VPN or Vpwn? How Afraid Should You be of VPN Traffic Identification?

Tanmay Rajore, Jithin S, Arnav Gupta, Keshav Gambhir, Anindya Prithvi, Sambuddho Chakravarty
IIIT Delhi
{tanmay19118, jithins, arnav21236, keshav19249, anindya20024, sambuddho}@iiitd.ac.in

***Abstract*—Several governments are gradually choosing to monitor VPN traffic. In this paper, we explore how hard or easy it would be for large ISP-scale adversaries to identify and block VPN traffic. More specifically, we try to answer questions like should ordinary netizens fear such decisions or whether it is not as trivial to identify and block all sorts of VPNs.**

A recent study found that blocking and identifying OpenVPN endpoints is feasible for small ISPs. We explored detecting OpenVPN and alternatives like TLS, SSH, IPSec/IKEv2, Wireguard, and proprietary VPNs. Analyzing seven popular commercial and open-source VPN services, we identified patterns for detection. While OpenVPN is easily spotted, many alternatives resist identification, some using tactics like obscure TLS ClientHello SNI strings. We demonstrated evasion methods, including altering packet sizes, sending dummy traffic to confuse middleboxes, and obscuring plaintext strings. We also proposed a scalable mechanism for OpenVPN services to hide identifiable plaintext without affecting user or gateway scalability.

I. Introduction

Internet filtering and efforts to bypass them, are a proverbial "cat-and-mouse chase". In fact, a large body of work on (anti-)censorship focuses on this, *i.e.*, either designing difficult to block systems, or on how hard or easy it is for the adversary to spot them [21], [26], [34], [38], [46], [47], [51], [70], [74], [76]. Either way, most efforts seem to have ignored *virtual private networks* (VPNs) [7]. VPNs are a class of tunneling systems where the clients' packets (often the entire packet from L3-L5) is encrypted and then encapsulated inside a different packet before being relayed to a VPN *gateway*. The destination IP of the outer packets is the IP address of the gateway. The gateway decrypts these packets and decapsulates the inner packets, which are then routed to their appropriate destination. For an eavesdropper, the traffic appears fully encrypted, and he/she cannot determine the intended destination of the packets. This makes VPNs suitable as a censorship circumvention tool.

Governments across the globe are now moving to identifying and blocking VPN-based services [17], [20], [45], [66], [67]. *E.g.*, in 2022, the Indian government announced its decision to monitor VPN services directly. It also ordered various commercial VPN providers to keep track of users' profiles and data (pertinent to what sites are being accessed *etc.*) [45], [67]. Naturally, VPN providers that are neither headquartered and nor have gateways there, may choose not to comply. In such cases, the governments could choose to identify and block commercial VPN traffic, like how web is filtered.

In 2022, Xue *et al.* [78] demonstrated that OpenVPN, a VPN protocol, can be identified by adversaries, making it an unsafe option for circumventing censorship. Their analysis revealed ways in which an adversary can detect OpenVPN traffic and gateways. Besides OpenVPN many commercial VPNs also support additional tunelling protocols, including Wireguard, TLS, IPSec/IKEv2, and even proprietary ones [2], [5], [31], [60]. In this paper, we try to proactively answer the high-level question, *i.e. how afraid should ordinary netizens be of the government order to block/monitor VPN services?* More specifically, we analyze several popular VPN services (each using a different transport mechanism) to identify ones that could be easily spotted. As also described in Subsec. III-C, we chose the VPN services in terms of their popularity and the unique transport protocols/modes they used.

Most commercial VPN services employ a two-step process to allow users to create tunnels to VPN gateways. The first step involves the users' authentication, sharing configuration files and parameters with him/her that bear the IPs of specific (or several) VPN gateways, handshake signing keys, *etc.* Thereafter, the client uses these to negotiate a tunnel with the gateway. For our research, we analyzed both of these types of messages to spot patterns that make them prone to identification. We analyzed several commercial and non-commercial VPN services/applications. In particular, we chose NordVPN [2], ExpressVPN [31], ProtonVPN [60], VyprVPN [69], Perfect Privacy [3], Windscribe [5] and WARP [25]. Each of these was chosen based on their popularity and the fact that they cover most of the popular VPN transports, *i.e.* OpenVPN, Wireguard, TLS based, SSH *etc.*[1]

For the purposes of this paper we consider two different classes of VPN solutions. First are the open-source VPN implementations (*e.g.*, OpenVPN, WireGuard, IPSec) that are freely available software projects whose code, protocol specifications, and deployment configurations can be examined and modified by anyone (*i.e. vanilla*). In contrast, proprietary VPN protocols (*e.g.*, NordLynx, Lightway, Chameleon, Stealth mode, *etc.*) are closed-source or vendor-specific extensions that commercial providers bundle into their applications. Though they are built atop standard protocols (*e.g.*, WireGuard or OpenVPN), proprietary offerings may add custom hand-

[1] We ignored those that relied on insecure protocols like PPTP.

978-3-903176-74-4

shakes, obfuscation layers, or authentication backends that are not publicly documented. The following are some of our key observations:

- OpenVPN-based systems are by and large fingerprintable. The only exception is the pre-shared key mode that is difficult to fingerprint. However, this is difficult to scale as it requires a complex key management scheme.
- Besides OpenVPN based services there are several others like ProtonVPN Stealth, VyprVPN chameleon, and Perfect Privacy (OpenVPN+SSH), that rely on TLSv1.3 or encapsulate OpenVPN inside SSH packets, thereby making it difficult to fingerprint. Interestingly, some like IPSec/IKEv2 are quite prevalent for transporting packetized cellular voice and corporate VPNs. Blocking them could potentially lead to large collateral damage.
- Generally, commercial VPN services have two communication phases – user authentication and tunnel establishment/data transport. A determined adversary would observe both of these to look for identifiable patterns.
- The authentication protocol usually involves exchanging credentials over an HTTPS connection. The `ClientHello` SNI, bearing the VPN service provider's domain name, is often a giveaway. In several cases, this can be deliberately modified to appear as if that belongs to an innocuous domain. This makes them go undetected by middleboxes, without hindering the authentication step.
- In cases that rely on TLSv1.3, often the `ClientHello` is obfuscated, either by using an obscure string that makes it "fly under the radar" or by not using the SNI extension altogether. Either way, TLSv1.3 ensures that the certificates are encrypted, and thus does not reveal much about the domain or service being accessed.

To summarize, the main contributions of this work are:

1) A novel, detailed analysis of several popular open-source and commercial VPN services to assess the ease of identifying their traffic. Examining services using major transports, *e.g.*, OpenVPN, TLS, Wireguard, IPSec/IKEv2, SSH, and proprietary variants, we also tested their obfuscation features. OpenVPN and Wireguard proved identifiable, but many alternatives resisted detection effectively.
2) We propose censorship evasion techniques for VPNs with identifiable transports, including spoofing TLS `ClientHello` SNI fields during authentication and disrupting censorship middleboxes through packet size manipulation or spoofed packets.
3) We propose a scalable OpenVPN handshake scheme that creates a covert channel for standard message exchange using Diffie-Hellman key exchange. It integrates easily into existing setups and outperforms pre-shared key modes in multi-user environments.

II. Background and Related Work

A. Background

*1) **VPN filtering**:* In recent years, several governments have gradually decided to monitor VPN services, extending their influences to coerce VPN service hosting companies into tracking users' profiles and accessed sites. [17], [20], [66] This global shift has prompted researchers to investigate the feasibility of large-scale VPN fingerprinting, emphasizing easily identifiable signatures rather than focusing on specific national contexts. Previous studies have delved into the practicalities of extensive Internet filtering on a global scale, examining factors such as the number of network equipment requiring monitoring and identifying optimal paths for the installation of censorship middleboxes [40]. Building on this research, our study takes a broader perspective, assessing the viability of large-scale VPN identification by exploring universally recognizable signatures. This approach aims to address the evolving landscape of online privacy and surveillance concerns on an international level.

*2) **OpenVPN**:* OpenVPN, an open-source tunneling protocol, uses the kernel tunneling subsystem to create a virtual interface on the client. It encrypts and encapsulates entire packets as application-layer payloads, which the server decrypts, decapsulates, and forwards. Responses follow the same path. OpenVPN supports multiple modes with varying levels of confidentiality and privacy.

1) ***OpenVPN TLS mode:*** This mode of OpenVPN utilizes a handshake derived from TLS. This mode can work using both TCP and UDP. In both, the client and server run a slightly modified TLS handshake to derive a session key. While the mode provides forward secrecy, it is easy to fingerprint due to the presence of plaintext OpenVPN headers.
2) ***Pre-shared key mode:*** This mode uses a pre-shared keys to encrypt communication, hiding protocol details with pseudo-random payload bytes but lacking forward secrecy. Most commercial VPN services avoid it due to the need for complex, privacy-preserving key distribution, bootstrapping, and session management systems.

*3) **Wireguard**:* Recently, many VPN providers have adopted *WireGuard* [27] as a default alongside OpenVPN. Like OpenVPN, WireGuard uses a virtual network interface to encapsulate and encrypt application packets before transmission. It implements a lightweight 1-RTT handshake and key-exchange protocol derived from the Noise framework(specifically the *Noise_IKpsk2_25519_ChaChaPoly_BLAKE2s pattern*) [64] over UDP, providing mutual authentication, perfect forward secrecy, and identity hiding.

*4) **IPSec/ESP**:* IPSec is a Layer 3 tunneling protocol that encrypts and encapsulates IP packets within new IP headers. It operates mainly in two modes, *viz.*, *Authentication Header (AH)* and *Encapsulating Security Payload (ESP)*, with ESP providing both confidentiality and authentication. IPSec uses IKEv2, typically with Diffie-Hellman, for client authentication and key exchange. After negotiation, client packets are encrypted and sent to the VPN gateway, which decrypts and forwards them.

*5) **TLS and SSH based VPNs**:* Many VPNs, like OpenVPN, tunnel TCP/IP packets over TLS or SSH. TLS ensures confidentiality, authenticity, and privacy. Tools like Stunnel let clients establish TLS connections, through which encrypted packets are sent to the VPN server for decryption and routing.

Many services use SSH tunnels to transport packets, negotiating encryption and signing keys like TLS. Client traffic is encrypted, encapsulated, and sent to the server for decryption and forwarding. Compared to WireGuard and OpenVPN, TLS and SSH-based VPNs are harder to identify, as their traffic often resembles regular TLS or SSH flows.

6) ***Proprietary****:* Several VPN services employ proprietary protocols, such as Lightway from ExpressVPN, which operates in two modes: UDP and TCP. We noted that Lightway (UDP) transmits a handshake packet with identifiable plaintext, resembling TLSv1.2's ServerHello. In contrast, Lightway (TCP) establishes a TLS session akin to typical web requests, using a standard TLSv1.3 connection setup. This mode can be partially identifiable, as described ahead in Subsubsec. IV-B1.

B. Related Work

Extensive research has explored traffic fingerprinting of Tor and similar privacy tools [28], [42], [53], [58], [59], [71], including identifying their usage patterns and transport protocols. These techniques typically leverage features such as packet timing, size, burst patterns, and directionality (*i.e.*, characteristics that remain observable despite encryption) for identifying traffic associated with such anonymization tools.

Studies on web filtering have mainly focused on state-level censorship and blocking tools like Tor [43], [72], [77], with comparatively less attention given to how different VPN protocols are spotted and blocked.

Xue *et al.* [78] examined OpenVPN traffic DPI methods like Opcode-based fingerprinting and ACK-based fingerprinting (packet size), alongside active probing. They found that 'vanilla' OpenVPN is easily fingerprinted, making its flows and gateways detectable by ISP-level adversaries. However, many commercial VPNs also support other protocols like WireGuard, TLS, and IPSec.

A complementary study [61] introduced *VPNalyzer*, an automated tool for assessing VPN privacy leaks during tunnel failures. Such failures often lead to the traffic being rerouted directly (without tunnelling) via censoring ISPs, making it easy to identify and block them. While this highlights the dangers of tunnel fallback, it does not address how tunnels themselves can be identified or blocked in real time.

We analyze common VPN protocols, *viz.*, OpenVPN, obfuscated OpenVPN, Wireguard, TLS-based VPNs, and IPSec/IKE, alongside their commercial implementations, to determine their susceptibility to being spotted (and subsequently blocked) through network traffic inspection by powerful censors. Focusing on near-real-time detection without learning-based methods, we consider a realistic ISP-level adversary. Additionally, we also inspect how they are hosted (on which hosting infrastructure) and how user authentication protocol messages are also a vector for identification (*e.g.*, SNI headers in TLS-based VPNs). Our study of popular protocols (both "vanilla" and commercial)[2], improve VPN's understanding for users and researchers.

III. Preliminaries and Overview of Empirical Analysis

A. Problem Description

Given recent VPN filtering and monitoring orders, a key question arises: how easily can VPN use be detected? Research indicates that OpenVPN is simple to fingerprint and identify. Our study investigates the difficulty of identifying not only OpenVPN but also other common VPN protocols and tunneling methods employed by commercial VPNs.

B. Threat Model

We assume an ISP-level adversary operating under government directives, using with DPI-capable firewalls and middleboxes, with turn-key solutions such as DNS blocking, SNI filtering and service shutdowns. Observed in countries like India [62], [79], Syria [23], Iran [19], Italy [15], Pakistan [52], Afghanistan [16] and Thailand [37], these adversaries usually use off-the-shelf DPI devices that filter traffic based on IP addresses, port numbers, or SNI fields, and may also spot fixed byte patterns at specific offsets.

Several past efforts propose threat models involving *Global Passive Adversaries* [53] or state-level censors with sophisticated computing infrastructure deployed across entire networks (*e.g.* China) [29], [72], [78]. Such models are largely impractical for most nation-states, so we adopt a more realistic threat model. In addition, prior work [36] has noted that such highly capable adversaries would face significant challenges in buffering and processing traffic with low latency while ensuring accurate protocol identification.

Our threat model reflects real-world feasibility in terms of network visibility, economic and operational costs, and legal or political constraints. We assume the adversary has the following technical capabilities:

- An adversary can scan for fixed, known byte patterns at specific offsets within packets(e.g. TLS ClientHello SNI). It cannot extract and track consistent but modified byte sequences introduced by obfuscation (eg: XOR-patch), as administrators cannot "predict" what such transformations may look like, while configuring firewalls.
- Attempts to identify and block VPN flows, including filtering by IP or port and SNIs.
- May actively probe endpoints to detect VPN gateway behavior. (Middleboxes typically lack scanning capabilities; however, performing occasional scans to probe endpoints is feasible and deployable.)

C. Choice of VPN Services

We chose commercial VPN services based on two criteria: Firstly, we looked at the support of various VPN protocols like OpenVPN, Wireguard, IKEv2-IPSec, and TLS. In addition, we prioritized services offering proprietary protocols

[2]Excluding decentralized ones like *NYM* [57] or *Mysterium* [55]

that claim enhanced privacy and fingerprint resistance. Second, we considered popularity. There is no single definitive source of VPN user statistics. We collated information from various third-party websites [44], [65], [68], app store download statistics [18], [39], and official claims [32], [56], [60], [69]. Based on these, we selected 7 commercial VPN services which are *ExpressVPN*[EV], *NordVPN*[NV], *ProtonVPN*[PV], *VyprVPN*[VP], *Perfect Privacy*[PP], *Windscribe*[WS] and *WARP*[CF]. For brevity, we refer to them using the abbreviations throughout the paper. The details of the protocols supported by these providers, and their modes of operations, are presented in Tab. I

D. Overview of Our Approach

Most VPN services support multiple operating systems, with OS/device-specific libraries influencing ciphers, packet sizes, and transport modes. We analyzed VPNs on Windows (v10/v11), Linux (v5.19), MacOS (Mojave), and Android (v11/v12), assessing all available transport options.

1) Desktop analysis: We installed VPN applications on test machines and captured traffic via `pcap` at both the virtual tunnel interface and the physical network adapter (WiFi/Ethernet). This revealed how applications generate packets and encapsulate them for transmission. We focussed on two key operations: user authentication and gateway selection/connection.

2) Smartphone analysis: On Android 11 devices (*e.g.*, Motorola G54), we installed VPN apps from Google Play and connected them via WiFi to desktops acting as hotspots, routing traffic between the WiFi and the wired interface. Traffic was captured on the desktop as `pcaps` and authentication responses were examined by descrypting them with a MITM proxy [4]

3) VPN server-side (end-point) analysis: For analyzing the VPN traffic, we used the following three-node testbed:

1) A VPN gateway running either vanilla OpenVPN, vanilla Wireguard, vanilla IPSec or XOR-patched OpenVPN gateway, that we controlled. For evaluating commercial implementations, we relied on their respective end-point servers/gateways.
2) A desktop client connecting to a webserver through the gateway.
3) An Apache-hosted webserver itself, to which the client (2) connects.

By capturing traffic at both the client and the VPN endpoints (under our control), we analyzed differences in traffic patterns across protocols, their commercial implementations, and operational models, to evaluate detectability.

4) Emulating adversaries that inspect network traffic: To emualate an adversarial setting, we used a Dell R320 server (Intel Xeon E5-2400, 32GB RAM) running Pfsense [54] with default settings. Using this setup, we analyzed traffic patterns observed by a firewall when applying different circumvention methods (see Subsec. V-B).

IV. Empirical Network Identification

We outline experimental setups for analyzing VPN systems to detect unique network traffic patterns that might indicate a known VPN service. Our findings assess the feasibility of identifying VPN traffic. Typically, connecting to VPN gateways involves two steps (see Fig. 1): users authenticate with a server to exchange configuration files, secret keys (if needed), and the gateway's IP, then the client connects to the gateway using this data. Our experiments to identify VPN packets used manual methods, avoiding learning-based pattern recognition, similar to prior work [78]. We assume an adversary that doesn't use such techniques, with details provided in Sec. VI.

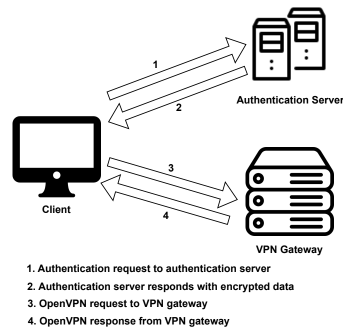

Fig. 1: The sequence of handshakes for commercial VPN providers. The first handshake is with the authentication server to validate the authentic user. The second handshake is with the VPN gateway endpoint for the tunnel establishment.

A. Analyzing OpenVPN Based Services

*1) **Analyzing vanilla OpenVPN**:* OpenVPN client and server rely on configuration files that bear various relevant pieces of information, *e.g.* VPN server's IP and port number, the usage mode (UDP or TCP), authentication mechanism (user-auth pass or user certificate based), hardcoded HMAC keys for control and data packet authentication and integrity protection.

Following the process highlighted above, we captured and analyzed the traffic. The following were the key observations:

1) In OpenVPN TLS mode, control packets and packet headers, including the data opcode, remain in plaintext, despite encrypted data, as noted by Xue *et al.*
2) In the case of pre-shared key mode, the header (bearing the opcodes) is also encrypted with the pre-shared keys and cannot be trivially spotted. Unlike the TLS mode, it does not provide forward secrecy and does not scale-up for multiple users (with each sharing a different pre-shared secret with the VPN server).
3) XOR-patch [14] can be used alongside vanilla OpenVPN. This scrambles the payload of OpenVPN by performing multiple XORs and buffer reversal [41]. However, the first opcode byte undergoes only one XOR operation. As a result, it remains distinct, making it consistent across packets in the same direction. Most VPN providers using XOR-patch rely on a fixed XOR string for all client connections, causing request and response packets to share the same first byte, which may signal the XOR use. In

private OpenVPN tunnels, refreshing the XOR string between sessions is possible. Still, the protocol doesn't allow mid-session updates, leaving it detectable by adversaries checking first-byte values in request-response pairs. This requires a custom firewall to inspect only the first byte of request packets post-TCP handshake, a feature most commercial firewalls don't offer.

Vanilla OpenVPN traffic is identifiable (both TCP and UDP). Pre-shared key resists it but doesn't scale.The XOR-patch attempts to obfuscate traffic by scrambling the payload; however, it leaves the first byte structurally unchanged (i.e., not fully scrambled) causing repeating byte values in request and response packets, detectable by a stateful adversary, looking for such repetitions. However, as noted in Subsec. V-D, most commercial firewalls do not provide configurable options to spot such sequences across packets in either direction (*i.e.*, client gateway or vice versa). Also, It is noteworthy that OpenVPN developers advise against its use because the code has not been audited [14].

*2) **Analyzing OpenVPN-based commercial services**:* We analyzed OpenVPN configurations in leading VPN providers and summarize our findings below.

1) ***NordVPN:*** NordVPN supports OpenVPN (UDP/TCP). Our analysis revealed that client configurations include static HMAC signature keys, while header encryption keys remain unused.
   Using dynamic analysis tools (*e.g.*, `lsof`, `strace`, `ptrace`, `ltrace`), we confirmed that upon login, the configuration updates with the VPN gateway's IP and port, indicating a two-step handshake: authentication via HTTPS (SNI: `nordapps1.com`, certificate CN: `*.nordcdn.com`), followed by OpenVPN negotiation. This makes NordVPN identifiable and blockable. On Android, authentication sometimes occurs over UDP, but the process remains similar.
   NordVPN employs the unofficial XOR patch with a globally shared, plaintext scramble key, making it trivially detectable. Furthermore, our analysis showed that all installations share the same HMAC key, enabling precise endpoint verification. Unlike prior timing-based methods [78], this allows an adversary to craft a valid P_CONTROL_HARD_RESET_CLIENT_V2 message, triggering a P_CONTROL_HARD_RESET_SERVER_V2 response, confirming the endpoint as OpenVPN.
2) ***ExpressVPN:*** ExpressVPN uses OpenVPN with additional encryption for configuration files, which are only available to subscribers. Unlike NordVPN, it offers a throughput test on macOS/Windows to find optimal servers, inadvertently enabling adversaries to enumerate endpoints (though this feature is absent on Linux/Android).
   During authentication, ExpressVPN avoids exposing the `ClientHello` SNI field by omitting it in TLS/TCP mode (a rare and detectable fingerprint [34]) or using a UDP handshake, which reveals no authentication details. Unlike NordVPN, ExpressVPN does not use the XOR patch, but its endpoints remain probeable. Upon subscription, users receive a static HMAC key, similar to NordVPN, allowing crafted probes to identify gateways, particularly on Linux where throughput tests fail. Other vulnerabilities align with NordVPN's.
3) ***ProtonVPN:*** ProtonVPN supports OpenVPN (UDP/TCP) and TLS-based tunneling. Authentication occurs over HTTPS/TLS (`verify.proton.me` as SNI), making it easily identifiable and blockable. As with ExpressVPN, users receive HMAC handshake keys post-subscription, facilitating endpoint confirmation.
   ProtonVPN caches VPN gateway IPs post-authentication, connecting solely to them. We confirmed this by disconnecting the network, launching the client, and observing connection attempts to all regional endpoints. This behavior helps adversaries enumerate and block VPN gateways.

B. *Analyzing TLS and SSH-Based Services*

Several VPNs utilize TLS-based tunneling to imitate browser-based handshakes, evading censors. [3].

*1) **Analyzing TLS-based services**:*

1) ***NordVPN:*** Besides OpenVPN, NordVPN also provides TLS-based VPN services. This works as a browser-integrated TCP proxy. Only the browser's traffic is encrypted and encapsulated; other protocol messages *e.g.* DNS queries are not sent via this tunnel.Protocol analysis showed that the browser plugin connects to standard VPN gateways also used for OpenVPN. However, the browser plugin connects to port 80 on the gateway but uses encrypted traffic. Furthermore, we discovered that it uses a TLSv1.2 handshake with SNI pointing to `xx.nordvpn.com`. We observed that this mode is only supported by the browser extension, and not the regular NV client application.
2) ***ExpressVPN:*** ExpressVPN's proprietary *Lightway TCP* protocol establishes a TLSv1.3 tunnel for both the handshake and encrypted payload transmission. Packet analysis revealed that authentication and data transfer share the same connection, with traffic appearing as standard TLSv1.3 in tools like Wireshark. However, anomalies include the absence of SNI extensions and the use of a single cipher suite (ECDHE-RSA with AES-256-GCM and SHA384), both uncommon in typical browser traffic and potentially raising suspicion.
3) ***ProtonVPN:*** ProtonVPN's proprietary `Stealth` protocol leverages TLSv1.3 to fully mimic browser-initiated handshake messages. Its `ClientHello` packets include an obscured, randomized SNI field (*e.g.*, `'bdcyw.tr'`, `'ofd.es'`) and advertise 16 cipher suites, aligning with typical browser behavior. Unlike ExpressVPN's Lightway TCP, this design makes Stealth traffic challenging to detect by adversaries.

[3] These are often labeled as "proprietary"

4) ***VyprVPN:*** VyprVPN supports a proprietary TLS-based protocol called *Chameleon*, which encapsulates transport packets using TLSv1.3. However, this outer TLS can be identified through the `ClientHello` SNI field containing strings such as `api.goldenfrog.com`.
5) ***Perfect Privacy:*** PerfectPrivacy supports a TLS-based protocol (OpenVPN via `Stunnel`) using TLSv1.2, where the server certificate is transmitted in plaintext, making it identifiable. It also offers SSH-based tunneling, which does not exhibit identifiable patterns.
6) ***Windscribe:*** Windscribe offers two proprietary protocols, *Stealth* and *Wstunnel*, alongside open protocols such as OpenVPN and WireGuard. In all cases, the client first authenticates via TLSv1.3 with an authentication server, whose `ClientHello` SNI field includes `api.windscribe.com`. This identifiable string could lead to blocking Windscribe entirely. Additionally, as the authentication server is hosted on Cloudflare, altering the SNI field results in connection failures (as detailed in Subsec. VI-A). Post-authentication, Stealth and Wstunnel protocols use TLSv1.3 for VPN gateway communication, embedding the gateway's IP address directly in the `ClientHello` SNI field as an ASCII string. Consequently, **ISP middleboxes** configured to filter packets based on domain names *may* fail to recognize and block these VPN packets.

*2) **Analyzing SSH-based services**:* SSH also provides built-in tunneling capabilities, encrypting system/application traffic using negotiated SSH keys and transporting it via SSH packets to the peer. *Perfect Privacy* is a popular example of an SSH-based VPN service. Analysis of Perfect Privacy's VPN traffic across modes (SSH, OpenVPN, TLS, etc.) revealed that OpenVPN can be tunneled over SSH using SSH-negotiated keys. All VPN gateways reside in the same subnet, which could hint at a VPN cluster. However, the traffic could also plausibly resemble standard SSH in a cloud environment.

C. Analyzing Wireguard Services

*1) **Analyzing vanilla Wireguard**:* Wireguard is increasingly being adopted as the new default mode on all selected VPNs. To analyze the behavior of the Wireguard protocol, we have set up the Wireguard server and client machines running Linux.

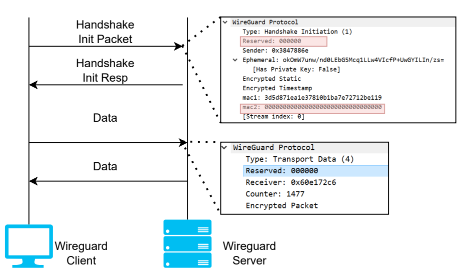

Fig. 2: Wireguard handshake(highlighting identifiable patterns in handshake and data packets).

Using Wireshark, we analyzed WireGuard handshake and data packets, which are encapsulated within UDP packets. All

VPN Fingerprintability				
Protocol	**Type**	**Provider**	**Fingerprintable**	**Remarks**
OpenVPN (vanilla)	TCP UDP	NV,EV,PV	●	Control and data channel plaintext opcode patterns.
OpenVPN (Vanilla)+PSK	TCP UDP	None	○	Full packet encryption no visible patterns in traffic flow.
OpenVPN (Vanilla-XOR-based)	TCP UDP	NV,EV	◐	Control and data channel partially obfuscated using fixed XOR-string, but repeating sequence of apparently XOR-ed opcode at the same offset. Difficult to detect by commercial middleboxes like Fortinet 2600F series.
OpenVPN (Over Stunnel)	TCP	PP	○	Similar to the browser traffic.
OpenVPN (Over Obfsproxy)	TCP	PP	◔	No identifiable pattern but Obfsproxy can be identified [35].
OpenVPN (Over SSH)	TCP	PP	◐	Packet length sequence Identifiable
OpenVPN (Chameleon)	TCP	VP	○	Fully encrypted OpenVPN. flow with no identifiable features.
TLSv1.3 (Lightway)	TCP	EV	◔	Advertizes only a single cipher. suite in the TLS `ClientHello`.
TLSv1.3 (Stealth)	TCP	PV,WS	○	Similar to browser traffic with obscured SNI.
Websocket	TCP	WS	◔	SNI field and server name identifiable.
Wireguard (Vanilla)	UDP	VP,PV,WS	●	Fixed/repeating patterns in handshake packets.
Wireguard (NordLynx)	UDP	NV	●	Fixed/repeating patterns in handshake. packets.
Wireguard (WARP)	UDP	CF	●	Uses fixed/repeating pattern in Wireguard's `reserved` field.
UDP (Lightway)	UDP	EV	◔	Plaintext fingerprintable characteristic. in a single handshake message
IPsec/IKEv2	UDP	WS,VP,PP	○	High collateral damage if filtered.

TABLE I: The 4 levels of VPN identifiability (*Non-identifiable* (○)– No identifiable characteristics, *partially-identifiable*(◔)– *i.e.*, only 1 packet contains identifiable information, Identifiable (◐)-identifiable under certain conditions, and Easily Identifiable(●)–fixed across multiple packets) and their service providers as of 2024. The last column lists identifiable features of each specified protocol.

WireGuard packets start with three reserved bytes (immediately following the UDP header) set to zero. Additionally, the handshake initialization packet (HANDSHAKE_INIT) includes two 16-byte fields, MAC1 and MAC2, both initialized to zero. Moreover, all packets are relatively small (< 200 bytes). These characteristics render WireGuard susceptible to detection (see Figure 2).

*2) **Analyzing commercial Wireguard-based services**:*

1) ***NordLynx (NordVPN-wireguard):*** NordLynx is a proprietary system built upon vanilla WireGuard with additional privacy enhancements. Its designers identified privacy risks with WireGuard endpoints due to maintaining mappings between clients and their visited sites. NordLynx addresses this by translating all client source IPs into a single IP (similar to NAT) before routing packets to their destinations, preventing gateways from associating specific users with visited sites. Nevertheless, the payload of NordLynx packets remains indistinguishable from vanilla WireGuard, thus equally susceptible to detection.
2) ***WARP:***
WARP, developed by Cloudflare, is derived from the WireGuard protocol. Upon launching the application, a TLSv1.3 handshake occurs, sending a `ClientHello` message with the SNI field set to `api.cloudflareclient.com`.

After this initial exchange, subsequent UDP packets are sent to a different IP address. Using Wireshark's WireGuard dissector, we confirmed most of these packets as WireGuard data packets. An adversary might suspect these packets to be WireGuard traffic due to the identifiable `ClientHello` message containing the specific SNI. Unlike vanilla WireGuard, WARP's first three reserved bytes are not zeros but represent the user's clientID [22].

D. Analyzing IPSec(ESP)/IKEv2 Based Services

We also analyzed IPSec(ESP) based VPN services. It is the oldest form of VPN/tunneling scheme. It integrates endpoint authentication with message confidentiality and integrity. IPSec relies on IKEv2 for key exchange. It provides security guarantees like TLS.

We identified four VPN services using IPSec/IKEv2 for tunnel setup and encrypted data transport: Windscribe, VyprVPN, ProtonVPN, Perfect Privacy. For each service, we analyzed user authentication mechanisms and gateway communication.
Authentication server messages: Authentication occurred via HTTPS between clients and authentication servers. TLS `ClientHello` messages typically included SNI fields matching the service providers' domains. Except for Windscribe, hosted on Cloudflare, the SNI fields of other providers could be manipulated for obfuscation.
Encrypted/encapsulated data transport: Traffic between clients and VPN gateways was encrypted and encapsulated as IPSec (ESP), exhibiting no identifiable patterns. Arbitrarily blocking IPSec/IKEv2 traffic could lead to collateral damages, since services like VoLTE, VoWiFi, and corporate firewalls, actively rely on ESP encapsulation [30], [33], [48], [80].

V. Fingerprint Evasion and Firewalls

To understand how to bypass identification, it is important to understand how a client connects to a VPN end-point. As described above in Sec. IV, it is a two-step process. The first step involves communication to the authentication server that shares important information like the gateway IPs, keys for various purposes *etc.*. The adversary can virtually block all commercial VPN services by identifying and dropping such messages. The second step involves identifying packets that are likely destined for the gateways.

We now describe various fingerprint evasion strategies that may be implemented at client side.

A. ClientHello SNI Poisoning

As noted, VPN services host their authentication servers in cloud platforms like Amazon, Cloudflare, Fastly and Akamai, and they are typically accessed with standard HTTPS. The TLS `ClientHello` SNI field often reveals target domain(e.g.: `napps1.com` of NordVPN), opening a possibility of evading the detection by fudging this field. We tested this by manipulating the SNI during the VPN handshake; the server responded with the login page, confirming acceptance. This worked for servers on Akamai, GSL Network, Micron Hosting, and AWS but consistently failed on Cloudflare, which strictly enforces domain-SNI matching. This notably affects NordVPN, which relies heavily on Cloudflare. Alternatively, modifying `/etc/hosts` for local DNS resolution—mapping the portal's IP to an unfiltered domain—and setting the SNI accordingly can bypass detection, except on Cloudflare. Encrypted DNS (DoT/DoH) is another viable option. For such tests, we used Python's `scapy` library to manage TLSv1.2 handshakes. We used Linux `netfilter queue` to intercept packets and modify the SNI field of TLS `ClientHello` message in real-time. Thereafter, we forwarded the modified packets after recalculating the necessary checksums.

B. Evade Middleboxes by MTU Manipulation

For bypassing DPI of stateful firewalls, suspicious patterns could be distributed across multiple packets rather than clustered in one. We achieved by lowering the client's Maximum Transmission Unit (MTU) [4], causing the fields like SNI to be spread across packets. A typical `ClientHello` packet is 400–600 bytes. We reduced the interface MTU 500 bytes using Linux `net-tools ifconfig` package. This approach obscures authentication (often over HTTPS).

To test firewall state disruption, we sent small-sized web requests to sites, that we blocked on a local Linux `iptables` deployment, with a supporting `netfilterqueue` python program, filtering TLS ClientHello packets, based on SNI header field[5]. Using the default MTU sizes (1500 bytes), the packets were blocked. However, reducing it to 500 bytes, the packets go undetected. This strategy also helped access sites that were otherwise blocked by the government [62].

1) ***Impact of MTU Manipulation****:* To assess the performance impact of MTU reduction, we measured page load times for Trancolist[583GN] [49] top 1k websites via selected VPN protocols, comparing results before and after lowering the MTU from 1500 to 500 bytes.

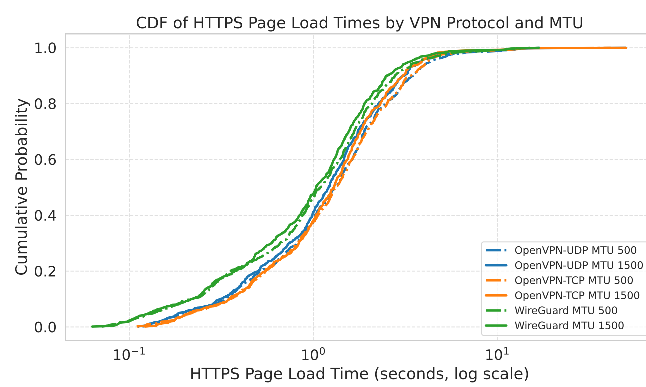

Fig. 3: Cumulative distribution of HTTPS page load times for different VPN protocols (OpenVPN-UDP, OpenVPN-TCP, and WireGuard) under varying MTU sizes (500 and 1500 bytes). WireGuard generally offers faster load times, while OpenVPN shows minimal performance variation between MTU settings.

[4] *I.e.*, distinct from IP fragmentation, which middleboxes may reassemble.
[5] Alternatively, a netfilter kernel module could also be used for the same.

As shown in Fig. 3, reducing MTU to 500 bytes does not significantly impact page-load times accross VPN protocols. DNS requests can reveal VPN usage, and hence, users may employ DNS over HTTPS (DoH) or DNS over TLS (DoT) to evade adversaries, necessitating evaluation of their added latency. Using Cloudflare's 1.1.1.1 DoH and DoT services, we compare baseline DNS request response time, against DoH and DoT with various MTU sizes. As evident from Fig. 4, the performance of DoH and DoT doesn't vary much across MTU values, although they are about 2.5s slower than regular DNS.

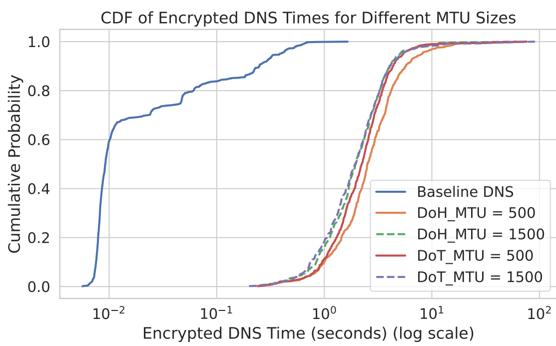

Fig. 4: Cumulative distribution of DNS resolution times (log scale) for plaintext DNS (baseline) versus encrypted DNS protocols (DoH and DoT) under different MTU sizes.

C. Sending Malformed OpenVPN Packets

1) ***Using malformed packets that would likely confuse middleboxes****:* In OpenVPN, every packet bears known opcodes that signify their types. In previous work [78], an attacker records bidirectional traffic to detect repeating opcode patterns Thus, we tried sending malformed packets (*i.e.* copies of the original packets with altered opcode fields). These were ignored by the server due to HMAC mismatches. The client then retransmits the original packet, and the communication ensues normally. This likely disrupts the opcode fingerprinting algorithm (Algorithm-1) presented in Section 6.1 of [78].

2) ***Actively probing OpenVPN gateways with malformed packets****:* Xue *et al.* proposed identifying VPN endpoints via active probing, exploiting differences in OpenVPN's connection termination times, based on message timeouts vs. incorrect HMAC verification. However, we found that OpenVPN servers typically ignore handshake packets with incorrect HMACs, prompting the client to retransmit and continue communication.

D. Firewalls

We tested the DPI capabilities of our campus Fortinet 2601F firewall (FortiOS 7.6 [11]) by hosting a vanilla OpenVPN cloud server. With the VPN filter enabled, it blocked vanilla OpenVPN traffic, but failed to detect traffic for XOR-obfuscated OpenVPN server [14]. Although the XORed byte pattern remains deterministic across packets(§IV-A), the firewall cannot dynamically learn/inspect these transformed sequences at arbitrary positions, allowing obfuscated traffic to pass. Modern firewalls [10]–[13] typically rely on static, hardcoded signatures tied to known protocol behaviors, which makes them effective against standard VPN traffic but insufficient against obfuscation techniques. Xue *et al.*'s method detects XOR-obfuscated VPN connections using SIEM-tools like *Zeek* [1]. Unlike middleboxes, this approach is impractical for real-time blocking at ISP scales due to its offline nature and scalability limitations.

VI. Discussions

A. Limitations and Challenges

We now elaborate on the various limitations of our work and the challenges we faced during our study.

1) ***WARP's localized gateways:*** WARP is a CloudFlare product and is designed to connect to the nearest CDN edge server (hosting the VPN gateway service). While this provides good performance, the CDN often causes the clients to connect with edge-hosted VPN gateways in the same country. This altogether defeats the very purpose of using VPNs. There is a good chance the client may not evade censorship by ISPs in his/her own country.
2) ***SNI, HTTP host and Common Name mismatches:*** We discovered that for VPN authentication sites hosted on Cloudflare, the edge servers usually forward the request to the appropriate domain if the HTTP Host field of the inner HTTP packet matches the SNI field. In case of a mismatch between the two fields, *e.g.* in case of crafted SNIs, the connection terminates with a HTTP 403.
3) ***Enumerating VPN end-points*****:** A determined adversary could identify VPN endpoints by repeatedly connecting to their gateways and blocking the associated IPs. To simulate this, we performed automated connection attempts every 30 minutes for several weeks from cloud-hosted clients in the USA, Germany, Australia, Japan, and India, using default client settings. Our study focused on NordVPN, ProtonVPN, and WARP.[6]
While trying to enumerate the end-points, we discovered an interesting way of working of ProtonVPN. During the initial handshake, the client retrieves a list of IP pairs and their associated countries via a public API. Further analysis revealed a two-hop architecture, where a single entry IP may route traffic to multiple exit IPs [9]. Over three months, we collected data from the API, compiling a comprehensive list of 9,737 unique entry-exit pairs across 112 countries.
We retrieved NordVPN's IP address list using a public API. Unlike Proton, IPs were not paired. Using the same methodology, we identified 7,418 unique IPs spanning 112 countries.
Cloudflare WARP clients connect to region-specific gateways, typically within the same country, based on CDN routing for performance. Testing from multiple locations showed clients consistently receiving IPs from a limited set

[6] VyprVPN and Windscribe were excluded due to Linux compatibility issues. Windscribe's CLI consistently crashed on connection attempts.

of Cloudflare-owned /24 or /16 subnets, indicating localized routing with preferred regional IP clusters. However, blocking these gateways is difficult due to dynamic IP assignment and CDN-hosted infrastructure.

4) ***Temporal variations in VPNs' characteristics*:** We conducted our tests and measurements over a span of one year, repeating them as-is three times. For each of these, we used the same setup and the VPN services. Traffic generated and type of transport used didn't change.
5) ***Use of learning-based heuristics to spot VPNs*:** Encrypted traffic analysis and traffic fingerprinting involving ML/DL classifiers are often prone to high FPs and FNs, especially when applied without considering the importance of the features and expected outcomes. Also, such models temporally degrade, due to the dynamic nature of network traffic [24]. There would always be flows whose patterns were not considered while training. Such models would require periodic training. It is not impossible to surmise such threats, but they may be only within the reach of powerful adversaries, that may have massive computing (and cooling) infrastructure, that may not be feasible [36].
6) ***VPN configurations, versions and modes:*** This study's results stem from testing VPN applications of various providers using versions and configurations reflected in Table II. Future updates, including new features, security patches, or tunnelling methods, may alter connection speeds, stability, and detectability. Our methodology and findings provide diagnostic mechanisms for users and VPN providers, identifying design weaknesses, and guide improvements. As VPNs evolve, stakeholders should regularly reassess performance and resilience for optimal security and reliability.
7) ***VPN Protocol fallbacks:*** We tested commercial VPN providers' built-in protocol fallback mechanisms by deliberately blocking the primary protocols at our firewall. When WireGuard traffic was restricted, PV's *Smart Protocol* feature automatically shifts traffic to a TCP OpenVPN. Similarly, VP employs *Automatic Troubleshooting* that methodically cycles through available protocols, including its proprietary *Chameleon* protocol. WS offers *Smart VPN Automation* allowing protocol-specific configurations for different networks with automatic fallback capabilities. EV prioritizes its proprietary Lightway protocol but reverts to OpenVPN when connectivity issues arise. Similarly, NV's WireGuard-based NordLynx protocol automatically switches to OpenVPN (TCP) when faced with connection failures. Such adaptive behavior is consistent across multiple providers, each implementing its fallback strategy.

B. Key Management and Future Work

Most VPN protocols employ some form of key exchange protocol for its clients, employing mechanisms to ensure forward secrecy, deriving per user encryption and signature keys, used for the tunnel's confidentiality and message integrity. However, the handshake protocols often bear plaintext information, identifiable by DPI-capable middleboxes.

For such cases, OpenVPN's PSK mode offers a notable advantage by encrypting the complete data packet, including the opcode field, effectively concealing protocol-specific signatures, at the cost of key-management capabilities. The PSK mode requires distributing unique symmetric key files to each user, posing distribution and management challenges, besides lacking forward secrecy. On the contrary, WireGuard, despite using efficient cryptographic primitives, remains susceptible to identification, due to distinct handshake patterns. Some commercial providers have implemented proprietary key management schemes, but don't safeguard against identification.

Correspondingly, NV's NordLynx generates ephemeral WireGuard keys per session, managed and expired via database entries [50]. PV clients require signing into `account.protonvpn.com`, and generating custom WireGuard configuration files, with fresh public-private key pairs, which are downloaded and used, with identity keys revoked upon logout or session closure [75]. Session keys are negotiated every couple of minutes by WireGuard's internal 1-RTT handshake (deriving new symmetric keys for each interval) [63]. Similarly, CF employs a proprietary *Key Management System (KMS)* to generate keys for its WireGuard clients, relying on zero-trust JWTs for client authentication [6], [8].

This fundamental tension between key exchange and management, and protocol obfuscation, necessitates a new approach that achieves both objectives simultaneously without requiring proprietary infrastructure. To address this gap, We propose a novel OpenVPN handshake using per-session Diffie–Hellman key exchange instead of static pre-shared keys, providing fresh encryption and signature keys with forward secrecy and minimal latency. Appendix A outlines the protocol, security rationale, and considerations for prime generation and key derivation. Implementation and empirical validation are planned for a future study.

C. Ethical Considerations

Our experiments, conducted over months to minimize network and VPN gateway load, never disrupted or breached other users' traffic. Critics may claim that our work aids repressive regimes by highlighting VPN traffic patterns for blocking. However, our goal is to help netizens select VPNs to evade surveillance and censorship. We demonstrate that distinguishing OpenVPN or WireGuard flows between private/corporate and commercial uses is non-trivial. Protocols like IPSec are widely used for tunnelling (*e.g.* 4G Video over Wifi), lead to collateral damage if arbitrarily filtered.

VII. Conclusion

As censorship increase, governments increasingly monitor VPNs. Thus we see an urgent need to evaluate how easily ISP-scale censors can identify VPN flows. Prior work showed OpenVPN flows and endpoints are easily detected in the wild. We offer a new perspective, finding alternatives like TLS, SSH, and IPSec harder to spot and block without collateral damage. We also explored evasion by reducing MTU to split detectable bytes across packets, minimally affecting performance.

References

[1] "The zeek network security monitor." [Online]. Available: https://zeek.org/

[2] "Nordvpn," https://nordvpn.com/best-vpn-site/, 2012.

[3] "Perfect privacy," https://www.perfect-privacy.com/, 2012.

[4] "Mitm proxy," https://mitmproxy.org/, 2013.

[5] "Windscribe vpn," https://windscribe.com/, 2016.

[6] "Cloudflare warp: Challenges," https://blog.cloudflare.com/warp-technical-challenges/, 2020, accessed: 2025-05-06. [Online]. Available: https://blog.cloudflare.com/warp-technical-challenges/

[7] "Wikipedia,(vpn)," https://en.wikipedia.org/wiki/Virtual_private_network, 2022.

[8] "Cloudflare security," 2023. [Online]. Available: https://developers.cloudflare.com/reference-architecture/architectures/security/

[9] "Alternative routing," https://protonvpn.com/blog/deutsche-welle-partnership, 2024.

[10] "Checkpoint firewall," https://www.checkpoint.com/quantum/application-control/, 2025.

[11] "Fortinet vpn filter," https://fortiguard.fortinet.com/appcontrol/17244, 2025.

[12] "Juniper firewall," https://threatlabs.juniper.net/home/search/#/details/?sigtype=app&sigid=ApplicationID:OPENVPN, 2025.

[13] "Paloalto firewall," https://docs.paloaltonetworks.com/network-security/security-policy/administration/objects/applications, 2025.

[14] "Tunnelblick," https://tunnelblick.net/cOpenvpn_xorpatch.html, 2025.

[15] G. Aceto, A. Montieri, and A. Pescapé, "Internet censorship in Italy: An analysis of 3G/4G networks," in *IEEE ICC*. IEEE, 2017, pp. 1–6.

[16] H. B. Acharya, A. Ramesh, and A. Jalaly, "The world from Kabul: Internet censorship in Afghanistan," in *IEEE INFOCOM WKSHPS*, April 2019, pp. 1061–1062.

[17] AFP, "Vpns in iran," https://telecom.economictimes.indiatimes.com/news/iran-to-criminalise-sale-of-vpns-minister/94975787, 2022.

[18] Apple Inc., "App Store," https://www.apple.com/in/app-store/.

[19] S. Aryan, H. Aryan, and J. A. Halderman, "Internet censorship in Iran: A first look." in *USENIX workshop on (FOCI)*, 2013.

[20] A. Avilov, "Russia will block vpn services," https://www.themoscowtimes.com/2024/02/05/marc-fogel-my-american-teacher-is-the-kremlins-political-pawn-a83973, 2024.

[21] D. Barradas, N. Santos, L. Rodrigues, and V. Nunes, "Poking a hole in the wall: Efficient censorship-resistant internet communications by parasitizing on WebRTC," in *Proceedings of the 27th ACM (CCS '20)*, 2020.

[22] Z. Bloom, "Warp blog - technical challenges," https://blog.cloudflare.com/warp-technical-challenges/, 2019.

[23] A. Chaabane, T. Chen, M. Cunche, E. De Cristofaro, A. Friedman, and M. A. Kaafar, "Censorship in the wild: Analyzing internet filtering in Syria," in *ACM/SIGCOMM IMC*. ACM, 2014, pp. 285–298.

[24] G. Cherubin, R. Jansen, and C. Troncoso, "Online website fingerprinting: Evaluating website fingerprinting attacks on tor in the real world," in *USENIX Security*, 2022, pp. 753–770. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/cherubin

[25] Cloudflare, "Cloudflare warp," https://1.1.1.1/, 2018.

[26] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*. USENIX Association, 2004.

[27] J. A. Donenfeld, "Wireguard: next generation kernel network tunnel." 2017.

[28] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of encrypted and vpn traffic using time-related features," in *International Conference on Information Systems Security and Privacy*, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:21535780

[29] R. Ensafi, P. Winter, A. Mueen, and J. Crandall, "Analyzing the great firewall of china over space and time," *PoPETS*, vol. 1, 04 2015.

[30] ERNW, "Ernw whitepaper," https://ernw.de/download/newsletter/ERNW_Whitepaper_60_Practical_Attacks_On_VoLTE_And_VoWiFi_v1.0.pdf, 2017.

[31] ExpressVPN, "Expressvpn," https://www.expressvpn.com/, 2009.

[32] ExpressVPN, "We've reached 4 million active subscribers!" https://www.expressvpn.com/blog/4-million-subscribers/, 2023, accessed: 2025-03-29.

[33] Fortinet, "Fortinet commercial: Configuring the ipsec vpn," https://docs.fortinet.com/document/fortigate/5.6.0/cookbook/786021/configuring-the-ipsec-vpn, 2022.

[34] S. Frolov and E. Wustrow, "The use of tls in censorship circumvention," in *Proceedings of the 26th Symposium on (NDSS '19)*. Internet Society, February 2019.

[35] A. Færøy, "A child's garden of pluggable transports," https://gitlab.torproject.org/legacy/trac/-/wikis/doc/AChildsGardenOfPluggableTransports, 2021.

[36] K. Gallagher, D. Barradas, and N. Santos, "Rethinking realistic adversaries for anonymous communication systems," *PoPETS FOCI*, vol. 23, pp. 81–87, 2023.

[37] G. Gebhart and T. Kohno, "Internet censorship in Thailand: User practices and potential threats," in *IEEE (EuroS&P)*. IEEE, 2017, pp. 417–432.

[38] J. Geddes, M. Schuchard, and N. Hopper, "Cover your acks: Pitfalls of covert channel censorship circumvention," in *ACM CCS*, November 2013.

[39] Google, "Google Play," https://play.google.com, accessed: 2025-03-29.

[40] D. Gosain, A. Agarwal, S. Shekhawat, H. B. Acharya, and S. Chakravarty, "Mending wall: On the implementation of censorship in india," in *SecureComm*. Springer International Publishing, 2018.

[41] C. M. Hagan, "Openvpn xor patch," https://github.com/clayface/openvpn_xorpatch, 2018.

[42] J. Hayes and G. Danezis, "K-fingerprinting: A robust scalable website fingerprinting technique," in *USENIX Security*. USENIX Association, 2016.

[43] N. P. Hoang, A. A. Niaki, J. Dalek, J. Knockel, P. Lin, B. Marczak, M. Crete-Nishihata, P. Gill, and M. Polychronakis, "How great is the great firewall? measuring china's DNS censorship," in *USENIX Security*, 2021. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/hoang

[44] R. Hodge, "Best VPN Service for 2025," https://www.cnet.com/tech/services-and-software/best-vpn/, 2025, accessed: 2025-03-29.

[45] S. Jain, "India crackdown on vpns," https://www.businessinsider.in/tech/news/india-expands-its-crackdown-on-vpn-bans-its-employees-from-using-vpn-and-cloud-services/articleshow/92272901.cms, 2022.

[46] J. Karlin, D. Ellard, A. W. Jackson, C. E. Jones, G. Lauer, D. P. Mankins, and W. T. Strayer, "Decoy routing: Toward unblockable internet communication," in *USENIX FOCI*, August 2011.

[47] S. Khattak, L. Simon, and S. J. Murdoch, "Systemization of pluggable transports for censorship resistance," *CoRR*, vol. abs/1412.7448, 2014.

[48] D. R. Kuhn, T. J. Walsh, and S. Fries, "Sp 800-58: Security considerations for voice over ip systems," 2005.

[49] V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczynski, and W. Joosen, "Tranco: A research-oriented top sites ranking hardened against manipulation," in *NDSS*, ser. NDSS 2019. Internet Society, 2019. [Online]. Available: http://dx.doi.org/10.14722/ndss.2019.23386

[50] N. Lynx, "Nordlynx," https://nordvpn.com/blog/nordlynx-protocol-wireguard/, 2020, accessed: 2025-05-06. [Online]. Available: https://nordvpn.com/blog/nordlynx-protocol-wireguard/

[51] H. M. Moghaddam, B. Li, M. Derakhshani, and I. Goldberg, "Skypemorph: Protocol obfuscation for Tor bridges," in *ACM CCS*, October 2012.

[52] Z. Nabi, "The anatomy of web censorship in pakistan." in *USENIX FOCI*, 2013.

[53] M. Nasr, A. Bahramali, and A. Houmansadr, "Deepcorr: Strong flow correlation attacks on tor using deep learning," in *ACM CCS*, 2018. [Online]. Available: https://doi.org/10.1145/3243734.3243824

[54] Netgate, "pfsense," https://www.pfsense.org/, 2023.

[55] M. Network, "Mysterium network," https://www.mysterium.network/, 2024.

[56] NordVPN Blog, "Nordvpn survey reveals: Users still trust free vpns," https://nordvpn.com/blog/nordvpn-usage-survey/, 2024, accessed: 2025-03-29.

[57] NYM, "Nymvpn," https://nymvpn.com/en/nymVPN-litepaper.pdf, 2024.

[58] A. Panchenko, F. Lanze, J. Pennekamp, T. Engel, A. Zinnen, M. Henze, and K. Wehrle, "Website fingerprinting at internet scale." in *NDSS*, 2016.

[59] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, "Website fingerprinting in onion routing based anonymization networks," in *ACM WPES*, 2011, p. 103–114. [Online]. Available: https://doi.org/10.1145/2046556.2046570

[60] ProtonVPN, "Protonvpn," https://protonvpn.com/, 2018.

[61] R. Ramesh, L. Evdokimov, D. Xue, and R. Ensafi, "Vpnalyzer: systematic investigation of the vpn ecosystem," in *NDSS*, 2022, pp. 24–28.

[62] K. Singh, G. Grover, and V. Bansal, "How India censors the web," in *ACM WebSci*, 2020, pp. 21–28.

[63] P. Support, "Wireguard key rotation," https://protonvpn.com/support/wireguard-key-rotation, 2023, accessed: 2025-05-06.

[64] W. D. Team, "Wireguard protocol," 2023, accessed: 2025-05-06. [Online]. Available: https://www.wireguard.com/protocol/

[65] TechRadar, "The best VPN service 2025," https://www.techradar.com/vpn/best-vpn, 2025, accessed: 2025-03-29.

[66] R. Usman, "Beyond borders: Exploring global vpn censorship," https://www.digitalinformationworld.com/2023/07/beyond-borders-exploring-global-vpn.html, 2023.

[67] R. Verma, "Explained: Indian government makes user data collection mandatory for vpns," https://www.businessinsider.in/tech/news/it-ministry-orders-vpn-providers-to-store-user-data-for-five-years-tech-news/articleshow/91334830.cms, 2022.

[68] A. Vigderman and G. Turner, "Best VPN of 2025," https://www.security.org/vpn/best/, 2025, accessed: 2025-03-29.

[69] VyprVPN, "Vyprvpn," https://www.vyprvpn.com/, 2010.

[70] L. Wang, K. P. Dyer, A. Akella, T. Ristenpart, and T. Shrimpton, "Seeing through network-protocol obfuscation," in *Proceedings of the 22nd ACM (CCS '15)*. New York, NY, USA: ACM, October 2015, pp. 57–69. [Online]. Available: http://doi.acm.org/10.1145/2810103.2813715

[71] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, "Effective attacks and provable defenses for website fingerprinting," in *USENIX Security*. USA: USENIX Association, 2014.

[72] Z. Wang, Y. Cao, Z. Qian, C. Song, and S. V. Krishnamurthy, "Your state is not mine: A closer look at evading stateful internet censorship," in *ACM/SIGCOMM IMC*, 2017, pp. 114–127.

[73] Wikipedia, "Miller–rabin primality test," https://en.wikipedia.org/wiki/MillerRabin_primality_test, 1980.

[74] P. Winter and S. Lindskog, "How the Great Firewall of China is blocking Tor," in *USENIX workshop on (FOCI)*, 2012.

[75] P. Wireguard, "Vpn config," https://protonvpn.com/support/wireguard-configurations, 2023, accessed: 2025-05-06. [Online]. Available: https://protonvpn.com/support/wireguard-configurations

[76] E. Wustrow, C. M. Swanson, and J. A. Halderman, "Tapdance: End-to-middle anticensorship without flow blocking," in *USENIX Security*, 2014.

[77] D. Xue, B. Mixon-Baca, ValdikSS, A. Ablove, B. Kujath, J. R. Crandall, and R. Ensafi, "TSPU: Russia's Decentralized Censorship System," in *ACM/SIGCOMM IMC*. Nice, France: Association for Computing Machinery, 2022.

[78] D. Xue, R. Ramesh, A. Jain, M. Kallitsis, J. A. Halderman, J. R. Crandall, and R. Ensafi, "OpenVPN is open to VPN fingerprinting," in *USENIX Security*, 2022. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/xue-diwen

[79] T. K. Yadav, A. Sinha, D. Gosain, P. K. Sharma, and S. Chakravarty, "Where the light gets in: Analyzing web censorship mechanisms in India," in *ACM/SIGCOMM IMC*, 2018.

[80] T. Yildirim and P. Radcliffe, "Voip traffic classification in ipsec tunnels," in *2010 International Conference on Electronics and Information Engineering*, vol. 1, 2010, pp. V1–151–V1–157.

APPENDIX

A. Proposed Protocol

We present a step-by-step walkthrough of our scheme with reference to Fig. 5.

1) The server generates a tuple $T = (g, p, g^y, S_r)$, bearing the DH exponent information and a random number, and shares it with the authentication server.
2) The client authenticates against the authentication server.
3) The server responds with the tuple T to the client, bearing other information like the gateway's IP.
4) The client generates its own DH public exponent $g^x \text{mod} p$, and random number C_r, using x.
5) The client also derives $g^{xy} \text{mod} p$ (premaster-key (PMK)), using the tuple shared by the authentication server.

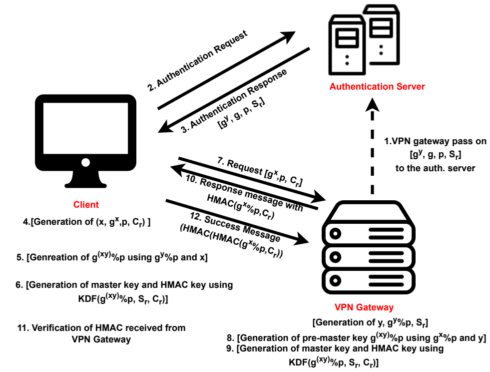

Fig. 5: Proposed per session obfuscation key establishment scheme to overcome challenges of using pre-shared keys.

6) The client uses a KDF to generate the master-key(MK) and signature-key(SK)) from $g^{xy} \text{mod} p$, C_r and S_r.
7) The client sends (C_r, g^x) tuple to the VPN gateway (whose IP address was obtained during user authentication). This would appear as random bytes and would be ignored by the adversary.
8) The gateway also derives $g^{xy} \text{mod} p$, from y and the g^x obtained from the client in step 7.
9) The gateway now derives the session encryption and HMAC keys using $g^{xy} \text{mod} p$, C_r and S_r. These are the same as the ones with the client *i.e.* MK and SK.
10) Using SK, it signs the client⇒server message (from the previous step) and sends it back to the client.
11) The client verifies the received signature using its own copy of SK (derived above in step 6).
12) After verification, the client acknowledges the server using HMAC of the server⇒client message, using SK.

We used Miller-Rabin primality test [73] to generate large random primes.

VPN Provider (App Version)	Protocols Tested
NordVPN (6.15–7.12) Linux/Win/Andr/Mac	NordLynx (WG, default), OVPN-UDP, OVPN-TCP, Browser TLS
ExpressVPN 11.7–11.30) Linux/Win/Andr/Mac	Lightway (TLS 1.3, default), OVPN-UDP, OVPN-TCP, Browser TLS
Warp (2023.8–2024.6) Linux/Win/Andr	Warp (mod. WireGuard, default)
ProtonVPN (4.6–5.3) Linux/Win/Andr	WireGuard (default), OVPN-UDP, OVPN-TCP, Stealth
VyprVPN (5.1–5.2) Linux/Win/Andr	OVPN-UDP (default), OVPN-TCP, Chameleon
Windscribe (3.75–3.77) Linux/Win/Andr	WireGuard (default), OVPN-UDP, OVPN-TCP, WStunnel, IKEv2/IPsec
Perfect Privacy (11) Linux/Win	OVPN-UDP (default), OVPN-TCP, SSH-2, IKEv2/IPsec

TABLE II: VPN Providers, App Versions, and Protocols Tested (2023–2024)