DNS-Morph: UDP-Based Bootstrapping Protocol for Tor

Rami Ailabouni[1(✉)], Orr Dunkelman[1], and Sara Bitan[2]

[1] University of Haifa, Haifa, Israel
railaboun@staff.haifa.ac.il, orrd@cs.haifa.ac.il
[2] Technion, Haifa, Israel
sarab@cs.technion.ac.il

Abstract. Tor is a popular system for anonymous communication and censorship circumvention on the web, this puts Tor as a target for attacks by organizations and governmental bodies whose goal is to hinder users' ability to connect to it. These attacks include deep packet inspection (DPI) to classify Tor traffic as well as legitimate Tor client impersonation (active probing) to expose Tor bridges. As a response to Tor-blocking attempts, the Tor community has developed *Pluggable Transports (PTs)*, tools that transform the appearance of Tor's traffic flow.

In this paper we introduce a new approach aiming to enhance the PT's resistance against active probing attacks, as well as white-listing censorship by partitioning the handshake of the PT from its encrypted communication. Thus, allowing mixing different PTs, e.g., ScrambleSuit for the handshake and FTE for the traffic itself. We claim that this separation reduces the possibility of marking Tor related communications. To illustrate our claim, we introduce DNS-Morph: a new method of transforming the *handshake* data of a PT by imitating a sequence of DNS queries and responses. Using DNS-Morph, the Tor client acts as a DNS client which sends DNS queries to the Tor bridge, and receives DNS responses from it. We implemented and successfully tested DNS-Morph using one of the PTs (ScrambleSuit), and verified its capabilities.

Keywords: Tor · UDP · DNS · Bootstrapping · Bridge · Pluggable Transport · Censorship · Circumvention

1 Introduction

Censoring countries continuously try to block their citizens' connections to Tor, these attempts began with simple methods like blacklisting Tor's website [35] so users would not be able to reach it and download the Tor client software [33], and got more sophisticated to include actively downloading the Tor nodes (also called relays) list from the Tor *Directory Servers* and blacklisting them, deploying DPI to search for Tor communication characteristics (e.g., Tor's TLS handshake cipher suite [46]), as well as active probing (impersonating a Tor client and connecting to suspicious servers to check whether they run a Tor relay) [1,15,41].

S. Dolev et al. (Eds.): CSCML 2021, LNCS 12716, pp. 244–265, 2021.
https://doi.org/10.1007/978-3-030-78086-9_19

The Tor community, on the other side, develops methods to bypass these blocking attempts, mainly Tor *Bridges*[1] and *Pluggable Transports (PTs)*.

Pluggable Transports (PTs) [34] are a generic framework for the development and the deployment of censorship circumvention techniques. Their main goal is to obfuscate the connection between a Tor client and a bridge serving as a Tor entry guard, so it looks benign. The PT consists of two parts as seen in Fig. 1, one is installed on the Tor client side, and the other is installed on the bridge's side. The PT exposes a SOCKS proxy [26] to the Tor client application, and obfuscates or otherwise transforms the traffic, before forwarding it to the bridge. On the bridge's side, the PT Server side exposes a reverse proxy that accepts connections from PT clients and decodes the obfuscation/transformation applied to the traffic, before forwarding it to the actual bridge application. Data transformation/obfuscation and the reverse operations are done by the *Transport Modules/Obfuscation Protocols* used by the PT.

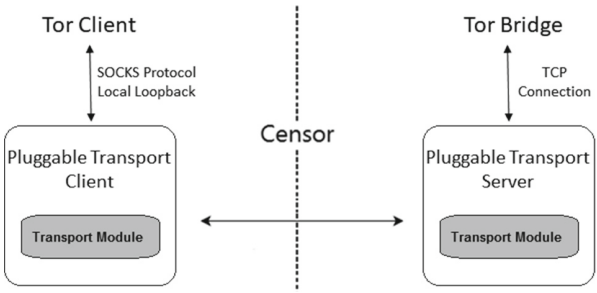

Fig. 1. Pluggable transports design

As of Sep. 2017,[2] the available and deployed obfuscation protocols in the Tor Browser are: obfs3 [22], obfs4 [23], ScrambleSuit [47], FTE [40], and meek [17]. These obfuscation protocols can be divided into two groups:

1. Random stream protocols (obfs3, obfs4, and ScrambleSuit). These protocols' communication is shaped as streams of random bytes that cannot be associated with any known protocol. These protocols have two phases: a handshake phase in which the two participating parties securely exchange keys and/or tickets, and a communication phase, consisting of exchange of encrypted messages using the established keys.
2. Structured stream protocols (FTE and meek), which try to mimic known white-listed protocols such as HTTP.

[1] Bridge: a relay which is unlisted in the public directory servers lists. The information needed to connect to a bridge is obtained out-of-Tor (e.g., via BridgeDB [32], an Email, or in person).

[2] Since Sep. 2017, some changes were introduced to Tor: ScrambleSuit is no longer supported and was replaced by obfs4.

Censoring countries with active probing capability can expose bridges communicating using obfs3. In addition, random stream protocols are of the "look-like-nothing" protocols, which means that they can be identified and blocked by a censor using a white-listing strategy, as their fingerprint (including their handshake) does not fit any known protocol (see for example our experiments with DPI tools in Sect. 10.3).

On the other hand, structured stream protocols that mimic widely used protocols, are resistant to white-listing based blocking. However, they do not protect against active probing [11,18].

Our goal is to improve random stream protocols by using the advantages of structured stream protocols, while maintaining their protection against active probing. We believe that the separation of these protocols to a handshake and communication phase, and encapsulation of their handshake in packets of a known, widely used, white-listed protocol, will strengthen their ability to avoid censorship and detection by DPIs.

DNS was chosen as a protocol to encapsulate the handshake and meet the conditions discussed before for the following reasons:

1. It is one of the most critical protocols of the Internet [25]. Blocking or greatly interfering with this protocol for any reason will induce unacceptable costs on the censoring countries.
2. DNS queries can be automatically relayed from one DNS server to another, until they reach their final destination (recursive DNS queries). This allows relaying data between several DNS servers as an additional layer of protection against connections' tracking and blocking.
3. DNS is a UDP-based protocol. UDP is connection-less and the efforts required to perform DPI of UDP traffic are significantly higher compared to TCP traffic.

1.1 Our Contribution

DNS-Morph is implemented as an obfuscation layer for random stream protocols. This layer encapsulates the protocol handshake in DNS queries and responses in a way that avoids protocol abnormalities.[3]

We focus on the handshake phase in this paper because it is the phase DPI tools and active probes target when searching for Tor traffic and bridges.

In order to show the advantages of DNS-Morph, we implemented it in Python and integrated it into Tor's Obfsproxy code [24]. As Obfsproxy works only with TCP, and our DNS-Morph works with UDP, we added UDP support to Obfsproxy.

We tested our design with the ScrambleSuit protocol[4] in a censoring environment. Success rates, connection timing, and bandwidth are provided in the experiments Section (Sect. 10).

Our source code is available online [6].

[3] Encapsulating the two phases of a protocol into DNS packets will cause high DNS traffic, which can raise suspicion of DNS tunneling.

[4] Similar design should also work for obfs4.

2 Related Work

Protocol Obfuscation: Five obfuscation protocols are deployed and available to use with Tor as PTs:

1. Obfs3 [22]: builds an additional layer of encryption over Tor's TLS connection in order to hide its unique characteristics. An un-authenticated customized Diffie-Hellman handshake [31] is used to exchange encryption keys. As a result, this protocol is susceptible to active probing attacks.
2. ScrambleSuit [47]: protects against active probing attacks by using out-of-band exchanged secrets and session tickets for authentication. ScrambleSuit is also capable of changing its network fingerprint (packet length distribution, inter-arrival times, etc.). This protocol is the predecessor of obfs4 and is subject to white-listing based censoring.
3. Obfs4 [23]: has the same features as ScrambleSuit, but utilizes the *Elligator* technique [38] for public key obfuscation, and the *ntor* protocol [13] for one-way authentication. This results in a faster protocol than ScrambleSuit and the addition of bridge authentication. This protocol can also be blocked by white-listing based censoring.
4. Meek [17]: uses a technique called *Domain Fronting* [4] to relay the Tor traffic to a Tor bridge through third-party servers (i.e., content delivery networks (CDNs) like Amazon CloudFront and Microsoft Azure).
5. Format-Transforming Encryption (FTE) [40]: transforms Tor traffic to arbitrary standard protocols' formats using their language descriptions.

Some other PTs are also available but are not integrated in the Tor Browser. These PTs can be found online [12].

DNS Tunneling: is the act of communicating data of any content inside DNS queries and responses. Three components are used in DNS Tunneling:

1. Client which sends data in DNS queries and acts like a DNS client.
2. Server which tunnels the client data and sends back DNS responses like a DNS server. This server usually has a registered domain name.
3. Encapsulation mechanism of data into DNS queries and responses, and a corresponding decapsulation mechanism for extracting this data from the DNS queries and responses.

Some of the available DNS tunneling tools are Dnscat [9], Dns2tcp [8], and iodine [14].

3 Threat Model

Our threat model consists of a *nation-state censor* that desires to block users from connecting to Tor. This censor might use DPI to examine session packets and active probing to check whether suspected servers are Tor bridges or servers. This censor might also be moving towards a white-listing strategy and

start blocking access to applications for personal use like Skype [30] or WhatsApp [36]. However, we assume that the censor may not be willing to block fundamental services like HTTP, DNS, IMAP, and FTP, as this can break legitimate communications, thus inducing high economical costs and causing unbearable collateral damage.

We believe this threat model is realistic as recent reports suggest that some censoring countries are continuously tightening their Internet control and taking a comprehensive approach to block all outgoing virtual private networks (VPNs) traffic [5].

We also assume that the censor does not perform DNS poisoning. This attack is discussed in Sect. 11.1 (Future Works).

4 Obfsproxy Design

Obfsproxy [24] is an open source software written using Python, and is used by Tor. This software implements the PT design mentioned before, in addition to the obfuscation protocols obfs3 and ScrambleSuit.

Due to the fact that our DNS-Morph is integrated into Obfsproxy, we first describe the Obfsproxy design, and then (in Sect. 5) the modifications done to support DNS-Morph.

Figure 2 (without the red dashed parts) shows the Obfsproxy high level design. Each side, the client and the server, consists of two entities: a Tor client and an Obfsproxy client on the client side, and an Obfsproxy server and a Tor bridge on the bridge side.

Obfsproxy components have two main layers: a networking layer, responsible for connections' establishment, and an obfuscation layer, responsible for the Tor handshake and the data obfuscation.

The connection between the Tor client and the Tor bridge is composed of three components:

1. An "Upstream" connection between the Tor client and the Obfsproxy client on the client side.
2. A "Downstream" connection between the Obfsproxy client and the Obfsproxy server.
3. An "Upstream" connection between the Obfsproxy server and the Tor bridge on the bridge side.

The chronological operation flow of a Tor client and a Tor bridge is as follows:

1. Client side: launches an Obfsproxy client, which starts a TCP SOCKS listener on its upstream connection.
   Bridge side: launches Tor bridge and the Obfsproxy server. The Obfsproxy server starts a TCP listener on its downstream connection. The Tor bridge starts a TCP listener on its upstream connection.
2. Client side: when the Tor client is launched by the user, it connects to the SOCKS TCP listener of the Obfsproxy client on its side. Then, the Obfsproxy client initiates a TCP connection to the Obfsproxy server on the bridge side (downstream connection).

3. Bridge side: when the Obfsproxy server receives the downstream connection request from the Obfsproxy client, it starts its obfuscation layer.
4. Client side: when the downstream TCP connection is established, the client side initiates its obfuscation layer which commences the handshake between the Obfsproxy client and server.
5. If the protocol handshake fails then Obfsproxy modules on both sides close their downstream connection and return to Step 1. If the handshake succeeds, the Obfsproxy server connects to the Tor bridge (upstream connection) and starts receiving data from the Obfsproxy client (downstream connection), decrypting it, and then sending it to the bridge.

At the end of Step 5, the connection between the Tor client and the bridge is fully functional, and the Tor client is connected to the Tor network through the Tor bridge.

5 DNS-Morph Design

The fact that the handshake phase and the encrypted data exchange phase are two separate phases, this allows us to separate them also in the Obfsproxy design and change properties of each phase without affecting the other one.

We now describe our modifications to the Obfsproxy design to replace its handshake by a DNS-based once. Our new DNS-Morph design is shown in Fig. 2, and includes three added components (highlighted in dashed red):

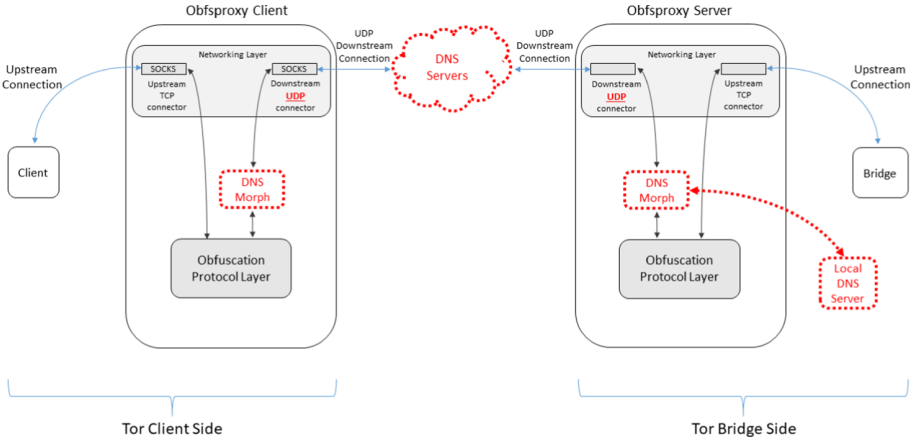

Fig. 2. DNS-Morph design: consists of the Obfsproxy design and the new components (dashed red): downstream UDP connector, DNS-Morph, and a local DNS server (Color figure online)

1. Local DNS server: this is a standard DNS server that receives DNS queries and returns DNS responses, used for active probing resistance, as explained in Sect. 8.
2. Downstream UDP connector: the connector sends and receives UDP packets. The connector also takes care of the reliability functionality (as discussed in Sect. 6).
3. DNS-Morph component:
   (a) Encodes/decodes data to/from Base32 [28].
   (b) Chops data into fragments and encapsulates them into DNS queries or responses.
   (c) Decapsulates data fragments from DNS queries or responses, and reassembles these fragments.
   (d) Encrypts/decrypts data.
   (e) Builds DNS queries and responses.
   (f) Communicates with the local DNS server to send and receive queries and responses.

Our changes to the original Obfsproxy flow of operation in order to support the DNS-Morph flow of operation are:

1. Downstream connection was changed from TCP to UDP. The Obfsproxy server listens on port 53 UDP, like a standard DNS server.
2. Obfsproxy client initiates a UDP downstream connection on port 53 to the server. Using the DNS-Morph module, handshake data is encoded to Base32, chopped into fragments of length 20–50 characters,[5] and encapsulated into DNS queries of type A (address mapping records).
   These queries are sent to the Obfsproxy server using direct or indirect routes:
   (a) Direct: Obfsproxy client sends the queries directly to the Obfsproxy server's address.
   (b) Indirect: Obfsproxy client uses the recursive DNS property and sends the queries with a domain name registered by the Obfsproxy server operator. This way the DNS queries travel between different DNS servers until they reach the Obfsproxy server.

   Direct route usage might look anomalous [42, p. 28], hence, we recommend using the indirect route.
3. The total length in bytes of the handshake data is encoded inside the first DNS packet, so that the receiving side knows when to stop receiving data, and start processing it.
4. The Obfsproxy server buffers each DNS query it receives, until the total length is reached. In order to preserve resemblance to a standard DNS server, the Obfsproxy server sends the received queries to the local DNS server available on its side. When a response is received from the DNS server, the Obfsproxy server sends it back to the Obfsproxy client. When the buffered data forms a

[5] [44, p. 251] suggests that the average DNS query length is 36–59 bytes. [19] suggests that most of the DNS packets that were captured were of size 70–98 bytes (domain name length of 30–58 bytes).

complete handshake data, the DNS-Morph module decodes it from Base32, reassembles it, and forwards it to the obfuscation protocol layer for further processing.

5. After the Obfsproxy client finishes sending the handshake data, it should receive handshake data from the Obfsproxy server. As a result, the Obfsproxy client sends one additional dummy DNS query (contains no protocol handshake data) so that the Obfsproxy server can start sending back its handshake data as a response to this query. The dummy query is used by the Obfsproxy client to trigger transmission of the handshake data by the Obfsproxy server shaped as DNS responses.
6. When the Obfsproxy server sends back its protocol handshake data, it encodes it to Base32, chops it into fragments of length 20–50 bytes, and encapsulates them into CNAME DNS records. Then sends these records encapsulated together with A type records as responses to the dummy type A DNS queries which the Obfsproxy client sent before. Again, the first DNS response packet will include the total length of the Obfsproxy server sent handshake. The Obfsproxy client decapsulates the data from the DNS responses, buffers it, and keeps sending dummy queries until a complete handshake data is received. Then, the Obfsproxy client decodes the received handshake data back from Base32, reassembles it, and forwards it to the obfuscation layer.
7. When the protocol handshake is successfully completed, and tickets and/or session keys are created, both Obfsproxy sides switch from a UDP downstream connection to a TCP connection. The Obfsproxy server side launches a TCP downstream listener on a port exchanged out-of-band and waits for the Obfsproxy client to connect back to it. When the TCP connection happens, the connection between the Tor client and the bridge is fully functional. In case of a protocol handshake failure, both the Obfsproxy client and server close their downstream connection, and the Obfsproxy server continues to behave like a standard DNS server.

6 DNS-Morph Reliability

DNS-Morph depends on UDP, hence, we cannot rely on the network transport layer to provide reliable data transmission. Therefore, our implementation must guarantee arrival of the sent packets to the receiver's obfuscation layer in the same order that they were sent in, while adding a minimal delay. To achieve this, we use the methods explained below:

6.1 Received Packets Acknowledgments

The receiver sends an acknowledgment each time it receives a packet. If the sender does not get an acknowledgment, it means that the packet or its acknowledgment has failed to reach their destination, so the sender must resend the packet.

We divide the handshake phase into two parts according to the different roles played by the Obfsproxy client:

1. **Obfsproxy client sends handshake data:** each time it sends a DNS shaped packet, the Obfsproxy server should send back an acknowledgment in the shape of a valid DNS response to this query.
   The Obfsproxy client behaves similarly to the *selective repeat protocol* using a window size of 4 packets, with a few modifications due to the randomized DNS query identifier (ID). Obfsproxy client stores the sent packets in a list sorted by their sending order. The query ID serves as a search key to the list. Each time the client receives an acknowledgment it removes the matching packet from the list. If three acknowledgments are received, but none of them matches the first packet in the list, then this packet is resent with a new DNS query ID.
2. **Obfsproxy client receives handshake data:** as explained before, each dummy DNS query sent by the Obfsproxy client during this part serves two purposes, the first is to trigger the Obfsproxy server to send one additional handshake packet as a response to the dummy DNS query, and the second is to acknowledge receiving the previous handshake packet sent by the Obfsproxy server.
   During this part, the Obfsproxy client implements the *stop and wait protocol*: each time the Obfsproxy client wishes to receive handshake data, it sends a dummy DNS query and waits a while (discussed later) for the Obfsproxy server to send its handshake data as a response to this query. If nothing is received from the Obfsproxy server, then the Obfsproxy client resends the last DNS packet.[6] This procedure is repeated twice (the same packet is sent at most three times in total), and if no responses are received after that, the handshake process is terminated (in a failure).
   The time the Obfsproxy client waits for a handshake packet before resending the dummy DNS packet again is calculated by the known weighted average formula [39, p. 226] for round trip time measuring:

$$RTT = (\alpha \times RTT_{new}) + ((1 - \alpha) \times RTT_{old})$$

   RTT_{new} is the newly sampled RTT, RTT_{old} is the previously calculated RTT, and $\alpha = 1/8$ as recommended in [29].

6.2 Sorting Received Packets

DNS-Morph adds to each packet a 16-bit identity number, which is encrypted (as further explained in Sect. 6.3) and then encoded using Base32 to 4 ASCII characters in start of the packet's payload. This number is initialized to the length of the total sent data in the payload of the first DNS packet, and is increased by one each time a packet is sent, so newly sent packets have a bigger identity number than the previously sent ones.

Each time a packet is received, the receiver decodes and then decrypts the DNS-Morph identity bytes of its payload (see Sect. 7 for an example), checks

[6] Resending a failed DNS query with the same DNS identifier is a common practice among DNS resolvers.

if they form a legal identifying number, and if they do, the packet's data is buffered. When the whole handshake data is received, the receiver sorts the packets according to the identity number and reassembles them back to a real handshake data, passing it to the obfuscation layer for further processing.

In case of receiving two packets with the same identity number, the receiver does not buffer the second received packet but acknowledges the sender about it. This can happen for example because: 1. the first packet's acknowledgment failed to reach the sender, so the sender resends the packet. 2. an attacker is trying to send a packet that contains the same identity number in its payload.

6.3 DNS-Morph Identifiers' Encryption and Decryption

Encrypting the DNS-Morph identifier inside the encoded data of the DNS packets relies on an out-of-band shared password (similar to ScrambleSuit and obfs4) [32].

Our encryption uses 128-bit AES in counter mode. For each identifier, new key and initialization vector (IV) are created as the output of: HMAC-SHA-256(shared password, X || handshake data encoded in the current packet) [27], when $X = 0$ for Obfsproxy client encryption, $X = 1$ for Obfsproxy server encryption, and || means concatenation.

This encryption protects the identifier from being read without the knowledge of the shared key. This protects DNS-Morph from censoring attacks that identify the masked Tor connection by saving all DNS traffic between two connecting entities A and B, extracting the characters that include the identifier, decoding them, and checking whether:

1. The decoded characters form sequential numbers (identifiers).
2. The decoded characters of the first DNS packet form a number that matches the total length of the data in all the DNS packets.

6.4 DNS-Morph Multiple Sessions Support

In addition to encrypting the DNS-Morph identifiers, DNS-Morph also supports multiple clients handshake sessions by adding an encrypted 8-bit session ID to the packet's payload. This ID is randomly chosen by the DNS-Morph client and is encrypted together with the DNS-Morph identifiers using the same keys and IVs.

7 DNS-Morph Encoded Packets

We show how the encoded handshake data looks inside a DNS query and a DNS response packets. Consider, for example, handshake data that a ScrambleSuit client sends to the Obfsproxy server. After encoding this data using Base32 we receive encoded data of 450 characters. This data is chopped to fragments of length 20–50 characters, the first fragment of the chopped data looks like

this "ti3zuto4jrz5r22wsu4ar". To encapsulate this fragment inside a DNS query packet, we need to add encrypted DNS-Morph identity and session identity to it. The key and IV for the encryption of the DNS-Morph identity and session identity are created by:

$$\text{Key} \,||\, \text{IV} = \text{HMAC-SHA-256}(\text{shared password}, 0 \,||\, \text{ti3zuto4jrz5r22wsu4ar})$$

After computing the key and the IV, we use them to encrypt the DNS-Morph identity (which is 450 - the encoded handshake length in characters for the first packet) and the session identity, which is randomly generated by the client (95 for example).

$$\text{"enpin"} = Encode32(AES - CTR_{Key||IV}(450||95))$$

After computing the encrypted DNS-Morph ID and the session ID, they are encoded and then concatenated with "ti3zuto4jrz5r22wsu4ar", and with the string ".bridge.domain", to create the query:

"enpinti3zuto4jrz5r22wsu4ar.bridge.domain"

which is then encapsulated in a DNS packet and sent to the Tor bridge side. The bridge side, decodes the first 5 characters of the query, decrypts them using the rest part of the query not including ".bridge.domain", and the shared password. Then, it finds out that the length of the encoded handshake to receive for session 95 is 450 characters.

The response packet sent by the Obfsproxy server is a DNS response packet as generated by the local DNS server.

8 DNS-Morph: Security Analysis

We now discuss the security of our DNS-Morph design. Since DNS-Morph was built as an additional protection layer for random stream protocols, it inherits their security properties as a carrier protocol for them. The aim of this section is to prove that DNS-Morph does not harm the security properties of the encapsulated protocol but enhances them.

After DNS-Morph finishes the handshake phase the encapsulated protocol will still enjoy the same security properties as before, while its handshake security was enhanced.

We further discuss the security design of DNS-Morph assuming that the encapsulated protocols satisfy some conditions:

1. They must have a handshake phase.
2. They must be active probing resistant.
3. Their traffic (the handshake and the encrypted data exchange phases) must be indistinguishable from randomness.
4. They must provide data integrity. Providing data authenticity is recommended but is not a must.
5. The traffic exchanged during the handshake should be as short as possible without affecting the security properties of the protocol, as it affects the number of DNS packets exchanged using DNS-Morph (discussed more later).

8.1 Censor's DPI Capabilities

We now further elaborate on our assumptions concerning the censor's capabilities. We assume that the censor has the following DPI capabilities (which are used to detect DNS tunneling) installed in its ISP infrastructure, either on the DNS servers themselves, or on dedicated firewall/routers:

1. The DPI mechanism can sort DNS packets by their arrival times, build a pseudo-session out of them using a 4-tuple (UDP, IP_{client}, IP_{server}, Port_{server}), and attempt to detect the structure of "a session".
2. The DPI mechanism can search the DNS packets payload for any structural data such as counters, flags or words to signal "start", "stop", "resend", etc.
3. The DPI mechanism can consider the length of a DNS query/response and alert if DNS packets lengths regularities/irregularities are detected. For example, alerting about any domain name request longer than 52 characters [7].
4. The DPI mechanism can consider the number of DNS queries/responses for each 4-tuple or domain name, and alert if this number exceeds a certain threshold, or if the number of queries and responses differs significantly from what can be classified as benign behavior.
5. The DPI mechanism can detect irregular DNS packets sequences, e.g., a series of queries followed by a series of responses.
6. The DPI mechanism can use regular expressions or entropy estimation to detect suspicious DNS packets payloads.

8.2 DNS-Morph DPI Resistance

We added to DNS-Morph the following counter-measures that defeat the above DPI threat model, and make it exposure resistant:

1. DNS-Morph can send its packets in an arbitrary order and sort them on the receiver side, by decrypting the encrypted identifiers.
2. DNS-Morph encodes the encrypted payload inside DNS queries/responses, and concatenates encrypted packet and session IDs, thus, no structural data is available in the payload.
3. All the DNS queries/responses of DNS-Morph are of random size between 20–50 bytes, which is the size of a standard DNS query/response [19,44].
4. The number of DNS-Morph exchanged queries/responses depends on the encapsulated protocol. If we take ScrambleSuit for example, we can make some modifications, which will not affect the protocol security traits when DNS-Morph is added, but can reduce the total amount of DNS exchanged packets to as little as 26 DNS packets, an amount not so far from the number of DNS packets exchanged when visiting popular websites such as Google (10–17 packets), YouTube (11–13 packets), and Facebook (24–27 packets), which are among Alexa's top 10 global websites for the year 2021 [2].
5. The Obfsproxy server cannot send a DNS response without receiving a DNS query. A small delay can be added while the Obfsproxy client sends the DNS queries, which can eliminate DNS sequence abnormalities.

6. It is hard to use regular expressions in order to check for the "validity" of domain names without getting a lot of false positives, as many domain names nowadays can look random (or have a random looking part), due to domain fronting and load balancing services.
   We compare CloudFront domain names with DNS-Morph produced ones in Sect. 8.5.

8.3 Additional Attacks and Resistance

In addition to using DPI, a censor can tamper with DNS packets payload in many ways, such as:

1. The censor can change uppercase letters to lowercase, and vice versa. This change should not affect real DNS as it is case-insensitive, but might affect encoded and encapsulated data as changing cases changes bytes of the real data. As discussed in Sect. 9, our method of encoding and decoding data is resilient to this kind of changes.
2. The censor can change the DNS packet payload data or some of it (e.g., DNS poisoning). While deeming DNS poisoning out of this research scope, it is important to note that DNS poisoning in this context is a denial of service attack, i.e., it will cause a handshake failure, but will not reveal more information about the exchanged data than the information that was revealed by the original handshake.
3. The censor can inject DNS packets in two different ways:
   (a) The injected DNS packet contains random data. The receiver will not consider this packet as a part of the handshake as the possibility of this data to include an encrypted identifier using the out-of-band keys, is small.
   (b) The injected DNS packet is a replayed packet. This packet will be considered as a network reliability issue and will not be considered as part of the exchanged handshake.
4. The censor can observe the DNS responses sent back by the server, and try to launch an HTTP session to the IP written in these responses in order to examine if this IP points to a real server. Assuming that the IP returned by the DNS server is the IP of the DNS-Morph server, this server can run a simple HTTP server that displays or redirects to a random web-page.

8.4 Active Probing and Replay Attack Resistance

Our new Obfsproxy server acts like a real DNS server. Each time a DNS query is received, a real DNS response is sent back by a real DNS server. Assuming that the obfuscated protocol is active probing resistant, the probe will send DNS packets and receive real DNS responses. If the probe does not know the out-of-band shared password/key between the real Tor client and bridge, the obfuscated protocol on the Obfsproxy server side will reject the probe's handshake request, while the Obfsproxy server will continue to respond as a real DNS server.

The same also holds with respect to replay attacks assuming that the obfuscated protocol is replay-attack resistant. The obfuscated protocol will reject any replayed packet, while the Obfsproxy server continues to respond to it as a real DNS server.

8.5 Domain Names' Entropy

To show that the domain names used by DNS-Morph have the same entropy capacity as of regular domain names used in CDNs we have performed a simple test. We took 32 DNS packets directed at CloudFront services and 36 DNS packets obtained from a DNS-Morph handshake. We applied gzip and bzip2 to a file containing only the prefix of the domain names. The compression ratio using gzip of the CloudFront domain names was 66.5% whereas of the DNS-Morph domain names was 67.3%. In the case of bzip2, the ratios are 70% and 71.2%, respectively.

Given the fact that the handshake is done once per session, it is easy to see that attacks based on estimating the entropy in the domain names are unlikely to offer high precision due to the small difference in compression rates (suggesting somewhat related entropy capacity) [45].

9 DNS-Morph Design Considerations

9.1 Choice of DNS

We now discuss various trade-offs we made in DNS-Morph's design, and explain our design decisions: First, as mentioned before DNS is an essential Internet protocol, and its complete blocking is highly unlikely due to the high cost of doing so, namely, practically disconnecting from the internet. In addition, DNS supports a variety of query types which gives us more freedom in choosing the record type that can encapsulate our encoded data.

Following the selection of DNS, we had to use UDP (TCP sessions in DNS are usually used for zone transfers between DNS servers). This resulted in the need to add a reliability layer for DNS-Morph as loss of even a single packet of handshake data on any side causes a handshake failure (discussed more in Sect. 6).

In addition, the following actions were taken to enable DNS-Morph to mimic (as much as possible) an ordinary DNS communication:

We decided to add dummy DNS queries/responses which do not carry handshake data on both sides during the handshake phase, in order to prevent DNS protocol anomalies where the client sends multiple queries, and the server responds with multiple answers after all the requests were received, or the Obfsproxy server sending DNS responses without any sent queries.

9.2 Choice of Base32

We chose Base32 for encoding the handshake data which can include any byte value into characters that follow the domain name system rules [25]:

1. Domain name can include labels and the character ".".
2. Labels must start and end with a letter or a digit, and have as interior characters only letters, digits, and hyphens.
3. Letters can be any one of the 52 alphabetic characters "A–Z" in uppercase and "a–z" in lowercase.
4. Digits can be any one of the ten digits "0–9".

While we could have used Base64 [28] or Base58 [3], DNS is not case sensitive. In such encodings, a censor can rewrite every single DNS query to a lowercase one, which does not harm normal DNS requests, but breaks DNS-Morph. These factors narrow our encoding options to a base that has uppercase letters or lowercase letters, but not both. Base32 includes the alphabetic characters "A–Z", the digits "2–7" and the character "=", which is suitable for our uses after changing it to the digit "1". We also changed the uppercase letters that Base32 produces to lowercase letters inside the DNS queries/responses in order to make them look more consistent with regular DNS queries/responses. All the letters on the receiver side are converted back to uppercase before the Base32 decoding process, which makes our design resistant to attacks like the one described before.

We note that we can also use Base36, which includes the alphabetic characters "A–Z" and the digits "0–9". Using Base36 can protect DNS-Morph against a censor with the ability to spot the lack of the digits "0, 8, 9" from DNS-Morph packets, as these digits can appear in DNS labels, but are not included in our modified Base32. However, we decided to not use this for the ease of implementation. Future works may wish to explore that approach.

9.3 Query Types

In our choice of which DNS query/response types to use for handshake data encapsulation, we used the following guidelines:

1. The type of the DNS query/response must accept by their definition the amount and the character set of the data we want to encapsulate inside them. For example, the type A query can include characters and numbers (domain names), but the type A response includes 32-bit IPv4 addresses, thus, encapsulating encoded data that does not look like a valid IPv4 address inside a type A response is not possible.
2. Using certain DNS query/response packet types must not contradict the general DNS query/response types statistics over the Internet.
3. The types of the queries and responses must match.

To send the Obfsproxy client handshake data, we use type A queries, and receive type A responses from the Obfsproxy server. To send the Obfsproxy server

handshake data, we use CNAME and A DNS records, which are encapsulated together in the same DNS packet, and sent as DNS responses for type A DNS queries. We chose these types because they meet the aforementioned conditions, and in the same time are widely used on the Internet. The encapsulation of these records in DNS packets is done using the "dnslib" Python library [10], aiming to build DNS packets that look as real DNS client/server packets.

9.4 Recursive DNS

Finally, we decided to use the recursive DNS property and send the queries with a domain name registered by the Obfsproxy server operator, in order to protect against packets tracking. This also enables to bypass networks' firewall rules which forbid a direct DNS packet to be sent outside the network, besides the network's own DNS server.

10 Tests and Results

We have implemented our DNS-Morph design using Python, and tested it successfully with the ScrambleSuit PT. ScrambleSuit's handshake, is depicted in Fig. 3.

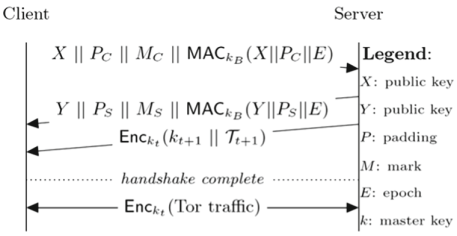

Fig. 3. ScrambleSuit's handshake.

During the handshake, a client and a server generate 4096-bit even private keys x and y, respectively, and the corresponding public keys $X = g^x (mod\ p)$, $Y = g^y (mod\ p)$. After exchanging the public keys, they agree on a shared private key $k_t = Y^x (mod\ p) = X^y (mod\ p)$. Then, using the shared key, the server sends back an encrypted ticket and a new private key (used by a shortened handshake version in future handshakes). An out-of-band shared key k_B is used to calculate the MACs and the marks (used to facilitate the localization of the MACs).

10.1 Test Setup

For our experiments, we set up a Tor client in a censoring environment, which is described in Sect. 10.2. This client connects to a Tor bridge residing at a

university located in Israel. This Bridge is connected to the Internet using an ISP **not known** for any censoring activities. The two sides (the client's side and the bridge's side) had the Obfsproxy software with the DNS-Morph component, and tried to establish Tor's first connection between them. We ran 30 connection attempts in each test, measuring rates of successful handshakes, their timing, and their bandwidth.

Between connection attempts, all the temporary files and session states generated by previous attempts (e.g., session tickets) were deleted to ensure that ScrambleSuit generates everything and behaves the same all the time.
With this setup, we performed three different experiments:

1. "Original Design" tests, to test the original Obfsproxy without DNS-Morph. This Obfsproxy uses TCP only.
2. "DNS-Morph Direct" tests, to test the Obfsproxy including DNS-Morph. DNS queries were directly sent from the Obfsproxy client to the Obfsproxy server address.
3. "DNS-Morph Indirect" tests, to test the Obfsproxy, including DNS-Morph. DNS queries were sent indirectly to the Obfsproxy server address by other DNS servers (using recursive DNS queries).

For our "DNS-Morph" tests, we also did one additional type of tests: we decreased the maximal size of the random padding added to the protocol (see Fig. 3). ScrambleSuit picks a random padding length of 0–1308 bytes to change the protocol's signature. Due to the use of DNS queries, such a protection is less needed. Thus, we tested the system when the maximal size of the random padding is at most 100 bytes.

Decreasing the padding length range reduces the **maximal** total number of DNS packets (queries and responses) per DNS-Morph handshake from 458 packets to 96 packets and the **average** total number from 262 packets to 81 packets (the **minimal** total number stays 26 packets). By doing so, we wanted to check if the handshake time improves, and if as a result of DNS servers offloading (DNS-Morph indirect), the handshake success rates will be higher compared to the first type tests, as DNS servers will be more willing to serve us.

10.2 Client's Testing Environment

This environment includes an ISP in Israel, which is **well known** for censoring Internet browsing for its customers (who seek this type of censorship). Content censored by this ISP can be pornography, violence, live casting, and videos. While trying to connect to websites like Google, YouTube, and Facebook, this ISP asked us to install CA certificates on the client's machine. Refusing to do so resulted in blocking client's access to these websites. Other websites were simply blocked without even giving the client an option to install certificates (i.e., Yahoo, Bing, and The Tor Project website).

It is worth mentioning that the certificates needed to be installed by this ISP are self-signed certificates issued to "Netspark" [21], a company providing real-time browsing data inspection and web content filtering services.

Connecting to Tor in this blocking ISP: when trying to connect in the default method, we encountered failures during different phases such as connecting to the directory servers, loading relay descriptors, or connecting to the entry relay. Trying to connect using obfs3 or FTE always failed.

Choosing obfs4 succeeded sometimes and on the other times it failed. Even on successful attempts a lot of the bridges were blocked.

In conclusion, we suspect that this ISP continuously collects information about Tor bridges and blocks them.

Results: We performed numerous connection tests during a period of 2 months (January–February 2018). Table 1 summarizes the results of the tests done on the 10th of February 2018, as a typical example for these results. We chose to present the results of only one day as all the other days produced similar results.

Despite the fact that the Tor client was connecting to the Tor bridge using a censoring ISP, this ISP could not block its connection, not of the original ScrambleSuit, nor of our DNS-Morph version. This indicates that DNS-Morph did not harm the original security properties of ScrambleSuit.

The 100% success rates in all the experiments can be explained by the fact that the client and the server are geographically close.

Table 1. Times and success rates of handshakes.

	Time (seconds)				Success rate	Bandwidth (bytes)
	Minimum	Maximum	Average	Median		
Original ScrambleSuit	0.044	0.101	0.087	0.091	100%	4042
Original ScrambleSuit with DNS-Morph Direct	0.891	1.813	1.113	0.953	100%	22140
Original ScrambleSuit with DNS-Morph Indirect	1.844	2.391	2.171	2.189	100%	24350
Shortened ScrambleSuit with DNS-Morph Direct	0.438	0.672	0.542	0.547	100%	6528
Shortened ScrambleSuit with DNS-Morph Indirect	0.594	0.798	0.708	0.719	100%	6621

ScrambleSuit - Original: ScrambleSuit with the original padding (0–1308 bytes).
ScrambleSuit - Shortened: ScrambleSuit with a shortened padding (0–100 bytes).

10.3 Deep Packet Inspection Tools

To evaluate DNS-Morph resistance against DPI we chose two open source tools: nDPI [20] version 2.2.0, and Libprotoident [16] version 2.0.12. We captured the packets transmitted between the Tor client and the Tor bridge from the previous tests (Sect. 10) using Wireshark and ran the two tools to analyze them.

Results: The original Scramblesuit connections were analyzed as "Unknown TCP" protocol packets by both tools, nDPI and Libprotoident. The DNS-Morph connections were analyzed by nDPI as ordinary DNS packets for the handshake phase, and SSL packets for the encrypted data exchange after the handshake was done. Libprotoident on the other hand analyzed the DNS-Morph connections as ordinary DNS packets for the handshake phase, and "Unknown TCP" protocol packets for the encrypted data exchange after the handshake.

11 Summary

In this work we described DNS-Morph, a method to hide PT's handshake communication in a series of DNS queries and replies. We implemented the system and checked that it successfully establishes a Tor connection between a Tor client and a Tor bridge.

The use of DNS offers several layers of security for this process: DNS blocking (or even strong manipulation) comes at a huge price for the censor, DPI attacks on DNS are harder to implement, and DNS enjoys an inherent resilience to blocking attempts due to the nature of recursive DNS queries.

In addition, we have added counter-measures designed to defeat UDP sessions' tracking mechanisms which, we believe still do not exist today in commercial DPI products.

While we have tested the implementation using the ScrambleSuit handshake, it is easy to see that this methodology could work with any PT that satisfies the conditions stated in Sect. 8. As already stated, after the handshake, one can transform the remainder of the Tor communication to a white-listed protocol using tools like FTE or meek to maintain white-listed behavior (while enjoying the security against active probing offered by the protected handshake).

11.1 Future Works

This research can be further developed to include more interesting topics and answer currently open questions:

- We are aware of the DNS poisoning some countries conduct not only to block sensitive content but also to promote local websites [37]. An interesting approach is to examine what exactly is done by the censor while performing DNS poisoning, and whether our DNS-Morph can finish the handshake despite these DNS poisoning attacks.
- In order to better mimic the behavior of a DNS client (and fulfill one more of the requirements for building a successful mimicking protocol [43]), a service can be installed on the client machine to collect data about the user's browsing behavior, such as: DNS packets timing, size, DNS client behavior if its query was not answered, etc. When the DNS-Morph client is started, it can analyze the collected data to better adapt the specific user and machine behavior.
- More techniques can be applied to DNS-Morph, to improve its resistance against DPIs, DNS tunneling detectors, and connection tracking:

- Adding more DNS record types than the currently used ones (A and CNAME).
- Use an encoding scheme which mimics as best as possible real domain names, instead of Base32.
- Use multiple DNS servers which collaborate, each accepting some of the queries of the handshake (and sending back the information). This way, the tracing of the DNS queries becomes harder (as the number of queries per DNS server is reduced).

References

1. Active Probing. https://www.cs.princeton.edu/~rensafi/projects/active-probing/. Accessed 1 Aug 2017
2. Alexa: The top 500 sites on the web. https://www.alexa.com/topsites. Accessed 24 Feb 2020
3. Base58. https://en.wikipedia.org/wiki/Base58. Accessed 10 Aug 2017
4. Blocking-resistant communication through domain fronting. https://www.bamsoftware.com/papers/fronting/. Accessed 15 Nov 2017
5. China Tells Carriers to Block Access to Personal VPNs by February. https://www.bloomberg.com/news/articles/2017-07-10/china-is-said-to-order-carriers-to-bar-personal-vpns-by-february. Accessed 16 Aug 2017
6. DNS-Morph source code, Github, note: Please contact the "Corresponding Author" to get access to the Github repository
7. DNS part 2: visualization. http://armatum.com/blog/2009/dns-part-ii/. Accessed 21 Aug 2017
8. Dns2tcp. https://www.aldeid.com/wiki/Dns2tcp. Accessed 20 Aug 2017
9. Dnscat. https://wiki.skullsecurity.org/Dnscat. Accessed 20 Aug 2017
10. dnslib: A library to encode/decode DNS wire-format packets. https://pypi.python.org/pypi/dnslib. Accessed 6 Oct 2017
11. FTE Transport Evaluation. https://trac.torproject.org/projects/tor/wiki/doc/PluggableTransports/FteEvaluation. Accessed 25 Sept 2016
12. Full List of Pluggable Transports. https://trac.torproject.org/projects/tor/wiki/doc/PluggableTransports/list. Accessed 20 Aug 2017
13. Improved circuit-creation key exchange. https://gitweb.torproject.org/torspec.git/tree/proposals/216-ntor-handshake.txt. Accessed 10 Sept 2016
14. iodine. http://code.kryo.se/iodine/. Accessed 20 Aug 2017
15. Knock Knock Knockin' on Bridges' Doors. https://blog.torproject.org/blog/knock-knock-knockin-bridges-doors. Accessed 7 Sept 2016
16. Libprotoident. https://research.wand.net.nz/software/libprotoident.php. Accessed 25 Aug 2018
17. Meek. https://trac.torproject.org/projects/tor/wiki/doc/meek. Accessed 22 Aug 2016
18. Meek Transport Evaluation. https://trac.torproject.org/projects/tor/wiki/doc/PluggableTransports/MeekEvaluation. Accessed 25 Sept 2016
19. More fun with DNS packet captures. https://www.coverfire.com/archives/2008/07/28/more-fun-with-dns-packet-captures/. Accessed 3 June 2017
20. nDPI - Open and Extensible LGPLv3 Deep Packet Inspection Library. https://www.ntop.org/products/deep-packet-inspection/ndpi/. Accessed 25 Nov 2017

21. Netspark. http://netspark.com/. Accessed 25 Nov 2017
22. Obfs3 Protocol Specification. https://github.com/NullHypothesis/obfsproxy/blob/master/doc/obfs3/obfs3-protocol-spec.txt. Accessed 20 Sept 2016
23. Obfs4 Protocol Specification. https://github.com/Yawning/obfs4/blob/master/doc/obfs4-spec.txt. Accessed 22 Aug 2016
24. ObfsProxy Source Code - Python. https://gitweb.torproject.org/pluggable-transports/obfsproxy.git. Accessed 25 Aug 2016
25. RFC 1123: Requirements for Internet Hosts - Application and Support. https://tools.ietf.org/html/rfc1123. Accessed 3 June 2017
26. RFC 1928: SOCKS Protocol. https://www.ietf.org/rfc/rfc1928.txt. Accessed 22 Aug 2016
27. RFC 2104: HMAC - Keyed-Hashing for Message Authentication. https://www.ietf.org/rfc/rfc2104.txt. Accessed 20 Nov 2017
28. RFC 3548: The Base16, Base32, and Base64 Data Encodings. https://tools.ietf.org/html/rfc3548.html. Accessed 6 Feb 2017
29. RFC 6298: Computing TCP's Retransmission Timer. https://tools.ietf.org/html/rfc6298. Accessed 10 June 2017
30. Skype: online text message and video chat services. https://www.skype.com. Accessed 25 July 2017
31. The UniformDH scheme - Ian Goldberg. https://lists.torproject.org/pipermail/tor-dev/2012-December/004245.html. Accessed 20 Sept 2016
32. Tor bridges database. https://bridges.torproject.org/. Accessed 30 Oct 2017
33. Tor Client Software. https://www.torproject.org/download/download. Accessed 10 Sept 2016
34. Tor: Pluggable Transports Specification. https://gitweb.torproject.org/torspec.git/tree/pt-spec.txt. Accessed 22 Aug 2016
35. Tor Website. https://www.torproject.org/. Accessed 25 Aug 2016
36. WhatsApp Messenger. https://www.whatsapp.com/. Accessed 25 July 2017
37. Anonymous: Towards a comprehensive picture of the great firewall's DNS censorship. In: 4th USENIX Workshop on Free and Open Communications on the Internet (FOCI 14), San Diego, CA, pp. 445–458. USENIX Association (2014). https://www.usenix.org/conference/foci14/workshop-program/presentation/anonymous
38. Bernstein, D.J., Hamburg, M., Krasnova, A., Lange, T.: Elligator: Elliptic-curve points indistinguishable from uniform random strings. Cryptology ePrint Archive, Report 2013/325 (2013). http://eprint.iacr.org/2013/325
39. Comer, D.E.: Internetworking with TCP/IP, Volume 1: Principles, Protocols, and Architectures, 4th edn. Prentice Hall PTR, Upper Saddle River (2000)
40. Dyer, K.P., Coull, S.E., Ristenpart, T., Shrimpton, T.: Protocol misidentification made easy with format-transforming encryption. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer 38; Communications Security, CCS 2013, pp. 61–72. ACM, New York (2013). https://doi.org/10.1145/2508859.2516657
41. Ensafi, R., Fifield, D., Winter, P., Feamster, N., Weaver, N., Paxson, V.: Examining how the great firewall discovers hidden circumvention servers. In: Proceedings of the 2015 ACM Conference on Internet Measurement Conference, IMC 2015, pp. 445–458. ACM, New York (2015). https://doi.org/10.1145/2815675.2815690
42. Fry, C., Nystrom, M.: Security Monitoring: Proven Methods for Incident Detection on Enterprise Networks. O'Reilly Media (2009). https://books.google.co.il/books?id=vJYCZFTdfd0C

43. Houmansadr, A., Brubaker, C., Shmatikov, V.: The parrot is dead: Observing unobservable network communications. In: 2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, 19–22 May 2013, pp. 65–79 (2013). https://doi.org/10.1109/SP.2013.14
44. Manaf, A., Zeki, A., Zamani, M., Chuprat, S., El-Qawasmeh, E.: Informatics Engineering and Information Science. Springer, Heidelberg (2011). https://books.google.it/books?id=zWarCAAAQBAJ
45. Paxson, V., et al.: Practical comprehensive bounds on surreptitious communication over DNS. In: Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13), Washington, D.C., pp. 17–32. USENIX (2013). https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/paxson
46. Winter, P., Crandall, J.R.: The great firewall of China: how it blocks Tor and why it is hard to pinpoint. USENIX Login **37**(6), 42–50 (2012)
47. Winter, P., Pulls, T., Fuß, J.: Scramblesuit: a polymorphic network protocol to circumvent censorship. In: Proceedings of the 12th Annual ACM Workshop on Privacy in the Electronic Society, WPES 2013, Berlin, Germany, 4 November 2013, pp. 213–224 (2013). https://doi.org/10.1145/2517840.2517856