ScrambleSuit: A Polymorphic Network Protocol to Circumvent Censorship

Philipp Winter
Karlstad University
Karlstad, Sweden
philipp.winter@kau.se

Tobias Pulls
Karlstad University
Karlstad, Sweden
tobias.pulls@kau.se

Juergen Fuss
Upper Austria University of Applied Sciences
Hagenberg, Austria
juergen.fuss@fh-hagenberg.at

ABSTRACT

Deep packet inspection technology became a cornerstone of Internet censorship by facilitating cheap and effective filtering of what censors consider undesired information. Moreover, filtering is not limited to simple pattern matching but makes use of sophisticated techniques such as active probing and protocol classification to block access to popular circumvention tools such as Tor.

In this paper, we propose ScrambleSuit; a thin protocol layer above TCP whose purpose is to obfuscate the transported application data. By using morphing techniques and a secret exchanged out-of-band, we show that ScrambleSuit can defend against active probing and other fingerprinting techniques such as protocol classification and regular expressions.

We finally demonstrate that our prototype exhibits little overhead and enables effective and lightweight obfuscation for application layer protocols.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols; C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*; K.4.1 [**Computers And Society**]: Public Policy Issues—*Privacy, Transborder data flow, Use/abuse of power*

Keywords

Tor; bridge; pluggable transport; active probing; traffic analysis; censorship; circumvention

1. INTRODUCTION

We consider deep packet inspection (DPI) harmful. While originally meant to detect attack signatures in packet payload, it is ineffective in practice due to the ease of evasion [1, 2, 3]. At the same time, DPI technology is increasingly used by censoring countries to filter the free flow of information or violate network neutrality [4]. We argue that what makes DPI particularly harmful is the *asymmetry of blocking effectiveness*, i.e., it is hard to stop motivated and skilled network intruders but very easy to censor ordinary user's Internet access. DPI technology ultimately fails to protect critical targets but succeeds in filtering the information flow of entire countries.

Numerous well-documented cases illustrate how DPI technology is used by censoring countries. Amongst others, China is using it to filter HTTP [5] and rewrite DNS responses [6]. Iran is known to use DPI technology to conduct surveillance [7]. In Syria, DPI technology is used for the same purpose [8]. Even more worrying, TLS interception proxies, an increasingly common feature of DPI boxes, are used to transparently decrypt and inspect TLS sessions which effectively breaks the confidentiality provided by TLS.

The rise of Internet censorship led to the creation of numerous circumvention tools which engage in a rapidly developing arms race with the maintainers of censorship systems. Of particular interest to censoring countries is the Tor network [9]. While originally designed as a low-latency anonymity network, it turned out to be an effective tool to circumvent censorship as well. Tor's growing success as a circumvention tool did not remain unnoticed, though. Tor is or was documented to be blocked in many countries including Iran [10], China [11], and Ethiopia [12], just to name a few. We argue that many circumvention tools—Tor included—suffer from two shortcomings which can easily be exploited by censors.

First and most importantly, they are vulnerable to *active probing* as pioneered by the Great Firewall of China (GFW) [11]: the GFW is able to block Tor by first looking for potential Tor connections based on the TLS client cipher list. If such a signature is found on the wire, the GFW reconnects to the suspected Tor bridge and tries to "speak" the Tor protocol with it. If this succeeds, the GFW blacklists the respective bridge. Active probing is not only used to discover Tor but—as we will discuss—also VPNs [13] and obfs2 [14], which is a censorship-resistant protocol. The relevance of active probing attacks is emphasised by the work of Durumeric et al. [15]. By conducting fast Internet-wide scanning, the authors were able to find approximately 80% of all active bridges at the time. From a censor's point of view, active probing is a promising strategy which greatly reduces collateral damage caused by inaccurate signatures. The attack is also non-trivial to defend against because censors can easily emulate real computer users.

WPES'13, November 4, 2013, Berlin, Germany.

http://dx.doi.org/10.1145/2517840.2517856.

Second, circumvention tools tend to exhibit a specific "flow signature" which typically remains static and is the same for all clients and servers speaking the protocol. An example is Tor's characteristic 586-byte signature (see §4.3.1). If a censor manages to deploy high-accuracy classifiers trained to recognise these very flow signatures, the respective protocol could easily be spotted and blocked. Most circumvention tools are not polymorphic which makes them unable to survive such filters.

In this work, we present ScrambleSuit; a blocking-resistant transport protocol which tackles the two above mentioned problems. ScrambleSuit defines a thin protocol layer on top of TCP which provides lightweight obfuscation for the transported application layer protocol. As shown in Figure 1, ScrambleSuit is independent of its application layer protocol and works with any application supporting SOCKS. As a result, we envision ScrambleSuit to be used by, amongst other protocols, Tor and VPN to tackle the GFW's most recent censorship upgrades. In particular, ScrambleSuit exhibits the following four features:

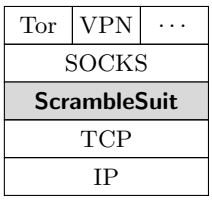


Figure 1: ScrambleSuit's protocol stack.

Pseudo-random payload: To an observer, ScrambleSuit's entire traffic is computationally indistinguishable from randomness. As a result, there are no predictable patterns which would otherwise form suitable DPI fingerprints. This renders regular expressions for the purpose of identifying ScrambleSuit useless.

Polymorphic: Despite the pseudo-random traffic, a censor could still block our protocol based on flow characteristics such as the packet length distribution. ScrambleSuit is, however, able to change its shape to make it harder for classifiers to exploit flow characteristics.

Shared secret: We defend against active probing by making use of a secret which is shared between client and server and exchanged out-of-band. A server only talks to clients if knowledge of the secret is proven by the client.

Usable: We seek to maximise ScrambleSuit's usability. Our protocol integrates in Tor's existing ecosystem. Furthermore, the moderate protocol overhead discussed in §5 facilitates comfortable web surfing.

Blocking-resistant protocols can be split into two groups. While the first group strives to *mimic typically whitelisted protocols* such as HTTP [16], Skype [17, 18] and email [19], the second group aims to *look like randomness* [20, 21, 22]. Randomised protocols have the shortcoming of not being able to survive a whitelisting censor[1]. Nevertheless, we decided in favour of randomising because mimicking comes at the cost of high overhead and we consider whitelisting on a nation scale—at least for most countries—unlikely even though it is often done in corporate networks and some countries appear to be experimenting with the concept [24, 25]. While at some point, our protocol might indeed become victim of a country's whitelisting policy, we believe that our approach provides a fast alternative in many censoring countries unwilling to go that far. So instead of maximising obfuscation while maintaining an acceptable level of usability, we seek to *maximise usability* while keeping an *acceptable level of obfuscation*.

We finally point out that unblockable network protocols do not exist. After all, censors could always "pull the plug" as it was already done in Egypt [26] and Syria [27]. By proposing ScrambleSuit, we do not claim to end the arms race in our favour but rather to raise the bar once again.

The contributions of this paper are as follows.

1. We propose ScrambleSuit, a blocking-resistant transport protocol.
2. We present two authentication mechanisms based on shared secrets and polymorphism as a practical defence against active probing and protocol classifiers.
3. We implement and evaluate a fully functional prototype of our protocol which is publicly available under a free license.

The remainder of this paper is structured as follows. In §2 we discuss related work which is followed by an architectural overview in §3. Then, §4 discusses ScrambleSuit's design in detail. The protocol is evaluated in §5 and the results are discussed in §6. We finally conclude the paper in §7.

2. RELATED WORK

Protocol Identification: Hjelmvik and John investigated to which extent supposedly obfuscated protocols such as Skype, BitTorrent's message stream encryption, and Spotify can be identified [28]. Based on their findings, Hjelmvik and John suggest evasion techniques for protocol designers which should make it harder to identify obfuscated protocols. Some of our design decisions were motivated by their suggestions. Similar to that, Wiley proposed a framework to dynamically classify network protocols based on Bayesian models [29]. This is an important first step towards the ability to compare and evaluate blocking-resistant transport protocols.

Protocol Obfuscation: The Tor project developed a blocking-resistant protocol called obfs2 [20]. The protocol implements an obfuscation layer on top of TCP and transports Tor traffic. A passive Man-in-the-Middle (MitM) however can decrypt obfs2 traffic. The successor, obfs3 [21], uses a customised Diffie-Hellman handshake to solve this problem. However, both, obfs2 and obfs3 can be actively probed and do not disguise flow properties. In fact, we found that the GFW is already blocking obfs2 bridges by actively probing them [14]. Later in this paper, we extend obfs3's handshake to be resistant against active probing. Wiley's Dust protocol [22] compares to obfs2 and obfs3 in that Dust payload looks like random data but the key exchange is handled out-of-band. Dust also employs packet padding to camouflage packet lengths. But unlike ScrambleSuit, Dust does not consider inter-arrival times and is unable to change its flow signature.

Weinberg et al. presented StegoTorus [16], an obfuscation framework similar to Tor's obfsproxy [30]. StegoTorus can

[1]The same fate can meet mimicked protocols if the censor decides to block the cover protocol. Oman is documented to block Skype [23] which would render Skype-based circumvention protocols useless.

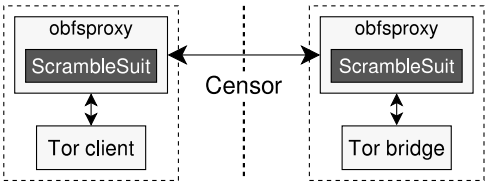


Figure 2: ScrambleSuit is a module for obfsproxy which provides a SOCKS interface for local applications. The network traffic between two obfsproxy instances is disguised by ScrambleSuit.

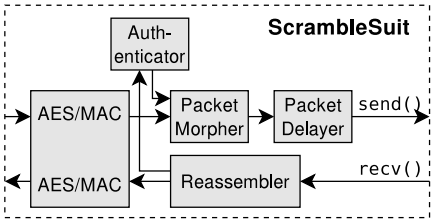


Figure 3: Internally, ScrambleSuit handles authenticated encryption of application data, client authentication as well as flow reshaping using a packet morpher and delayer.

complicate protocol identification on the application layer as well as on the transport layer. Tor connections can be multiplexed over multiple TCP connections and the application layer is camouflaged by mimicking a cover protocol such as HTTP. SkypeMorph, as presented by Moghaddam et al. [17] disguises Tor traffic as Skype video traffic. Similar work was done by Houmansadr et al. with FreeWave which is able to carry network traffic over VoIP connections [18]. Another attempt to tunnel otherwise censored network traffic over a "whitelisted" protocol was done by Zhou et al. by proposing SWEET [19] which uses email as its cover medium. With CensorSpoofer, Wang et al. propose to decouple the upstream from the downstream channel for censorship resistance [31]. The low-bandwidth upstream channel uses steganography whereas the high-bandwidth downstream channel makes use of IP spoofing to fool censors.

Houmansadr et al. claim that mimicking a cover protocol rather than using it as an actual medium is a flawed approach [32]. The authors showed numerous aspects in which StegoTorus, SkypeMorph and CensorSpoofer differ from their respective cover medium.

Lincoln et al. proposed DEFIANCE [33]: an architecture to protect Tor bridges from being probed and their respective descriptors[2] from being harvested by crawlers. The authors accomplish these goals by developing a novel rendezvous protocol as well as a technique called address-change signaling.

Port-knocking-based Authentication: Vasserman et al. proposed SilentKnock: an undetectable authentication system based on port knocking [34]. Clients can authenticate themselves towards a server with a single TCP SYN segment. If the authentication does not succeed, the server does not reply with a SYN/ACK segment, thus appearing offline.

Smits et al. adapted SilentKnock to better work with Tor bridges [35]. The result is called BridgeSPA. Like SilentKnock, BridgeSPA does not protect against connection hijacking and faces practical problems such as the inability to cope with NAT and the dependence on Linux kernels. While ScrambleSuit can not hide its "aliveness", it is not hindered by NAT, renders connection hijacking useless, and works on every platform having a Python interpreter.

3. ARCHITECTURAL OVERVIEW

ScrambleSuit is a module for obfsproxy which is an obfuscation framework developed by the Tor project [30]. As long as obfsproxy is running on the censored client as well as on the server, all network traffic in between both communication points can be obfuscated as dictated by the respective obfuscation modules. Figure 2 illustrates that obfsproxy acts as a proxy between the Tor client and the Tor bridge. While specifically designed for Tor, obfsproxy can be used by any application as long as it supports the SOCKS protocol.

Internally, ScrambleSuit is composed of several components which are depicted in Figure 3. Outgoing network data is first encrypted and then padded by the packet morpher. Before these packets are then sent over the wire, the packet delayer uses small artificial delays to disguise inter-arrival times. Finally, incoming network data is first reassembled to complete ScrambleSuit protocol messages and, after decryption, finally passed on to the local application.

3.1 Threat Model

Our adversary is a *nation-state censor* who desires to block unwanted network protocols and services which would otherwise allow users within the censoring regime the retrieval of unfiltered information or to evade the national filtering system. The censor is making use of payload analysis, flow analysis as well as active probing to identify and then block undesired protocols.

Furthermore, the censor has full *active and passive* control over the national network. The censor can passively monitor all traffic entering and leaving its networks in line rate. We further expect the censor to actively tamper with traffic; namely to inject, drop, and modify traffic as well as hijack TCP sessions.

The censor can also select a subset of suspicious traffic for further inspection on the slow path[3]. This could involve *active probing* as done by the GFW in order to block the Tor network [11]. We model our censor to also conduct active MitM attacks. While we believe that passive analysis and active probing are significantly easier to deploy, there is evidence that censors are starting to—or at least have the ability to—conduct active MitM attacks as well [36].

Our adversary is also training and deploying *statistical classifiers* to identify and block protocols. While computationally expensive, it would be imaginable that a censor uses this strategy at least on the slow path and perhaps even on the fast path when using inexpensive flow features and efficient classifiers.

[2]A bridge descriptor is essentially a tuple containing the bridge's IP address, port and fingerprint.

[3]We define the *slow path* as the minute analysis of a small traffic subset as opposed to the *fast path* which covers the majority of all network traffic and, as a result, has to be processed quickly.

3.1.1 Adversary Limitations

We expect the censor to be subject to economical constraints. In particular, we assume that the censor is not using a whitelisting approach meaning that only well-defined protocols pass the national filter. Whitelisting implies significant *over-blocking* and we expect this approach to collide with the censor's economical incentives. We also expect the censor to not block protocols when there is only weak evidence for the protocol being blacklisted. This is a direct consequence of avoiding over-blocking to minimise collateral damage.

Finally, we assume that the censor does not have access to or can otherwise influence censored users' computers. We believe that such a scenario is likely to occur in corporate networks but not on a national scale.

4. PROTOCOL DESIGN

This section will discuss ScrambleSuit's defence against active probing, its encryption and header format as well as how we achieve polymorphism.

4.1 Thwarting Active Probing

We defend against active probing by proposing two mutual authentication mechanisms which rely on a secret which is shared *out-of-band*. A ScrambleSuit connection can only be established if both parties can prove knowledge of this very secret. While one authentication mechanism (see §4.1.4) is designed to work well in Tor's ecosystem, the other mechanism (see §4.1.2) provides additional security and efficiency if ScrambleSuit is used by other application protocols such as a VPN.

With respect to Tor, there *already exists* an out-of-band communication channel which is used to distribute bridge descriptors to censored users. Naturally, we make use of this channel. If, however, ScrambleSuit is used to tunnel protocols other than Tor, users have to handle out-of-band communication themselves.

4.1.1 Proof-of-Work (Again) Proves Not to Work

Before deciding in favour of using a secret exchanged out-of-band, we investigated the suitability of client puzzles. Puzzles—a variant of proof-of-work schemes—could be used by a server to time-lock a secret. This secret can then only be unlocked by clients by spending a moderate amount of computational resources on the problem. One particular puzzle construction, namely time-lock puzzles as proposed by Rivest et al. [37], provides appealing properties such as deterministic unlocking time, asymmetric work load and inherently sequential computation which means that adversaries in the possession of highly parallel architectures have no significant advantage over a client with a single CPU.

While *a single* client puzzle can not be solved in parallel, a censor is able to solve *multiple* puzzles in parallel by assigning one puzzle to every available CPU core. This is problematic because our threat model includes censors with powerful and parallel architectures. We estimated the Tor network's bridge churn rate and found that the rate of new bridge IP addresses joining the network (i.e., the amount of puzzles to solve) is not high enough to be able to increase a well-equipped censor's work load beyond the point of becoming *impractical*; at least not without becoming impractical for *clients as well*. This balancing problem is analogous to why proof-of-work schemes are also believed to be unpractical for the spam problem [38].

In summary, proof-of-work schemes would not require a shared secret but we believe that this small usability improvement would come at the cost of greatly reduced censorship resistance. A censor in the possession of powerful computational resources would certainly be slowed down but could ultimately not be stopped. Active probing would simply become a matter of investing more resources.

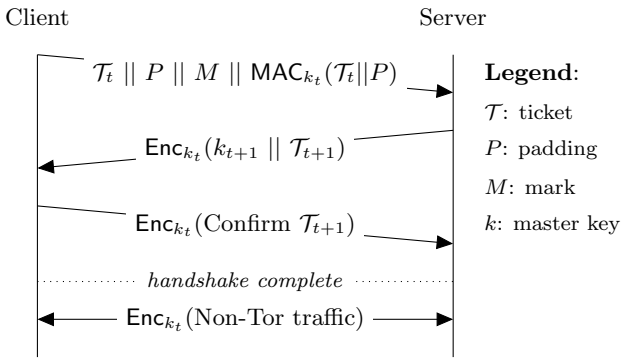


Figure 4: The client redeems a valid session ticket $\mathcal{T}_t$ containing the master key k_t. The server responds by issuing a new ticket $\mathcal{T}_{t+1}$ for future use. After the client confirmed the receipt, both parties then exchange application data.

4.1.2 Authentication Using Session Tickets

We now present the first of our two authentication mechanisms. We envision it to be used by insecure protocols which, unlike Tor, can not protect against active MitM attacks. A client can authenticate herself towards a ScrambleSuit server by redeeming a *session ticket*. A session ticket needs to be obtained only once out-of-band. Subsequent connections are then bootstrapped using tickets issued by the server during the respective previous connection. A real world analogy would be Alice redeeming a ticket in order to gain access to a football stadium. Upon entering the stadium (i.e., successful authentication), the guards give Alice a new ticket so that she is able to return for the next match. The same procedure then takes place for the next match. Session tickets are standardised in RFC 5077 [39] and part of TLS since version 1.0. We employ only a subset of the standard since we do not need its full functionality.

The basic idea is illustrated in Figure 4. The three-way handshake starts with the client *redeeming* a session ticket. The server then responds by *issuing* a new ticket. Finally, the client *confirms* the receipt of the newly issued ticket.

ScrambleSuit servers issue new session tickets $\mathcal{T}_{t+1}$ which embed a future shared master key k_{t+1} and an issue date d indicating the ticket's creation time. Session tickets are symmetrically encrypted and authenticated with secret keys k_S[4] only known to the server, i.e., $\mathcal{T}_{t+1} = \mathsf{Enc}_{k_S}(k_{t+1} \mid\mid d)$. As a result, a ticket is opaque to the client. Note that together with the ticket, a client also has to learn the master key k_{t+1} in order to be able to derive the same session keys as the

[4]For simplicity, we refer to these keys as just k_S while they are in fact two symmetric keys: one for encryption and one for authentication.

server; so clients always obtain the tuple $(k_{t+1} \mid\mid \mathcal{T}_{t+1})$ from servers.

Session tickets have the advantage that the server *does not have to keep track* of issued tickets. Instead, the server's state is outsourced and stored by clients. This reduces a server's load. To verify whether a ticket is still valid, servers simply decrypt the ticket and check the issue date d embedded in the ticket.

Whenever a client successfully connects to a ScrambleSuit server, the server issues a new ticket concatenated to the according master key $(k_{t+1} \mid\mid \mathcal{T}_{t+1})$ for the client. Recall that the ticket is encrypted and opaque to the client which is the reason why the master key is prepended. This tuple is then placed in a special ScrambleSuit control message (see §4.2) which is sent immediately after a ticket was successfully redeemed. Coming back to the real world example, Alice now has her new ticket which makes it possible for her to return for the next match.

Finally, the client confirms receipt of the new ticket by sending a dedicated confirmation message to the server. After that, the three-way ticket handshake is completed and application data can be exchanged.

Pseudo-random padding: A censor could now conduct traffic analysis by looking for TCP connections which always begin with the client sending $|\mathcal{T}|$ bytes to the server. To obfuscate the ticket's length, we introduce random padding P and authenticate the ticket $\mathcal{T}_t$ as well as the padding P by computing the message authentication code $\mathsf{MAC}_{k_t}(\mathcal{T}_t \mid\mid P)$ with k_t being the shared master key obtained by the client together with the ticket. Both parties will derive session keys from k_t as discussed in §4.2.

Locating the MAC: The MAC is computationally indistinguishable from the pseudo-random padding. To facilitate localisation of the MAC, we place a *cryptographic mark* M right in front of it. The mark is defined as $M = \mathsf{MAC}_{k_t}(\mathcal{T}_t)$. After extracting the ticket from the client's first chunk of bytes, the server is now able to calculate the mark and locate the MAC. We use HMAC-SHA256-128 [40] for the MAC and the mark.

Key rotation: We mentioned earlier that a ScrambleSuit server manages secret keys k_S which are used to encrypt and authenticate session tickets. This prevents clients from tampering with tickets and the server can verify that a newly received and authenticated ticket was, in fact, issued by the server. Servers rotate their k_S keys after a period of seven days. After the generation of new k_S keys, the superseded keys are kept for another seven days in order to decrypt and verify (but not to issue!) tickets which were issued by the superseded keys. As a result, tickets are always valid and redeemable for a period of *exactly seven days*; no matter when they were issued. As a result, as long as a user keeps reconnecting to a ScrambleSuit server at least once a week, *key continuity* is ensured and there is no need for additional out-of-band communication.

Foiling replay: At this point, a censor could still intercept a client's ticket and replay it. This would make the server issue a new ticket for the censor. While the censor would not be able to read the resulting ScrambleSuit control message—the shared master key k_t would be unknown—it is sound design to prevent communication without prior authentication.

We prevent replay attacks—or in other words: ticket double spending—by caching the master key k_t embedded in a ticket. If a server encounters an already cached k_t, it does not reply. We begin to cache a key k_t after a new session ticket was issued and the client confirmed that she correctly received the new ticket by using a special ScrambleSuit message type (see §4.2). We introduced the confirmation step because otherwise a censor could *preplay* a ticket, i.e., intercept it and send it *before* the client. That way, the server would add it to the replay table and would then reject the client's ticket because it would appear to be replayed. This would allow the censor to invalidate the tickets of clients. With the confirmation step, however, censors are no longer able to launch preplay attacks.

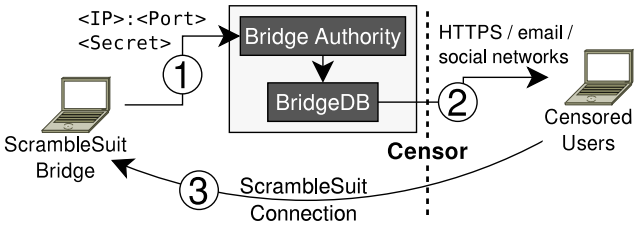


Figure 5: ScrambleSuit bridges send their descriptor to the Tor project's bridge authority ①. From there, it is distributed to censored users who learn about IP address, port and the secret *out-of-band* ②. Finally, direct connections can be established ③.

Inadequate for Tor: Session tickets provide a strong level of protection. Active probing and replay attacks are foiled while forward secrecy is provided. As a result, we envision session tickets to be satisfactory for most application protocols which do not have these properties. Session tickets alone do not, however, integrate well with Tor's existing ecosystem. The reason lies in how Tor bridges are distributed to users. The process is illustrated in Figure 5. Volunteers will set up ScrambleSuit bridges which then publish their descriptors—most notably IP address, port and shared secret—to the bridge authority which then feeds this information into the BridgeDB component ①. In the subsequent step, the gathered descriptors are distributed to censored users ②. The two primary distribution channels are email and HTTPS [41]: users can ask for bridges over email or they can visit the bridge distribution website[5] and obtain a set of bridges after solving a CAPTCHA. Finally, users can establish ScrambleSuit connections ③.

The problem is that *one bridge descriptor* is typically shared by *many users*. All these users would end up with an identical session ticket. This causes two severe problems. First, our replay protection mechanism does not allow reuse of session tickets. Only the first user would be able to authenticate herself. Second, session ticket reuse would lead to identical byte strings at the beginning of a ScrambleSuit handshake which would be a strong distinguisher. These problems lead us to an *additional authentication mechanism*, discussed in §4.1.4, which is optimised for Tor and can function with a secret which is shared by many users as shown in the scenario in Figure 5.

4.1.3 Tor-specific Session Tickets

If ScrambleSuit is used to tunnel Tor traffic (as opposed to other application protocols such as a VPN), a slightly

[5]URL: https://bridges.torproject.org.

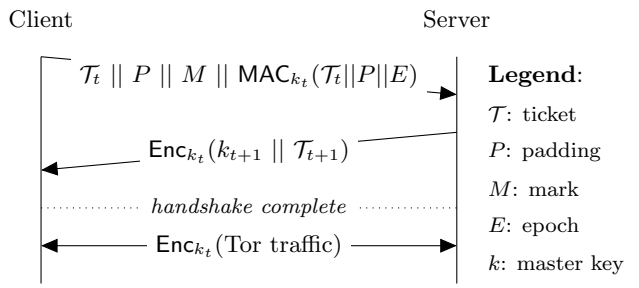


Figure 6: The Tor-specific session ticket handshake. In contrast to Figure 4, it *1)* has no ticket confirmation message and it *2)* employs a timestamp E.

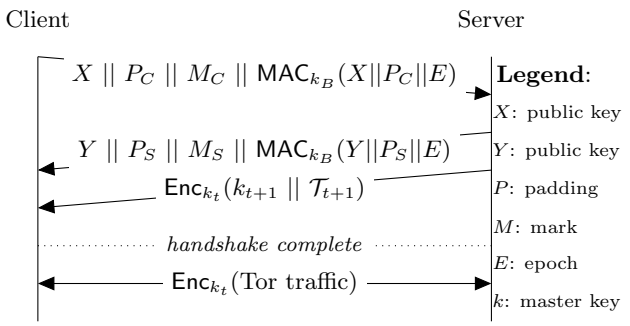


Figure 7: After client and server agreed on the master key k_t using Uniform Diffie-Hellman, the server is issuing a new session ticket for the client. Afterwards, both parties exchange Tor traffic.

modified session ticket handshake is used. As illustrated in Figure 6, the only two differences are:

1. The ticket confirmation message is *no longer necessary*.
2. The MAC in the first message *contains the variable* E holding a timestamp.

Recall that the purpose of the ticket confirmation message is to foil preplay attacks. When Tor is transported by ScrambleSuit, preplay attacks are no longer problematic because if a ticket is lost, the alternative authentication mechanism discussed in §4.1.4 is used instead. This possibility to "fall back" means that a lost ticket no longer implies the inability to authenticate as it does with the classical ticket scheme.

The variable E holds the current Unix timestamp divided by 3600, i.e., the number of hours which have passed since the epoch. Its purpose is to reduce the amount of keys in the replay cache. While this requires clients and servers to have loosely synchronised clocks, the server has to cache redeemed keys for a period of only one hour rather than seven days. We could not use E in the classical ticket scheme illustrated in Figure 4 because it would enable an attack. A censor could repeatedly terminate connections after a client tried to redeem its session ticket. After a while, the variable E will change since it holds a timestamp. This means that the MAC over the ticket handshake would change whereas the ticket *would not change*. We consider this a strong distinguisher.

4.1.4 Authentication Using Uniform Diffie-Hellman

Our second authentication mechanism is an extension of the Uniform Diffie-Hellman (UniformDH) handshake which was proposed in the obfs3 protocol specification [21, §3]. obfs3's handshake makes use of uniformly distributed public keys which are only negligibly different from random bytes (see Appendix A). As a result, UniformDH can be used to agree on a master key k_t without a censor knowing that Diffie-Hellman is used.

In contrast to obfs3, our version of UniformDH is based on the 4096-bit modular exponential group number 16 defined in RFC 3526 [42]. When initiating a UniformDH handshake, the client first generates a 4096-bit private key x. The least significant bit of x is then unset in order to make the number even. The public key X is defined as $X = g^x \pmod p$ where $g = 2$. The server computes its private key y and its public key Y the same way. To prevent a censor from learning that X is a quadratic residue mod p—a clear distinguisher—the client randomly chooses to send either X or $p - X$ to the server. The server can then derive the shared master key by calculating $k_t = X^y \pmod p$. Since the private keys x and y are even, the exponentiations $X^y \pmod p$ and $(p - X)^y \pmod p$ result in the same shared master key.

4.1.5 Extending Uniform Diffie-Hellman

In its original form, the UniformDH construction does not protect against active probing. A censor who suspects UniformDH can simply probe the supposable bridge and opportunistically initiate a UniformDH handshake. To prevent that attack, we now turn UniformDH's anonymous handshake into an authenticated handshake in order to be resistant against active attacks.

We do so quite similar to the session tickets discussed in §4.1.2. As depicted in Figure 7, we concatenate pseudorandom padding P, the mark M, and a MAC. The MAC authenticates the respective public key as well as the padding. The MAC is keyed by a shared secret k_B which is distributed together with the Tor bridge's IP:port tuple over email or HTTPS (see step ① in Figure 5). Similar to tickets, the client's mark $M_C = \mathsf{MAC}_{k_B}(X)$ is used to easily locate the MAC. Note that k_B *can be reused* because it is only used to key the MAC. The handshake is conducted using UniformDH with randomly chosen public keys. As a result, two subsequent UniformDH handshakes based on the same k_B will appear different to a censor. We defend against replay attacks by adding E, the Unix epoch divided by 3600, to the MAC and cache the MAC for a period of one hour.

A successful UniformDH key agreement is followed by the server issuing a session ticket for the client. The client will then redeem this ticket upon connecting to the server the next time. Accordingly, we expect the UniformDH handshake to be done *only once*: namely when a Tor client connects to a bridge for the first time. From then on, session tickets as presented in §4.1.3 will be used for authentication.

To a censor, the payload of both authentication schemes is computationally indistinguishable from randomness. Furthermore, both schemes employ a two-way handshake and padding is used for the two handshakes to exhibit the same average length. As a result, a censor who is assuming that a server is running ScrambleSuit is unable to tell whether a client successfully authenticated herself by using UniformDH or by redeeming a session ticket.

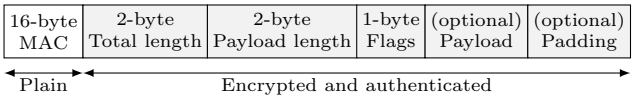


Figure 8: ScrambleSuit's message header format. The entire message is computationally indistinguishable from randomness.

Table 1: ScrambleSuit's protocol message flags.

Flag name	Bit #	Meaning
`FLAG_PAYLOAD`	1	Application data.
`FLAG_NEW_TICKET`	2	Newly issued session ticket and master key.
`FLAG_ACK_TICKET`	3	Acknowledgement of receipt of the session ticket.
`FLAG_PRNG_SEED`	4	PRNG seed to reproduce probability distributions.
`FLAG_REKEY`	5	Initiate rekeying before AES' counter wraps.

We finally stress that bootstrapping ScrambleSuit using UniformDH provides *less security* than when bootstrapped using session tickets. Since the secret key k_B for UniformDH will be used by multiple clients, a malicious client in the possession of k_B who is able to eavesdrop on the connection of another client using the same ScrambleSuit server can conduct active MitM attacks. While Tor does protect against active MitM attacks[6], this can be problematic for application protocols other than Tor. Therefore, we emphasise that session tickets are the preferred authentication mechanism for insecure protocols whereas our UniformDH extension's sole purpose is to make ScrambleSuit work well in Tor's infrastructure. Finally, we discuss usability considerations of our authentication mechanisms in Appendix C.

4.2 Header Format and Confidentiality

Our protocol employs a custom message format whose header is illustrated in Figure 8. In a nutshell, ScrambleSuit exchanges variable-sized messages with optional padding. The padding is always discarded by the remote machine.

The first 16 bytes of the header are reserved for an HMAC-SHA256-128 which protects the integrity and authenticity of the protocol message. In accordance with the encrypt-then-MAC principle, the HMAC is computed over the encrypted remainder of the message. The secret key required by the HMAC is derived from the shared master key k_t.

The HMAC is followed by two bytes which specify the total length of the protocol message. ScrambleSuit's maximum transmission unit is 1448-byte-sized messages. Together with an IP and TCP header (which includes the timestamping option), this adds up to 1500-byte packets which fill an Ethernet frame. In order to be able to distinguish padding from payload, the next two bytes determine the payload length. If no padding is used, the payload length equals the total length.

To separate application data from protocol signaling, we define a 1-byte message flag field. The semantics of all five flags is explained in Table 1. The first bit signals application data in the message body whereas a message with the second bit set contains a newly issued session ticket. The third bit (which can be set together with the first bit) confirms the receipt of a session ticket. Bit number four allows the server to send a pseudo-random number generator (PRNG) seed to the client which is used for our traffic analysis defence (see §4.3). Bit five initiates rekeying which happens before the counter used for AES overflows. To prevent entropy exhaustion attacks, rekeying can only be triggered by the server. We reserve the remaining three bits for future use.

The header is then followed by the message payload which contains the application protocol transported by ScrambleSuit. We employ encryption in order to hide the application protocol, the padding as well as ScrambleSuit's header. With regard to Tor, this means that the already encrypted Tor traffic is wrapped inside yet another layer of encryption. For encryption, we use 256-bit AES in counter mode. The counter mode effectively turns AES into a stream cipher. We use two symmetric keys: one for the traffic $C \rightarrow S$ and one for $S \rightarrow C$. Both symmetric keys as well as the respective nonces for the counter mode are derived from the shared 256-bit master key using HKDF based on SHA256 [43].

4.3 Polymorphic Shape

So far, we discussed defences against censors aiming to analyse packet payload or conduct active attacks to reveal ScrambleSuit's presence. However, a censor could make use of *traffic analysis*, i.e., analyse communication aspects other than the payload. In this section, we propose lightweight countermeasures to diminish—but not to defeat!—such attacks. In particular, we will teach every ScrambleSuit server to generate its own and unique "protocol shape"[7].

Our definition of ScrambleSuit's shape is twofold: we consider *packet lengths* and *inter-arrival times*. While encrypting our protocol messages renders payload analysis useless, these two flow metrics still leak information about the transported application [44, 45, 46]. As a result, we seek to disguise these characteristics in order to decrease the accuracy of protocol classifiers trained to identify our protocol. Our approach to this problem is *protocol polymorphism*.

We achieve polymorphism by creating *one protocol shape for every server*. When a ScrambleSuit server bootstraps for the first time, it randomly generates a 256-bit seed. This seed is then fed into a PRNG which is used to obtain two discrete probability distributions. These two distributions dictate the desired shape of packet lengths and inter-arrival times. Furthermore, a server communicates its unique PRNG seed to clients (see Table 1) after successful authentication. Since the seed is shared by both parties, they can generate identical probability distributions and thus shape their traffic the same way. A censor monitoring two distinct ScrambleSuit servers will observe different distributions for packet lengths and inter-arrival times.

Once our PRNG is seeded, we generate the two distributions by first determining the amount of bins n which is uniformly chosen from the set $\{1..100\}$. In the next step, we assign each bin b_i for $1 \leq i \leq n$ a probability by randomly picking a value in the interval $]0, 1 - \sum_{j=0}^{i-1} b_j[$ with $b_0 = 0$. The following gives an example for four bins.

[6] A Tor client contains hard-coded keys of the directory authorities which then sign the network consensus.

[7] This happens similar to the *scramble suits* in Philip K. Dick's novel "A Scanner Darkly".

$$b_0 \leftarrow 0 \quad (1)$$

$$b_1 \xleftarrow{R}]0, 1 - b_0[\quad (2)$$

$$b_2 \xleftarrow{R}]0, 1 - b_0 - b_1[\quad (3)$$

$$b_n \xleftarrow{R}]0, 1 - b_0 - ... - b_{n-1}[\quad (4)$$

4.3.1 Packet Length Adaption

It is well known that a network flow's packet length distribution leaks information about the network protocol [28, 44] and even the content [47, 46]. For instance, a large fraction of Tor's traffic is composed of 568-byte packets which is the result of Tor's internal use of 512-byte cells plus TLS' header (see Figure 10). These 568-byte packets form a strong distinguisher which can be used to spot a Tor flow by simply capturing a few dozen network packets as shown by Weinberg et al. [16]. To defend against such simple applications of traffic analysis, we modify the packet length distribution of our transported application.

Typically, non-interactive TCP applications transmit segments filling the network link's maximum transmission unit (MTU) as long as they have enough to "say". Applications will only send packets smaller than the MTU if there is not enough data in the send buffer. Due to their sheer volume[8], we deem MTU-sized packets to be of no interest to censors. What we aim to disguise is only packets *smaller than the MTU*. This is done by randomly sampling a packet length from the probability distribution over all our packet lengths. The original packet length is then padded to fit the sampled packet length. The padding can be anything in between 0 and 1520 bytes. The reason for 1520 instead of 1499 bytes is that the smallest unit we can transmit is an empty ScrambleSuit message: 21 bytes.

4.3.2 Inter-Arrival Time Adaption

Analogous to packet lengths, the distribution of inter-arrival times between consecutive packets has discriminative power and can be used by censors to identify protocols [49]. While inter-arrival times are frequently distorted by jitter, overloaded middle boxes and the communicating end points, it would be no sound strategy to assume the network to be unreliable enough to render measurements difficult.

We employ an obfuscation mechanism analogous to the packet length adaption discussed earlier: first, the shared PRNG seed is used to generate a pseudo-random probability distribution. After the amount of bins is determined, each bin is assigned a probability in the range of $[0, 10[$ milliseconds. The motivation for this choice is explained in Appendix B. Random samples are then drawn from this distribution which are used to artificially delay network packets. Similar to the implementation of SkypeMorph [17], we make use of a dedicated send buffer which is processed independently of the locally incoming data. This decoupling of the incoming and outgoing data stream makes it possible to "reshape" inter-arrival times.

4.3.3 Shortcomings

It is important to note that for a censor armed with a well-chosen set of features and sufficient computational resources, traffic analysis can be a powerful attack. Robust defences, on the other hand, are believed to be expensive [45]. For instance, Dyer et al.'s simple yet powerful VNG++ classifier [45] only makes use of coarse features such as connection duration, total bytes transferred and the "burstiness" of a flow. Significantly decreasing VNG++'s accuracy would cause a drastically increased protocol overhead.

Nevertheless, traffic analysis does not give censors a *certain answer*. False positives are always a problem and can lead to over-blocking. As mentioned in our threat model, we believe that the censor might use traffic analysis to select a subset of traffic for closer inspection but not to block flows.

[8]According to a study conducted by CAIDA [48], MTU-sized packets form a significant fraction of all observed packet lengths.

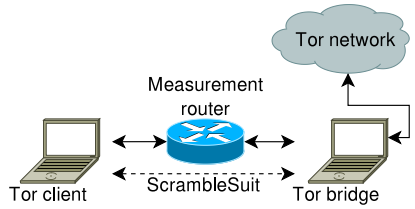


Figure 9: Our experimental setup used to measure ScrambleSuit's obfuscation and performance.

4.4 Cryptographic Assumptions

Our authentication mechanisms rely on *1)* AES-CBC, *2)* pseudo-random initialisation vectors, *3)* HMAC, *4)* pseudo-random padding and *5)* uniformly distributed Diffie-Hellman public keys. Exchanged application data is then encrypted using AES-CTR and authenticated by an HMAC. We expect all data exchanged between ScrambleSuit servers and clients to be computationally indistinguishable[9] from random data of the same length. As a result, we have the following assumptions regarding our cryptographic building blocks. We assume AES to be a pseudo-random permutation and our HMAC to be a pseudo-random function. The padding is assumed to come from a cryptographically secure PRNG and the public keys are assumed to be uniformly distributed (see obfs3 [21]).

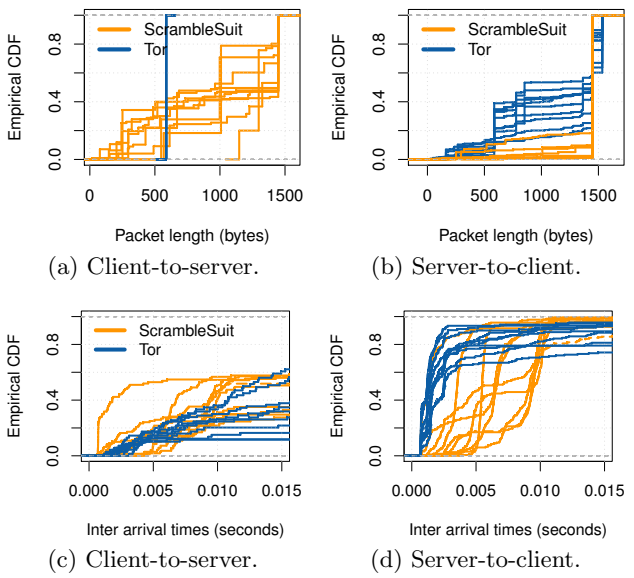
5. EXPERIMENTAL EVALUATION

We implemented a fully functional prototype of ScrambleSuit in the form of several Python modules for obfsproxy[10]. Our prototype consists of approximately 2,200 lines of code. We used the library PyCrypto [51] for cryptographic primitives. The measurements discussed below were all conducted using this prototype.

As illustrated in Figure 9, our experimental setup consisted of two Debian GNU/Linux machines which were connected over a router performing the measurements. All three machines were connected over 100 Mbit/s Ethernet. We expect this setup to be ideal for a censor because it minimises network interference such as jitter or packet fragmentation. As a result, we believe that a censor would do worse in practice. Both of our machines were running Tor v0.2.4.15-rc and obfsproxy. The Tor bridge was configured to remain private

[9]Our goal is to achieve a security level equivalent to at least 128 bits of symmetric security as defined in [50].

[10]The code is available under a free license at http://veri.nymity.ch/scramblesuit/.



(a) Client-to-server. (b) Server-to-client.

(c) Client-to-server. (d) Server-to-client.

Figure 10: Tor's and ScrambleSuit's packet length distribution and inter-arrival times for both, client-to-server and server-to-client traffic.

and only used by our client. The bridge then relayed all traffic into the public Tor network. Note that ScrambleSuit was only "spoken" in between the client and the bridge.

5.1 Blocking Resistance

It is difficult to evaluate the effectiveness of our obfuscation techniques since ScrambleSuit does not have a cover protocol to mimic. Otherwise, our evaluation would simply investigate the similarity between our protocol and its cover protocol. Instead of measuring ScrambleSuit's closeness to a mimicked protocol, we measure the *deviation* from its *transported application*, i.e., Tor. Intuitively, higher deviation would imply better obfuscation.

We obtained traces of packet lengths and inter-arrival times for ScrambleSuit and Tor. In the following, we qualitatively compare both traces. To create network traffic for these traces, we repeatedly downloaded the 1 MB Linux kernel v1.0 from kernel.org[11] on the client. We downloaded the file ten times over Tor and ScrambleSuit, respectively. The measurements only covered the download and not ScrambleSuit's authentication or Tor's bootstrapping.

The two packet length distributions are illustrated in Figure 10(a) and 10(b). The dark blue lines represent Tor flows whereas bright orange lines depict ScrambleSuit. The packet length distributions clearly show the prevalence of Tor's 586-byte packets; even more so in client-to-server traffic where the purpose of these packets is flow control. While server-to-client traffic carries less 586-byte packets, they still form a significant fraction of overall packet lengths. ScrambleSuit effectively eliminates this traffic signature and transmits more MTU-sized packets. Recall that every ScrambleSuit download represents only *one specific shape*. Different servers exhibit different shapes.

[11]URL: https://www.kernel.org/pub/linux/kernel/v1.0/linux-1.0.tar.bz2.

Figure 10(c) and 10(d) depict the inter-arrival times derived from the same data. Again, Tor is shown as dark blue and ScrambleSuit as bright orange lines. The inter-arrival times in Figure 10(c) tend to be rather high—only roughly 60% of Tor packets had an inter-arrival delay below 15 ms—because the bulk data was travelling from the server to the client. For this reason, the delays are significantly smaller in Figure 10(d).

Once again, ScrambleSuit visibly deviates from Tor's distribution. However, as we will show in §5.2.2, artificially increased inter-arrival times have a negative effect on throughput.

5.2 Performance

We are interested in both *computational* as well as *network* overhead. The following sections discuss the amount of overhead ScrambleSuit carries compared to a bare Tor connection.

5.2.1 Cryptographic Overhead

Our two authentication mechanisms come with small computational overhead. For both parties, UniformDH requires two modular exponentiations and two MAC generations[12]. Session tickets—which constitute the majority of authentications—are even cheaper: the client again calculates two MACs whereas the server symmetrically decrypts the ticket and verifies two MACs.

After authentication, protocol messages are symmetrically encrypted and protected by a MAC. We expect the low computational overhead to make ScrambleSuit suitable for resource-constrained devices such as smartphones.

5.2.2 Network Overhead

Our protocol's network overhead is increased by the artificial inter-arrival times, packet padding and the protocol header. In order to gain a good understanding of the exact network overhead, we created a 1,000,000-byte file containing random bytes and placed it on a web server operated by Karlstad University. We then downloaded this file using `wget`; 25 times over HTTP, Tor, ScrambleSuit and ScrambleSuit without inter-arrival time obfuscation, respectively. For Tor and ScrambleSuit, we established a new circuit for every download but we used the same entry guard to eliminate unnecessary variance. All data was captured after a Tor circuit has been established, so handshakes are not part of the data. We then calculated the mean μ and the standard deviation σ for several performance metrics. The results are depicted in Table 2.

The goodput refers to the application layer throughput. We achieved very high values for the HTTP download because the file transfer could be carried out over the LAN. Tor averaged at roughly 280 KB/s and ScrambleSuit achieved slightly more than half of that. Just like Tor, ScrambleSuit exhibits high standard deviation. The main reason for this is differences in Tor circuit throughput. ScrambleSuit without inter-arrival time obfuscation is comparable to Tor. In fact, it exceeds Tor's throughput but this can again be explained by varying circuit throughput. This shows that the obfuscation of inter-arrival times has the biggest impact on ScrambleSuit's throughput (see also Appendix B).

[12]It is necessary to calculate the authenticating MAC as well as the mark used to locate the MAC.

Table 2: Mean (μ) and standard deviation (σ) of the goodput, transferred KBytes and the total overhead. The data was generated based on the download of a 1,000,000-byte file.

	HTTP		Tor		ScrambleSuit		ScrambleSuit-nodelay	
	μ	σ	μ	σ	μ	σ	μ	σ
Goodput	6.3 MB/s	3.4 MB/s	286 KB/s	227 KB/s	148 KB/s	61 KB/s	321 KB/s	231 KB/s
C→S KBytes	23.1	1.6	66.4	6.5	122.9	24.1	111.9	17.9
S→C KBytes	1047	20.7	1130	12.8	1397.8	125.7	1342.7	112.2
Total overhead	7%	2.2%	19.6%	1.9%	52.1%	15%	45.5%	13%

The next two rows of Table 2 refer to the transferred KBytes from client to server (C→S) and server to client (S→C). Note that these metrics cover all the data which was present on the wire; including IP and TCP header. The consideration of IP and TCP overhead is important because ScrambleSuit's packet padding mechanism introduces additional TCP/IP packets. Unsurprisingly, Tor transferred more data than HTTP because of Tor's and TLS' protocol overhead. ScrambleSuit transferred the most data because of the additional protocol header as well as the varying packet lengths. Given ScrambleSuit's 1448-byte MTU, the 21-byte protocol header only accounts for 1.5% overhead.

The last row in Table 2 illustrates the total protocol overhead which is simply a function of the previous two rows. HTTP has the lowest overhead followed by Tor and finally ScrambleSuit. Our protocol exhibits 45–50% overhead which is about twice as much as Tor.

6. DISCUSSION

Active Probing: A censor could still actively probe a ScrambleSuit server. Upon establishing a TCP connection, a censor could proceed by sending arbitrary data. However, without knowing the UniformDH shared secret or possessing a valid session ticket, authentication can not succeed and the server will remain silent.

In contrast to SilentKnock and BridgeSPA, ScrambleSuit does not disguise its "aliveness". While this approach does leak information[13], it has the benefit of making ScrambleSuit significantly easier to deploy due to lack of platform dependencies such as the kernel interface `libnetfilter_queue` to parse raw network packets in userspace.

Injection, Modification, Dropping: A censor could tamper with an established ScrambleSuit connection by injecting, modifying or dropping packets. After authentication, all exchanged data is authenticated which allows the communicating parties to detect such tampering. If the authenticating HMAC of either party is invalid, the connection is terminated immediately.

Hijacking a ScrambleSuit connection is reduced to the same problem; a censor would bypass authentication but is unable to talk to the other party because the session keys are unknown. Finally, dropped packets would be handled by TCP's retransmission mechanism whereas terminated connections could manually be restarted by users.

Payload Analysis: Payload analysis would only yield data which is computationally indistinguishable from randomness. While most encrypted protocols negotiate session parameters in cleartext, VPNs with pre-shared keys and BitTorrent's message stream encryption also bootstrap using high-entropy network packets. Aside from that, it is difficult to survey how many "fully-random" protocols already exist in the wild. One approach would be to monitor large-scale network links and determine which fraction of TCP streams transports only high-entropy data. Similar work was done by White et al. [52] but the authors focused on per-packet rather than on per-connection measurements.

Ideally, more applications considered legitimate by censors would start transporting only high-entropy payload, thus inflating the set of protocols ScrambleSuit can hide amongst. This could be achieved by a major browser employing such encryption on the transport layer or even by an extension for the TLS protocol which would provide optional support for such a mode.

Flow Analysis: Flow analysis would yield a unique distribution of packet lengths and inter-arrival times which is different for every ScrambleSuit server. While strong traffic analysis defence is expensive, these attacks will always have a range of *uncertainty* causing false positives. Our goal was to further increase this uncertainty. We placed more value on defeating active probing attacks because in contrast to traffic analysis, they enable *deterministic* protocol identification.

Future Work: There is room for stronger traffic analysis defence. In particular, ScrambleSuit could be extended to keep track of the "burstiness" of exchanged traffic which is—amongst other features—exploited by the powerful VNG++ classifier. These bursts could then be deliberately distorted with the intention to confuse VNG++. Our shared PRNG seed could be used to make this distortion server-specific.

7. CONCLUSION

We presented ScrambleSuit; a lightweight transport protocol which provides obfuscation for applications such as Tor. The two major contributions of our protocol are the ability to defend against *active probing* and simple *protocol classifiers*. We achieve the former by proposing two authentication mechanisms—one general-purpose and the other specifically for Tor—and the latter by proposing morphing techniques to disguise packet lengths and inter-arrival times.

We further developed a prototype of ScrambleSuit and used it to conduct an experimental evaluation. We discussed the effectiveness of our obfuscation techniques as well as ScrambleSuit's overhead. Our evaluation suggests that our protocol can provide decent protection against censors who do not over-block significantly. As a result, we believe that

[13] A censor learns that a server is online but unwilling to talk unless given the "correct" data.

our protocol can provide a practical alternative in countries which do not whitelist Internet traffic.

In the near future, we aim to deploy ScrambleSuit as part of the Tor browser bundle.

Acknowledgements

We want to thank the anonymous reviewers, George Kadianakis, Harald Lampesberger, Stefan Lindskog, and Michael Rogers who all provided valuable feedback which improved this paper. We further want to express our gratitude to Internetfonden of the Swedish Internet Infrastructure Foundation for supporting the main author's work with a research grant.

All our data and code is freely available at: http://veri.nymity.ch/scramblesuit/.

8. REFERENCES

[1] Thomas H. Ptacek and Timothy N. Newsham. Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection. Technical report, Secure Networks, Inc., 1998.

[2] Olli-Pekka Niemi, Antti Levomäki, and Jukka Manner. Dismantling Intrusion Prevention Systems (Demo). In *SIGCOMM*. ACM, 2012.

[3] Mark Handley, Vern Paxson, and Christian Kreibich. Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics. In *USENIX Security*. USENIX Association, 2001.

[4] Marcel Dischinger, Alan Mislove, Andreas Haeberlen, and Krishna P. Gummadi. Detecting BitTorrent Blocking. In *IMC*. ACM, 2008.

[5] Richard Clayton, Steven J. Murdoch, and Robert N. M. Watson. Ignoring the Great Firewall of China. In *PETS*. Springer, 2006.

[6] Sparks, Neo, Tank, Smith, and Dozer. The Collateral Damage of Internet Censorship by DNS Injection. *SIGCOMM Computer Communication Review*, 42(3), 2012.

[7] Christopher Rhoads and Loretta Chao. Iran's Web Spying Aided By Western Technology, 2009. URL: http://online.wsj.com/article/SB124562668777335653.html.

[8] Jillian C. York. Government Internet Surveillance Starts With Eyes Built in the West, 2011. URL: https://www.eff.org/deeplinks/2011/09/government-internet-surveillance-starts-eyes-built.

[9] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation Onion Router. In *USENIX Security*. USENIX Association, 2004.

[10] The Tor Project. Iran. URL: https://censorshipwiki.torproject.org/CensorshipByCountry/Iran.

[11] Philipp Winter and Stefan Lindskog. How the Great Firewall of China is Blocking Tor. In *FOCI*. USENIX Association, 2012.

[12] The Tor Project. Ethiopia. URL: https://censorshipwiki.torproject.org/CensorshipByCountry/Ethiopia.

[13] Charles Arthur. China tightens 'Great Firewall' internet control with new technology, 2012. URL: http://www.guardian.co.uk/technology/2012/dec/14/china-tightens-great-firewall-internet-control.

[14] GFW actively probes obfs2 bridges, 2013. URL: https://bugs.torproject.org/8591.

[15] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. ZMap: Fast Internet-Wide Scanning and its Security Applications. In *USENIX Security*. USENIX Association, 2013.

[16] Zachary Weinberg, Jeffrey Wang, Vinod Yegneswaran, Linda Briesemeister, Steven Cheung, Frank Wang, and Dan Boneh. StegoTorus: A Camouflage Proxy for the Tor Anonymity System. In *CCS*. ACM, 2012.

[17] Hooman Mohajeri Moghaddam, Baiyu Li, Mohammad Derakhshani, and Ian Goldberg. SkypeMorph: Protocol Obfuscation for Tor Bridges. In *CCS*. ACM, 2012.

[18] Amir Houmansadr, Thomas Riedl, Nikita Borisov, and Andrew Singer. I want my voice to be heard: IP over Voice-over-IP for unobservable censorship circumvention. In *NDSS*. The Internet Society, 2013.

[19] Wenxuan Zhou, Amir Houmansadr, Matthew Caesar, and Nikita Borisov. SWEET: Serving the Web by Exploiting Email Tunnels. In *HotPETS*. Springer, 2013.

[20] The Tor Project. obfs2 (The Twobfuscator). URL: https://gitweb.torproject.org/pluggable-transports/obfsproxy.git/blob/HEAD:/doc/obfs2/obfs2-protocol-spec.txt.

[21] The Tor Project. obfs3 (The Threebfuscator). URL: https://gitweb.torproject.org/pluggable-transports/obfsproxy.git/blob/HEAD:/doc/obfs3/obfs3-protocol-spec.txt.

[22] Brandon Wiley. Dust: A Blocking-Resistant Internet Transport Protocol. Technical report, University of Texas at Austin, 2011.

[23] Viewing cable 09MUSCAT1039, SKYPE CRACKDOWN IN OMAN, 2009. URL: http://wikileaks.org/cable/2009/11/09MUSCAT1039.html.

[24] Russian "Clean Internet" experiment gets green light, 2013. URL: http://rt.com/politics/anti-pedophile-safe-internet-russian-169/.

[25] Small Media. Iranian Internet Infrastructure and Policy Report: Election Edition 2013 (April - June), 2013.

[26] Alberto Dainotti, Claudio Squarcella, Emile Aben, Kimberly C. Claffy, Marco Chiesa, Michele Russo, and Antonio Pescapé. Analysis of Country-wide Internet Outages Caused by Censorship. In *IMC*. ACM, 2011.

[27] Eva Galperin and Jillian C. York. Syria goes dark, 2012. https://www.eff.org/deeplinks/2012/11/syria-goes-dark.

[28] Erik Hjelmvik and Wolfgang John. Breaking and Improving Protocol Obfuscation. Technical report, Chalmers University of Technology, 2010.

[29] Brandon Wiley. Blocking-Resistant Protocol Classification Using Bayesian Model Selection. Technical report, University of Texas at Austin, 2011.

[30] The Tor Project. obfsproxy. URL: https://www.torproject.org/projects/obfsproxy.

[31] Qiyan Wang, Xun Gong, Giang T. K. Nguyen, Amir Houmansadr, and Nikita Borisov. CensorSpoofer: Asymmetric Communication using IP Spoofing for Censorship-Resistant Web Browsing. In *CCS*. ACM, 2012.

[32] Amir Houmansadr, Chad Brubaker, and Vitaly Shmatikov. The Parrot is Dead: Observing Unobservable Network Communications. In *Security & Privacy*. IEEE, 2013.

[33] Patrick Lincoln, Ian Mason, Phillip Porras, Vinod Yegneswaran, Zachary Weinberg, Jeroen Massar, William Simpson, Paul Vixie, and Dan Boneh. Bootstrapping Communications into an Anti-Censorship System. In *FOCI*. USENIX Association, 2012.

[34] Eugene Y. Vasserman, Nicholas Hopper, John Laxson, and James Tyra. SilentKnock: Practical, Provably Undetectable Authentication. In *ESORICS*. Springer, 2007.

[35] Rob Smits, Divam Jain, Sarah Pidcock, Ian Goldberg, and Urs Hengartner. BridgeSPA: Improving Tor Bridges with Single Packet Authorization. In *WPES*. ACM, 2011.

[36] Martin Johnson. China, GitHub and the man-in-the-middle, 2013. URL: https://en.greatfire.org/blog/2013/jan/china-github-and-man-middle.

[37] Ronald L. Rivest, Adi Shamir, and David A. Wagner. Time-lock Puzzles and Timed-release Crypto. Technical report, Massachusetts Institute of Technology, 1996.

[38] Ben Laurie and Richard Clayton. "Proof-of-Work" Proves Not to Work. In *WEIS*, 2004.

[39] Joseph Salowey, Hao Zhou, Pasi Eronen, and Hannes Tschofenig. RFC 5077: Transport Layer Security (TLS) Session Resumption without Server-Side State, 2008.

[40] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. RFC 2104: HMAC: Keyed-Hashing for Message Authentication, 1997.
[41] Zhen Ling, Xinwen Fu, Wei Yu, Junzhou Luo, and Ming Yang. Extensive Analysis and Large-Scale Empirical Evaluation of Tor Bridge Discovery. In *INFOCOM*. IEEE, 2012.
[42] Tero Kivinen and Mika Kojo. RFC 3526: More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE), 2003.
[43] Hugo Krawczyk and Pasi Eronen. RFC 5869: HMAC-based Extract-and-Expand Key Derivation Function (HKDF), 2010.
[44] Manuel Crotti, Maurizio Dusi, Francesco Gringoli, and Luca Salgarelli. Traffic Classification through Simple Statistical Fingerprinting. *SIGCOMM Computer Communication Review*, 37(1), 2007.
[45] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail. In *Security & Privacy*. IEEE, 2012.
[46] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. Touching from a Distance: Website Fingerprinting Attacks and Defenses. In *CCS*. ACM, 2012.
[47] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. Website Fingerprinting in Onion Routing Based Anonymization Networks. In *WPES*. ACM, 2011.
[48] CAIDA. Packet size distribution comparison between Internet links in 1998 and 2008, 2010. URL: http://www.caida.org/research/traffic-analysis/pkt_size_distribution/graphs.xml.
[49] Mohamad Jaber, Roberto G. Cascella, and Chadi Barakat. Can we trust the inter-packet time for traffic classification? In *ICC*. IEEE, 2011.
[50] ECRYPT II Yearly Report on Algorithms and Keysizes, 2012.
[51] Dwayne C. Litzenberger. PyCrypto - The Python Cryptography Toolkit. URL: https://www.dlitz.net/software/pycrypto/.
[52] Andrew M. White, Srinivas Krishnan, Michael Bailey, Fabian Monrose, and Phillip Porras. Clear and Present Data: Opaque Traffic and its Security Implications for the Future. In *NDSS*. The Internet Society, 2013.

APPENDIX

A. UNIFORMDH PUBLIC KEYS

UniformDH public keys form a distinguisher because they are not distributed over the entire space of 4096 bits. Public keys are always smaller than the modulus defined in RFC 3526 [42]. The unused bit space is, however, small enough for this distinguisher to be negligible. The probability P of a censor observing a UniformDH public key smaller than the 4096-bit modulus n (see reference [42] for n's value) equals:

$$P = \frac{n}{2^{4096} - 1}$$

A censor could now monitor a server's handshakes over an extended period of time in order to determine if the full 4096-bit space is never used which would be a sign of UniformDH. Doing that, a censor needs

$$\frac{ln(0.5)}{ln(P)} \approx 2^{66}$$

observations to have a confidence greater than 50% that a server is conducting UniformDH handshakes. Such an amount of observations is unpractical.

B. INTER-ARRIVAL TIME PARAMETERS

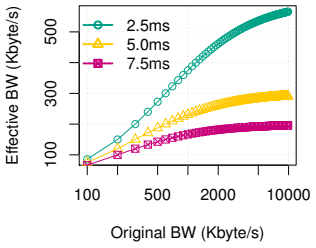


Figure 11: The cost in bandwidth when employing three different obfuscation delays.

The artificial increase of inter-arrival times has a negative impact on throughput. If chosen too high, it can quickly become a nuisance to users. Figure 11 illustrates the mapping from original bandwidth (without obfuscation) to effective bandwidth (after obfuscation) under three different average obfuscation delays. The higher the average delay, the smaller is the effective bandwidth. The plotted data is based on the following equation which yields the overhead α.

$$\alpha = \left(\frac{BPS}{MTU} \cdot E[d]\right) \cdot \frac{1}{1000} + 1$$

The variable MTU refers to the maximum transmission unit which is typically 1500. BPS stands for "bytes per second" and refers to the original bandwidth of the available network link. $E[d]$ represents the expected value of the respective probability distribution d, i.e., the average obfuscation delay. In Figure 11, we plot the expected values 2.5, 5 and 7.5 milliseconds, respectively. ScrambleSuit uses the interval of $[0, 10[$ milliseconds for artificial delays. This interval has an expected value of 5ms which we believe to be a reasonable balance between this obfuscation/performance tradeoff. It is ultimately limited to an effective throughput of 300 Kbyte/s and, when Tor is transported, can achieve throughputs around 150 Kbyte/s as we showed in our experimental evaluation.

C. USABILITY CONSIDERATIONS

In order for a user to successfully connect to a ScrambleSuit server, she needs a *triple*: an IP address, a TCP port and a secret which is either the UniformDH secret k_B or a session ticket tuple $(k_t \mid\mid \mathcal{T}_t)$. We expect these triples to be distributed mostly electronically; over email, instant messaging programs or online social networks. As a result, a user can simply copy and paste the entire triple into her configuration file.

We do, however, also expect verbal distribution of ScrambleSuit triples, e.g., over a telephone line. To facilitate this, we define the encoding format of secrets and tickets to be Base32 which consists of the letters A–Z, the numbers 2–7 as well as the padding character "=". The numbers 0 and 1 are omitted to prevent confusion with the letters I and O. Since there is no distinction between uppercase and lowercase letters, we hope to make verbal distribution less confusing and error-prone. After all, a ScrambleSuit bridge descriptor would look like:

```
Bridge scramblesuit 1.2.3.4:443 password=NCA6I6GZZD42BWUB
```

We believe that the prefix `password=` will find more acceptance amongst users than simply appending the secret.