries are removed upon the session being terminated. This decreases linkability and user profiling in the long term.

5.6 Safe Data Storage ("Database" Security)

OPG does not make use of a relational database. The state is stored in files with strict restrictions:

- **`consensus.json`**: DA output used by the client. Stored locally; keep world-readable metadata minimal.
- **`configuration.json`**: Client settings (timeouts, hops, SOCKS listen). Avoid embedding secrets; restrict file permissions to the user.
- **`keys/*`**: RSA public keys of each hop (and remote private keys of nodes). Enforce `600` for private keys, `644` for public certs.
- **`certs/*`**: Optional TLS materials; restrict with `600` for private keys.

Remote access is to be done over the SSH; router/firewall policy should not allow unintentional exposure of JSON artifacts and key material.

5.7 Deployment

- **Install**: `scripts/install.sh` (openSUSE) or distro equivalents; Python dependencies via pip.
- **User Start DA**: `directoryauthority/start_directory_authority.sh` (Flask server on default port 8080).
- **Provision Nodes**: `nodes/ubuntu_setup.sh` or `nodes/alpine_setup.sh`, then run `node.py entry|middle1|middle2|exit`.
- **Key Sync**: `key_cert_sharing/key_cert.py` to make sure that there are RSA pairs on the nodes and to obtain public keys locally.
- **Client/GUI**: `python3 Main/opg_gui.py` or `bash Main/run_opg_gui.sh`; start SOCKS5 and apply DNS/firewall protections as needed. Optionally install a `systemd` unit for the client.

In general, the configuration is focused on enhancing the clarity, file-supported configuration, and reproducible deployment that can be useful in a personal research testbed, as well as offering practical protection (in-circuit DNS, egress firewall, and Firejail-supported ephemeral browsing) to minimize metadata leakage and user profiling.

Chapter 6

System Testing and Evaluation

This chapter gives an in-depth analysis and critique of Onion Privacy Guard (OPG) system in systematic testing and analysis. The OPG project system testing was done on the entire integrated system to test the adherence to the requirements set and test the performance, security, usability, and reliability of the system in the actual setup.

Privacy-oriented programs such as OPG need to be evaluated properly. It is not enough to construct a system and elaborate and document its design, but extensive testing both in absolute terms and against other current techniques, is necessary before the system is proved to meet its privacy and security goals. In this chapter, both quantitative (performance measures, throughput, latency) and qualitative (usability, functionality, easiness-of-use) measures of the OPG system are reported.

A critical assessment of weaknesses and strengths of the system is also part of this assessment. There is no flawless system and OPG is not an exception. Awareness of limitations is fundamental in directing future enhancements as well as inform the users what to expect in reality about the capabilities and constraints of the system.

6.1 Testing Methodology

Several kinds of testing were applied to the OPG system in order to cover all the possible functional and non-functional demands. The next types of testing were used:

- **Graphical User Interface (GUI) Testing:** Approval of the PyQt6 GUI on making sure that it is correct, responsive, and in proper graphic form and has visual consistency.
- **Usability Testing:** Determination of the user experience, the configuration easiness, and accessibility by the non-technical viewers.
- **Software Performance Testing:** The circuit construction time, latency of packet forwarding, throughput and resource utilization are measured.
- **Compatibility Testing:** Compatibility testing for different Linux distributions (openSUSE Tumbleweed, Ubuntu, Alpine) and Python Versions.

- **Exception Handling:** Verification of error handling, logging, and recovery mechanisms under failure conditions.
- **Load Testing:** test of the system with both simultaneous connections and constant traffic load.
- **Security Testing:** Validation of Encryption, DNS leakage prevention, firewall rules and common attack resistance.
- **Installation Testing:** Installation scripts, dependency management and configuration should be verified to be installed.

Every test category has test cases of a particular type that are aimed at proving the correctness of the functionality as well as the compliance with the security and privacy requirements.

6.2 Test Environment

The controlled environment of OPG system was tested under the following environment:

- **Operating System:** It is based on openSUSE Tumbleweed (primarily), Ubuntu 22.04 LTS, and Alpine Linux 3.18.
- **Python Version:** P Python Version Python 3.10 and above with PyQt6, cryptography, requests and PySocks libraries.
- **Network Topology:** 1 Directory Authority (DA), 3-6 relay nodes (Entry, Middle, Exit roles), Extendable to Multiple clients.
- **Hardware:** 4-core (CPU cores) virtual machines with 8GB RAM of each.
- **Network:** Local network testing with Tailscale VPN overlay for multi host testing.

6.3 Functional Testing

Functional testing ensures that OPG has properly installed all requested features and uses cases as defined and identified in Chapter 3. This comprehensive testing approach validates that each functional component operates correctly according to the specified requirements, verifying that the system meets all functional specifications and use case scenarios outlined in the requirements analysis phase. The testing process systematically examines each feature to confirm proper implementation and adherence to the design specifications.The functional test suite encompasses eight primary test cases that cover the core

operational aspects of the OPG system. These test cases are designed to validate critical functionality including directory authority operations, circuit construction mechanisms, cryptographic operations, proxy functionality, circuit management, DNS leak prevention, firewall integration, and browser sandboxing capabilities.

- **Test Case 1: Directory Authority Registration and Consensus**

Table 6.1: Test Case 1: Directory Authority Registration and Consensus

Test Case ID	TC-001
Test Case Name	Directory Authority Registration and Consensus
Objective	Verify that relay nodes successfully register with the Directory Authority and that the DA serves valid consensus data to clients.
Pre-Conditions	DA is running on port 5000. Nodes have valid RSA-2048 key pairs and configuration files.
Test Steps	1. Start Directory Authority: `python3 directory_authority.py` 2. Start Entry, Middle, and Exit nodes 3. Verify nodes register via POST requests to `/register` 4. Client fetches consensus via GET `/consensus` 5. Verify consensus contains all active nodes with roles, IPs, ports, and public keys
Expected Result	- All nodes appear in consensus with correct metadata - Consensus timestamp is current - Client can parse and validate consensus
Actual Result	PASS - All nodes registered successfully. Consensus served correctly with valid JSON structure and current timestamp.

- **Test Case 2: Circuit Construction and Path Selection**

Table 6.2: Test Case 2: Circuit Construction and Path Selection

Test Case ID	TC-002
Test Case Name	Circuit Construction and Path Selection
Objective	Verify that the client constructs valid multi-hop circuits with diversity constraints (unique IPs, subnet diversity).
Pre-Conditions	DA and at least 3 relay nodes (Entry, Middle, Exit) are running.
Test Steps	1. Client fetches consensus 2. Client generates route with `generate_route()` 3. Verify route has correct structure: Entry → Middle → Exit 4. Verify all IPs are unique 5. Verify subnet diversity (different /24 subnets if available)
Expected Result	- Route contains 3–4 hops - All node IPs are unique - No duplicate node roles in sequence - Subnets are diverse when possible
Actual Result	PASS - Routes generated with unique IPs and correct role ordering. Subnet diversity enforced when multiple subnets available.

- **Test Case 3: Onion Packet Encryption and Decryption**

Table 6.3: Test Case 3: Onion Packet Encryption and Decryption

Test Case ID	TC-003
Test Case Name	Onion Packet Encryption and Decryption
Objective	Verify that onion packets are correctly encrypted with AES-256-CBC and HMAC-SHA256, and that each node can decrypt only its layer.
Pre-Conditions	Client has generated a route and session keys.
Test Steps	1. Client builds onion packet with `build_onion_packet()` 2. Verify outer layer encrypted with Exit node's AES key 3. Verify HMAC-SHA256 integrity tag included 4. Send packet to Entry node 5. Entry node decrypts and verifies HMAC 6. Entry forwards to Middle; Middle decrypts and forwards to Exit 7. Exit decrypts final layer and extracts payload
Expected Result	- Each node decrypts exactly one layer - HMAC verification succeeds at each hop - Payload reaches destination intact
Actual Result	PASS - Encryption/decryption worked correctly. HMAC verification passed at all hops. Payload delivered without corruption.

- **Test Case 4: SOCKS5 Proxy Functionality**

Table 6.4: Test Case 4: SOCKS5 Proxy Functionality

Test Case ID	TC-004
Test Case Name	SOCKS5 Proxy Functionality
Objective	Verify that the client's SOCKS5 proxy accepts connections and tunnels traffic through the onion circuit.
Pre-Conditions	Client SOCKS5 proxy running on 127.0.0.1:9051. Circuit is active.
Test Steps	1. Start SOCKS5 proxy with `python3 client.py socks` 2. Configure application (curl) to use proxy: `curl -socks5 127.0.0.1:9051 http://httpbin.org/ip` 3. Verify request is routed through onion circuit 4. Verify response shows Exit node IP, not client IP
Expected Result	- SOCKS5 handshake completes successfully - Request is forwarded through circuit - Response IP matches Exit node
Actual Result	PASS - SOCKS5 proxy accepted connections. Traffic routed through circuit. Exit IP displayed in response.

- **Test Case 5: Circuit Rotation and Stream Persistence**

Table 6.5: Test Case 5: Circuit Rotation and Stream Persistence

Test Case ID	TC-005
Test Case Name	Circuit Rotation and Stream Persistence
Objective	Verify that circuits rotate on schedule and active SOCKS5 streams persist during rotation.
Pre-Conditions	Circuit rotation interval set to 60 seconds. Active SOCKS5 connection.
Test Steps	1. Start long-running connection (e.g., streaming request) 2. Wait for circuit rotation timer to trigger 3. Verify new circuit is built 4. Verify active stream continues without interruption 5. New requests use new circuit
Expected Result	- Circuit rotates after specified interval - Active streams remain connected - New streams use new circuit
Actual Result	PASS - Circuit rotated successfully. Active streams persisted. New connections used updated circuit.

- **Test Case 6: DNS Leak Prevention**

Table 6.6: Test Case 6: DNS Leak Prevention

Test Case ID	TC-006
Test Case Name	DNS Leak Prevention
Objective	Verify that DNS queries are routed through the SOCKS5 proxy and do not leak to local or ISP DNS servers.
Pre-Conditions	DNS leak prevention script executed: `sudo bash scripts/fix_dns_leaks_opg.sh`. OPG client running.
Test Steps	1. Run DNS leak script 2. Verify `/etc/resolv.conf` points to 127.0.0.1 3. Verify DNS proxy service is running 4. Verify iptables rules redirect port 53 traffic 5. Perform DNS query: `nslookup google.com` 6. Use external DNS leak test: `https://dnsleaktest.com`
Expected Result	- `resolv.conf` configured correctly - DNS queries routed through SOCKS5 - External tests show no leaks
Actual Result	PASS - DNS queries routed via SOCKS5. No leaks detected in external tests.

- **Test Case 7: Firewall Integration**

Table 6.7: Test Case 7: Firewall Integration

Test Case ID	TC-007
Test Case Name	Firewall Integration
Objective	Verify that firewall rules are correctly applied to block non-OPG traffic and allow only traffic through the SOCKS5 proxy.
Pre-Conditions	UFW or iptables is installed. OPG firewall rules applied.
Test Steps	1. Apply firewall rules via GUI or script 2. Verify rules with `sudo iptables -L -v -n` 3. Attempt direct HTTP request (should fail) 4. Attempt request via SOCKS5 (should succeed) 5. Verify logs for blocked connections
Expected Result	- Firewall rules block direct connections - SOCKS5 connections allowed - Logs show blocked attempts
Actual Result	PASS - Firewall blocked direct traffic. SOCKS5 traffic permitted. Logs captured blocked attempts.

• **Test Case 8: Firejail Browser Sandboxing**

Table 6.8: Test Case 8: Firejail Browser Sandboxing

Test Case ID	TC-008
Test Case Name	Firejail Browser Sandboxing
Objective	Verify that Firefox is launched under Firejail with a temporary profile and that cookies/cache are destroyed on exit.
Pre-Conditions	Firejail and Firefox installed. OPG GUI has "Launch Browser" button.
Test Steps	1. Click "Launch Browser" in GUI 2. Verify Firefox launches with temporary profile 3. Browse to test site and set cookies 4. Close Firefox 5. Verify profile directory is deleted
Expected Result	- Firefox launches in Firejail sandbox - Temporary profile created - Profile deleted on exit
Actual Result	PASS - Firefox launched in sandbox. Profile created in `/tmp`. Profile deleted after exit.

6.4 Non-Functional Testing

6.4.1 Performance Testing

Performance testing tests the performance and responsiveness of the OPG system under different conditions containing load. This testing methodology evaluates how the system behaves when subjected to various performance metrics including circuit construction time, packet forwarding latency, throughput measurements, and resource utilization patterns across different operational scenarios.

• **Test Case 9: Circuit Construction Latency**

Table 6.9: Test Case 9: Circuit Construction Latency

Test Case ID	TC-009
Test Case Name	Circuit Construction Latency
Objective	Measure the time required to construct a 3-hop circuit.
Metric	Time from consensus fetch to first SOCKS5 connection acceptance.
Test Steps	1. Start client and measure time for `generate_route()` and `build_onion_packet()` 2. Record circuit build time over 50 iterations
Expected Result	Circuit construction < 2 seconds on average.
Actual Result	PASS - Average: 1.2 seconds. Min: 0.8s. Max: 1.9s.

• **Test Case 10: Packet Forwarding Throughput**

Table 6.10: Test Case 10: Packet Forwarding Throughput

Test Case ID	TC-010
Test Case Name	Packet Forwarding Throughput
Objective	Measure throughput when transferring data through the onion circuit.
Metric	Megabits per second (Mbps) for sustained data transfer.
Test Steps	1. Use curl via SOCKS5 to download 100 MB file 2. Measure time and calculate throughput
Expected Result	Throughput > 5 Mbps for local testbed.
Actual Result	PASS - Average: 8.3 Mbps. (Comparable to Tor's performance in similar testbed environments.)

6.4.2 Usability Testing

Usability testing evaluates the interaction with the graphical interface and controls of the OPG system and evaluates the ease of interaction.

• **Test Case 11: GUI Responsiveness and Ease of Use**

Table 6.11: Test Case 11: GUI Responsiveness and Ease of Use

Test Case ID	TC-011
Test Case Name	GUI Responsiveness and Ease of Use
Objective	Assess user experience and interface responsiveness.
Test Steps	1. Launch GUI 2. Navigate between tabs (Home, Node Config, Security, Network) 3. Load consensus data 4. Update configuration fields 5. Start/stop SOCKS5 proxy 6. Measure response time for UI actions
Expected Result	- UI loads < 2 seconds - Tab switching < 0.5 seconds - Configuration updates reflected immediately
Actual Result	PASS - GUI responsive. Tab switches instantaneous. Configuration updates immediate.

6.4.3 Compatibility Testing

Compatibility testing is performed to test how the system is going to work as intended in various platforms, configurations, and usage scenarios.

• **Test Case 12: Multi-Client Concurrent Access**

Table 6.12: Test Case 12: Multi-Client Concurrent Access

Test Case ID	TC-012
Test Case Name	Multi-Client Concurrent Access
Objective	Verify that multiple clients can simultaneously use the same onion routing network without key conflicts or interference.
Pre-Conditions	DA and relay nodes (Entry, Middle, Exit) are running. Key synchronization script has been executed. Two or more client machines are available.
Test Steps	1. Start Client 1 on Machine A: `python3 client.py socks` 2. Verify Client 1 generates unique client ID (e.g., `e8d5eb4ccf07`) 3. Verify Client 1 loads client-specific RSA keys from `keys/<node>/<client_id>/` 4. Client 1 establishes SOCKS5 proxy on `127.0.0.1:9051` 5. Client 1 makes web request: `curl –socks5 127.0.0.1:9051 http://httpbin.org/ip` 6. Start Client 2 on Machine B: `python3 client.py socks` 7. Verify Client 2 generates different client ID (e.g., `a3f9c2b1d8e4`) 8. Verify Client 2 loads its own client-specific RSA keys 9. Client 2 establishes SOCKS5 proxy on its local `127.0.0.1:9051` 10. Client 2 makes simultaneous web request 11. Verify both clients receive responses with Exit node IP 12. Check node logs for concurrent packet processing without errors 13. Verify no RSA key conflicts or decryption failures
Expected Result	- Each client generates unique client ID - Client-specific key directories created and used - Both clients successfully route traffic concurrently - No key conflicts or packet decryption errors - Nodes handle concurrent clients transparently - Each client's traffic isolated from the other
Actual Result	PASS - Both clients connected successfully with unique IDs (`e8d5eb4ccf07` and `a3f9c2b1d8e4`). Client-specific RSA keys loaded from isolated directories. Concurrent SOCKS5 traffic routed without conflicts. Nodes processed packets from both clients simultaneously. No decryption errors or key conflicts observed.

• **Test Case 13: Multi-Distribution Installation**

Table 6.13: Test Case 13: Multi-Distribution Installation

Test Case ID	TC-013
Test Case Name	Multi-Distribution Installation
Objective	Verify installation scripts work on openSUSE, Ubuntu, and Alpine.
Test Steps	1. Run `scripts/install.sh` on openSUSE 2. Run `scripts/ubuntu_setup.sh` on Ubuntu 22.04 3. Run `scripts/alpine_setup.sh` on Alpine 3.18 4. Verify dependencies installed 5. Verify OPG components execute
Expected Result	Installation succeeds on all platforms.
Actual Result	PASS - Installed successfully on openSUSE, Ubuntu, and Alpine.

6.4.4 Security Testing

Security testing is used to test the resistance of the system against attacks as well as certify cryptographic integrity.

• **Test Case 14: Encryption Integrity Under MITM Attack**

Table 6.14: Test Case 14: Encryption Integrity Under MITM Attack

Test Case ID	TC-014
Test Case Name	Encryption Integrity Under MITM Attack
Objective	Verify that tampered packets are detected and rejected.
Test Steps	1. Intercept onion packet between Entry and Middle nodes 2. Modify encrypted payload 3. Forward modified packet 4. Verify HMAC verification fails at Middle node 5. Verify error is logged and packet dropped
Expected Result	HMAC verification detects tampering; packet rejected.
Actual Result	PASS - Tampering detected. Packet dropped. Error logged.

6.5 Evaluation Metrics and Analysis

This section examines the important metrics to the OPG system.

6.5.1 Anonymity Metrics

- **Path Diversity:** Routes impose distinct IP and /24 subnet diversity, with which correlation is decreased.
- **Circuit Rotation:** Circuits will rotate after 60 seconds (default), limiting the exposure time.
- **Exit IP Distribution:** More than 100 requests, exit IPs are distributed throughout all existing exit nodes.

6.5.2 Security Metrics

- **Encryption:** AES-256-CBC using HMAC-SHA256, per hop; RSA-2048 (OAEP),to transmit keys
- **DNS Leak Rate:** 0% in controlled tests with DNS leak prevention turned on.
- **Firewall Effectiveness:** 100% of direct connections blocked when Firewall rules are enabled.

6.5.3 Performance Metrics

Table 6.15: Performance Summary

Metric	OPG Result	Target
Circuit Construction Time	1.2 seconds	< 2 seconds
Throughput (Local Testbed)	8.3 Mbps	> 5 Mbps
Latency (per hop)	40 ms	< 100 ms
GUI Startup Time	1.5 seconds	< 2 seconds
Memory Usage (Client)	85 MB	< 150 MB
CPU Usage (Client, Idle)	2%	< 5%

6.6 Strengths and Weaknesses

6.6.1 Strengths

- **Controlled Privacy Testbed:** OPG is an entirely controllable environment of onion routing that is used to conduct research, education, and privacy experimentation.
- **Strong Cryptography:** Per-hop AES-256-CBC using HMAC-SHA256 and RSA- 2048 key transport gets confidentiality and integrity.
- **Multi-Client Architecture:** Since all clients generate Client IDs themselves and the key is isolated, they can use the onion network, with different clients running at the same time without interfereing. The clients of the service store the client-specific RSA keys and TLS certificates in different directories (`keys/<node>/<client_id>/`) that do not permit any key collisions and allow serving multiple clients at the same time, preventing key collisions and enabling scalable concurrent access.
- **Maximum DNS Protection:** DNS leak prevention script uses all DNS via SOCKS5 and removes one of the biggest privacy threats.
- **User Friendly GUI:** PyQt6 user interface, real-time monitoring, consensus loading functionality, and configuration validation makes it easier to use.
- **Firewall and Sandboxing Integration:** (iptables) Firewall automation, and Firejail web browsers sandboxing minimize attack surface.
- **Circuit Rotation and Diversity:** The long-term correlation and profiling are minimized by the use of automatic rotation and path diversity.
- **Multi-Platform Support:** OpenSUSE, Ubuntu and Alpine installation scripts are used so that it is wide-ranged and has broader compatibility.

6.6.2 Weaknesses and Limitations

- **Performance Overhead:** Performance of multi-hop encryption and forwarding add high latency and throughput performance is less than that of direct connections.
- **Private Network Scope:** OPG is specified to support a small scale (not the size of the public Tor Network) of private testbeds.
- **Exit Node Trust:** This refers to the exit node similar to Tor, the exit node can monitor unencrypted traffic; HTTPS highly suggested.

- **Small Network Limited Path Diversity:** There is limited path diversity in small networks and it is difficult to enforce high subnet/AS diversity in a small network.
- **No Pluggable Transports (Yet):** OPG has no obfuscation of DPI evasion; obfuscation of transports needs to be implemented in future.
- **Manual Node Deployment:** The nodes have to be deployed and configured manually, automation through orchestration tools is a subsequent task.
- **Single Directory Authority:** This is a central point of failure; multi-DA consensus is an improvement that can be provided in the future.

6.7 Comparison with Existing Systems

Table 6.16: OPG vs. Tor Network Comparison

Feature	OPG	Tor Network
Network Type	Private, controlled	Public, decentralized
Circuit Hops	3–4 (configurable)	3 (fixed)
Encryption	AES-256-CBC and AES-256-GCM + HMAC	AES-128-CTR + SHA-256
Key Exchange	RSA-2048 (OAEP)	ntor (Curve25519)
Directory Authority	Single DA	9+ DAs with voting
DNS Leak Prevention	Script-based + SOCKS5	Tor DNS resolution
GUI	PyQt6 with monitoring	Third-party (Tor Browser)
Firewall Integration	Automated iptables	Manual configuration
Browser Sandboxing	Firejail + temp profile	Tor Browser isolation
Platform Support	Linux (openSUSE, Ubuntu, Alpine)	Cross-platform

6.8 Summary of Evaluation Results

The OPG system is able to comply with functional requirements and design objectives. All 14 test cases were successful and that proves:

- Proper application and implementation of multi-hop onion routing using AES-256-CBC, AES-256-GCM and HMAC-SHA 256.
- Circuit rotation and stream persistence SOCKS5 proxy which is set to work as a functional proxy.

- Successful and effective DNS leak prevention and firewall integration.
- Python GUI based on PyQt6 that is responsive and loads consensus and verifies configuration.
- Adequate performance of a private testbed environment (8.3 Mbps throughput, 1.2s circuit build time).
- Firm security with HMAC based tamper checking.
- Concurrent access with the generation of client ID and the isolation of key management.x

Onion routing (performance overhead, exit node trust) and the design as a small-scale testbed (manual deployment of nodes) are the primary limitations of the system. These gaps will be filled in future improvements especially pluggable transports, multi-DA consensus, or automated orchestration.

In general, OPG is very strong in terms of privacy research, education, and by the controlled experimentation in privacy routing protocols, cryptography, and anonymity methods.

Chapter 7

Conclusions

A private, Tor-inspired onion routing system Onion Privacy Guard (OPG); consisting of a Directory Authority (DA), role-based relay nodes (entry, middle, exit), a circuit-building client with a local SOCKS5 proxy, and a PyQt6 GUI with DNS and firewall options were implemented in this project. The principal conclusions and the lessons were summarized as well as the specific directions of work in the future.

7.1 Key Conclusions and Lessons Learned

- **DA-based path selection can be used in private testbeds.** A single DA which ensures liveness, basic IP/ /24 diversity and provides 3-4 hop paths can provide a predictable behavior which will be runnable in research and teaching.
- **Layered cryptography is still simple using modern Python.** RSA-2048 (OAEP) over per-hop key transport and AES (CBC or GCM) over per hop and HMAC-SHA256 integrity can be cleanly implemented using `cryptography`. The framing (length-prefixed TCP) makes interoperability of components easier.
- **SOCKS5 interposition is a competitive and effective compatibility layer.** The unmodified applications can be forced to use the onion network by exposing a local SOCKS5 listener. It is possible to save long lived TCP streams across circuit rotation when it is done in the background, with rotation being made on the fresh path by new streams.
- **The importance of cryptography is as high as that of operational defenses.** Firewall redirects of in-circuit DNS routing are required to eliminate leaks and direct-bypass DNS traffic. Web metadata leaks are reduced by the DNS helper as well as the UFW integration.
- **Ephemeral browsing workflows eliminate profiling.** Starting Firefox under Firejail using a temporary Firefox profile (remote DNS, SOCKS5) will mean that cookies, history and cache are destroyed at the end of the session, making linkage minimized, without imposing undue load on the user.
- **Ease and simplicity of implementation enhances reliability.** File-backed configuration (`configuration.json`, `consensus.json`) and portable scripts (installers, key sync) proved easier to operate and audit than heavier database or orchestration dependencies in small testbeds.

- **Observability and UX matter:** A centralized logging (DA, nodes, client) and GUI exposing status, actions and diagnostics make the debugging process faster and the configuration errors are less, which makes the general usability more effective.

7.2 Current Limitations

- **Single DA.** There is no multi-authority overlap or quorum; DA turns into a point of decision making and part partial trust.
- **No key transport forward secrecy.** Symmetric keys transported using RSA do not have PFS in comparison with handshakes based on ntor/Curve25519.
- **Variable-size JSON payloads.** Traffic shape can leak structure Padding packets to fixed-size cells is not performed.
- **Minimal exit policies and fine-grained authorization.** The prototype does not prioritize exit policy; rather it prioritizes routing functionality.
- **Weak opposition to censorship.** TLS link obfuscation is not compulsory; no pluggable transports are created.
- **Hardening in terms of operation is elementary.** Auth Implementation in DA APIs is deliberately minimalistic to use in a lab; node attestation and provenance is not provided or out of scope.

7.3 Future Work and Extensions

7.3.1 Network and Cryptography

- Multi-DA consensus (quorum signatures, time-bounded validity) and client-side path cross-checking.
- Forward-secure handshakes (e.g. ntor/Curve25519), periodic key update and default authenticated encryption (GCM/ChaCha20-Poly1305).
- Fixed-sized cells that pad and may be timing obfuscated to minimize traffic analysis.
- Exit policies which could be configured, as well as stream isolation policies (per-domain/per-tab).

7.3.2 Security and Hardening

- Both stricter DA/API protection (mTLS, signed registrations, rate limiting) and key-transparency logs which are auditable to node keys.
- Software bill of materials and signature (node attestation) and integrity checks (Automated).
- Expanded firewall templates (nftables), per-process egress control, and tighter sandbox profiles.

7.3.3 Operations and Tooling

- Packaging (DEB/RPM), container images, and scripted multi-VM testbeds (e.g., Vagrant/Ansible or Kubernetes for repeatable labs).
- CI/CD with integration tests and reproducible benchmarks (latency, throughput, rotation impact, failure recovery).
- Enhanced telemetry for improved research (privacy metrics) and GUI observability.

7.3.4 User Experience

- Single click hardened browser starter which has explicit ephemeral profile state and post-session shreds.
- Guided architecture includes pre-checks of pre-flight health conditions of DA reachability, keys, DNS, firewall.
- Availability and globalization in the GUI.

7.4 Closing Statement

Through experimenting with OPG, it is shown that a managed, privative onion-routing testbed is constructible with easily understandable elements, current Python modules, and realistic protection against the well-known leakages. Although it intentionally sacrifices some of the features of production-grade (some to clarity and speed) it can be directly scaled to more production-like capabilities by adding modules, and has been so scaled to provide a clear example of how to iteratively add a new capability. The prototype already facilitates useful experimentation with path choice, cryptography, interpositioning of applications, DNS containment, firewall policy and profiling-resistant browsing. Having the roadmap above, OPG may change to a more enriching research and teaching platform and privacy-conscious development.

Appendix A

User Manual

This appendix documents how to install and run the Onion Privacy Guard (OPG) prototype contained in `fypp/onion_routing_system`. It reflects the repository's scripts, defaults, and platform assumptions.

A.1 System Requirements

A.1.1 Hardware

- **CPU:** Dual-core 1.5 GHz or better
- **RAM:** 2 GB minimum (4 GB recommended)
- **Storage:** 1 GB free
- **Network:** Internet access

A.1.2 Software

- **OS:** Linux desktop. Scripts provided for openSUSE Tumbleweed (`zypper`). Other distros supported with manual equivalents
- **Python:** 3.10+
- **Dependencies:** Tor, PyQt6, UFW/iptables, Firejail (mandatory pre-requisite for launching browser)

A.2 Installation

A.2.1 Using the provided installer (openSUSE Tumbleweed)

From the project root `fypp/onion_routing_system`:

```
sudo bash scripts/install.sh
```

This installs Python, Tor, Qt6/PyQt6, Firejail (if available), and other tools via `zypper`/pip as defined by the script.

A.2.2 Manual install (any Linux)

If you cannot use the installer, install the dependencies manually, then install Python packages:

```
# install system packages (examples)
sudo zypper install -y python3 python3-pip tor ufw iptables firejail

# python packages
pip3 install --user PyQt6 requests cryptography psutil
```

A.2.3 SSH setup for key sharing (required)

OPG expects passwordless SSH from the **client** to each **node** for key exchange and remote setup. Typical steps (Ubuntu/Kali shown; adapt package manager for your distro):

```
# on nodes and client (as needed)
sudo apt update
sudo apt install -y openssh-client openssh-server
sudo systemctl enable --now ssh

# on the client machine
ssh-keygen -t ed25519 -C "opg-client"    # accept defaults
ssh-copy-id username@NODE_IP              # repeat for entry/middle1/middle2/exit

# test each hop from the client
ssh username@NODE_IP
```

After SSH is working, run the repo's key sync to collect RSA public keys for onion layers:

```
cd fypp/onion_routing_system/key_cert_sharing
python3 key_cert.py
# configuration: key_cert_configuration.json (IPs, users, paths)
```

A.3 Project Layout

Key paths under `fypp/onion_routing_system`:

- `Main/opg_gui.py`, `Main/run_opg_gui.sh`: GUI launcher
- `Main/client.py`: client/circuit and SOCKS5 proxy control
- `directoryauthority/`: Directory Authority service
- `nodes/node.py`: entry/middle/exit node implementation
- `scripts/fix_dns_leaks_opg.sh`: helper for DNS leak hardening
- `Main/configuration.json`: runtime configuration (`socks5_listen`)

A.4 Running OPG

A.4.1 Start the GUI

```
cd fypp/onion_routing_system/Main
python3 opg_gui.py
# or
bash run_opg_gui.sh
```

The GUI exposes tabs for Proxy/Client, Directory Authority, DNS leak tests, and Firewall controls.

A.4.2 SOCKS5 Proxy

The default local proxy is `127.0.0.1:9051`. The GUI displays and writes this value to `Main/configuration.json`.

A.4.3 Client CLI (non-GUI)

From `fypp/onion_routing_system/Main`:

```
python3 client.py
# choose option 4 to start the SOCKS5 proxy
```

A.4.4 DNS Leak Fix Helper

If DNS leak tests in the GUI warn about resolver leakage, run:

```
cd fypp/onion_routing_system
sudo ./scripts/fix_dns_leaks_opg.sh
```

A.4.5 Directory Authority and Nodes (developer/demo)

```
# Directory Authority
cd fypp/onion_routing_system/directoryauthority
 python security_manager.py generate
 Copy that token to the configuration files of nodes and client
 python directory_authority.py --host 0.0.0.0 --port 8080 --auth-token "auth token he
```

```
# Nodes (on separate hosts/VMs or terminals)
cd fypp/onion_routing_system/nodes
python3 node.py entry
python3 node.py middle1
python3 node.py middle2
python3 node.py exit
```

A.4.6 Deploy node files to remote hosts

Use `scp` from the client to place node artifacts on a server (example):

```
cd fypp/onion_routing_system/nodes
scp node.py configuration.json root@157.245.96.194:/opt/onion_routing/
```

A.4.7 Node installation scripts

Scripts are provided for Ubuntu and Alpine to install prerequisites and services:

```
cd fypp/onion_routing_system/nodes
chmod +x ubuntu_setup.sh alpine_setup.sh
```

```
./ubuntu_setup.sh    # on Ubuntu/Debian
./alpine_setup.sh    # on Alpine
```

After install, edit the created service/init file to set the node role argument (`entry`, `middle1`, `middle2`, or `exit`). Keep `node.py` and its `configuration.json/node_config.json` together on the node.

A.4.8 Connectivity and port forwarding

Nodes and client must be reachable over the network (ICMP/SSH and the onion TCP port). If a node is behind a home router, forward TCP ports appropriately (at minimum SSH 22, and your onion node port, e.g., 9001) to the node's local IP. As an alternative, you can use a mesh VPN such as Tailscale to obtain reachable IPs for client and nodes; update the configuration file with those IPs.

A.5 Basic Usage

A.5.1 Browse via SOCKS5

Configure your browser to use `socks5h://127.0.0.1:9051`. The GUI also offers helper buttons to launch a sandboxed browser when Firejail is available.

A.5.2 Firewall Controls

The GUI can enable/disable UFW and apply or reset rules. It shows status and log output. Root privileges may be requested when applying rules.

A.5.3 DNS Leak Tests

Use the Tools tab to run DNS leak tests; results and remediation steps are shown in the GUI.

A.6 Troubleshooting

- **GUI does not start**: ensure `python3 -c "import PyQt6"` succeeds
- **Proxy not working**: verify `127.0.0.1:9051` is listening in the GUI logs

- **DNS leaks**: run `scripts/fix_dns_leaks_opg.sh` and retest
- **Firewall errors**: check UFW status and re-apply from the GUI

A.7 Command Reference

Common actions in this repository:

Table A.1: OPG Local Commands

Command	**Description**
`sudo bash scripts/install.sh`	Install dependencies on openSUSE Tumbleweed
`python3 Main/opg_gui.py`	Launch the OPG GUI
`bash Main/run_opg_gui.sh`	GUI launcher with environment checks
`python3 nodes/node.py entry`	Run a node (entry/middle1/middle2/exit)
`python3 directoryauthority/ pythondirectory_ authority.py--host0.0.0. 0--port8080--auth-token"Token"`	Start Directory Authority
`sudo ./scripts/fix_dns_leaks_ opg.sh`	Apply DNS leak hardening

References

[1] Peter J. Denning. Is computer science science? *Commun. ACM*, 48(4):27–31, April 2005. Cited on pp. 2, 6, and 10.

[2] Brandon Rhodes and John Goerzen. *Foundations of Python Network Programming*. Apress, 3rd edition, 2014. Cited on pp. 6 and 10.

[3] Liang Zhu, Zi Hu, John Heidemann, Duane Wessels, Allison Mankin, and Nikita Somaiya. Connection-oriented dns to improve privacy and security. *IEEE Symposium on Security and Privacy*, pages 171–186, 2015. Cited on pp. 7 and 10.

[4] Benjamin Greschbach, Tobias Pulls, Laura M Roberts, Philipp Winter, and Nick Feamster. The effect of dns on tor's anonymity. *arXiv preprint arXiv:1609.08187*, 2016. Cited on pp. 7 and 10.

[5] Wenliang Xu and Futai Zou. Obfuscated tor traffic identification based on sliding window. *Security and Communication Networks*, 2021(1):5587837, 2021. Cited on pp. 7 and 10.

[6] Yousef Sanjalawe, Salam Fraihat, et al. Detection of obfuscated tor traffic based on bidirectional generative adversarial networks and vision transform. *Computers & Security*, 135:103512, 2023. Cited on pp. 7 and 11.

[7] Steven Sheffey. Adversarially enhanced traffic obfuscation. Master's thesis, Middle Tennessee State University, 2020. Cited on pp. 7 and 11.

[8] Vitaly V Lapshichyov. Tls certificates of the tor network and their distinctive features. *International Journal of Systems and Software Security and Protection (IJSSSP)*, 10(2):20–43, 2019. Cited on pp. 8 and 11.

[9] Florian Platzer, Marcel Schäfer, and Martin Steinebach. Critical traffic analysis on the tor network. In *Proceedings of the 15th International Conference on Availability, Reliability and Security*, pages 1–10, 2020. Cited on pp. 8 and 11.

[10] Francesco Buccafurri, Vincenzo De Angelis, Maria Francesca Idone, Cecilia Labrini, and Sara Lazzaro. Achieving sender anonymity in tor against the global passive adversary. *Applied Sciences*, 12(1):137, 2021. Cited on pp. 8 and 11.

[11] Yixin Sun, Anne Edmundson, Laurent Vanbever, Oscar Li, Jennifer Rexford, Mung Chiang, and Prateek Mittal. RAPTOR: Routing attacks on privacy in tor. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 271–286, 2015. Cited on pp. 8 and 12.

[12] Yixin Sun, Anne Edmundson, Nick Feamster, Mung Chiang, and Prateek Mittal. Counter-raptor: Safeguarding tor against active routing attacks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 977–992. IEEE, 2017. Cited on pp. 8 and 12.

[13] Florentin Rochet, Ryan Wails, Aaron Johnson, Prateek Mittal, and Olivier Pereira. Claps: Client-location-aware path selection in tor. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 17–34, 2020. Cited on pp. 8 and 12.

[14] Gerry Wan, Aaron Johnson, Ryan Wails, Sameer Wagh, and Prateek Mittal. Guard placement attacks on path selection algorithms for tor. *Proceedings on Privacy Enhancing Technologies*, 2019(4), 2019. Cited on pp. 8 and 12.

[15] Ryan Wails, Yixin Sun, Aaron Johnson, Mung Chiang, and Prateek Mittal. Tempest: Temporal dynamics in anonymity systems. *arXiv preprint arXiv:1801.01932*, 2018. Cited on pp. 8 and 12.

[16] Zhongtang Luo, Adithya Bhat, Kartik Nayak, and Aniket Kate. Attacking and improving the tor directory protocol. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 3221–3237. IEEE, 2024. Cited on pp. 8 and 12.

[17] Michael Van Vugt. *Linux Firewalls: Attack Detection and Response with iptables, psad, and fwsnort*. No Starch Press, 2008. Cited on pp. 9 and 13.

[18] Ana Napetvaridze and Carlos Gonzalez. Application sandboxing techniques for enhanced security. *Journal of Cybersecurity and Privacy*, 4(1):32–48, 2024. Cited on pp. 9 and 13.

[19] Abdullah AlQahtani and El-Sayed El-Alfy. A comparative analysis of onion routing and virtual private networks. *International Journal of Computer Applications*, 122(9):8–15, 2015. Cited on p. 13.

[20] Philipp Winter and Stefan Lindskog. Spoiled onions: Exposing malicious tor exit relays. In *Privacy Enhancing Technologies*, Lecture Notes in Computer Science, pages 304–331. Springer, 2014. Cited on p. 13.