Cecylia Bocovich and Ian Goldberg

Secure asymmetry and deployability for decoy routing systems

Abstract: Censorship circumvention is often characterized as a cat-and-mouse game between a nation-state censor and the developers of censorship resistance systems. Decoy routing systems offer a solution to censorship resistance that has the potential to tilt this race in the favour of the censorship resistor by using real connections to unblocked, overt sites to deliver censored content to users. This is achieved by employing the help of Internet Service Providers (ISPs) or Autonomous Systems (ASes) that own routers in the middle of the network. However, the deployment of decoy routers has yet to reach fruition. Obstacles to deployment such as the heavy requirements on routers that deploy decoy router relay stations, and the impact on the quality of service for customers that pass through these routers have deterred potential participants from deploying existing systems. Furthermore, connections from clients to overt sites often follow different paths in the upstream and downstream direction, making some existing designs impractical. Although decoy routing systems that lessen the burden on participating routers and accommodate asymmetric flows have been proposed, these arguably more deployable systems suffer from security vulnerabilities that put their users at risk of discovery or make them prone to censorship or denial of service attacks. In this paper, we propose a technique for supporting route asymmetry in previously symmetric decoy routing systems. The resulting asymmetric solution is more secure than previous asymmetric proposals and provides an option for tiered deployment, allowing more cautious ASes to deploy a lightweight, non-blocking relay station that aids in defending against routing-capable adversaries. We also provide an experimental evaluation of relay station performance on off-the-shelf hardware and additional security improvements to recently proposed systems.

Keywords: keywords, keywords

DOI Editor to enter DOI
Received ..; revised ..; accepted ...

1 Introduction

In recent years, Internet censorship has become an increasing world-wide concern. A 2016 Freedom House report declared that Internet freedom has now been in a steady decline for six consecutive years [25]. They reported that in 2016, roughly two-thirds of Internet users dealt with government censorship. This censorship aims to cut off access from websites that support political opposition, marginalized communities, and images that criticize or satirize those in power. Furthermore, journalists and users of social media that disseminate, or merely read, content that a censoring nation deems contrary have faced personal dangers such as arrest or increased scrutiny. This makes hiding the use of censorship resistance systems paramount in developing new circumvention technologies.

Tools for censorship circumvention range from simple proxies that hide the IP addresses of visited sites, to systems that disguise traffic patterns by padding packets or mimicking allowed protocols. These systems have evolved as a result of a cat-and-mouse game between nation-state censors and censorship resistors [34]. As new techniques for evading censorship arise, censors tweak their filtering systems to identify the weaknesses in existing tools that signal their usage. This makes hiding the fact that the user is using a specific tool (given that the censor knows the tool exists and the details of the system) critical to both the user's safety and the success of the censorship resistance system.

Decoy routing (also known as end-to-middle (E2M) proxying) [4, 11, 18, 24, 31, 39, 40] is a technique for censorship resistance that has the potential to skew the cat-and-mouse game in the favour of the censorship resistor. The key way decoy routing hides its usage is by *appropriation* of real, uncensored ("overt") traffic to provide access to covert information instead of *mimicking* allowed traffic. Mimicry is a common technique employed by censorship circumvention systems [6, 9, 28, 36, 38],

Cecylia Bocovich: Cheriton School of Computer Science, University of Waterloo, cbocovic@uwaterloo.ca

Ian Goldberg: Cheriton School of Computer Science, University of Waterloo, iang@cs.uwaterloo.ca

but by its nature deviates from real traffic in ways that a sufficiently advanced censor could detect [17]. Although such techniques have yet to be documented for use by nation-states in an effort to detect the usage of censorship resistance systems, they still pose a threat to individual users who may, now or in the future, face dire consequences for defying their jurisdiction's strict controls on Internet usage.

Although decoy routing provides strong security properties against both active and passive attacks, there are numerous obstacles to deployment. The deployment of a decoy routing system relies on the participation of autonomous systems (ASes) that own routers in the middle of the network. Previous work on the optimal placement of decoy routers aims to maximize the number of unblocked, overt sites available and minimize the required amount of deployed stations [5, 20, 30]. However, researchers have yet to convince large ASes to deploy decoy routing in a production setting. While recent work on analyzing a small-scale decoy routing deployment [13] provides hope that ISPs are willing to deploy lightweight decoy routers, we are still a long way from convincing the majority of ASes to adopt these systems for Internet freedom purposes. Concerns such as the hardware required to block, modify, or drop traffic at the router, the effect checking for steganographic tags would have on regular traffic, and the logistics involved in setting up and maintaining a relay station remain deterrents for both large and small ASes.

Furthermore, connections to overt sites are often asymmetric. While they may cross a router with a deployed decoy routing relay station on the path to an overt site, the path taken back from the overt site to the user may not cross the same router. This makes the deployment of decoy routing systems more difficult, perhaps necessitating a larger number of participant ASes. While some asymmetric solutions exist [11, 18, 39], they suffer from security vulnerabilities that could put users already under the scrutiny of a nation-state censor at risk. Waterfall, a recently proposed asymmetric decoy routing system, requires a relay station only on the downstream half of a flow [31]. This provides resistance against routing around decoys (RAD) attacks [32]. Furthermore, Waterfall provides significant security improvements to existing asymmetric designs by employing and improving upon the techniques used in Slitheen [4] to securely relay covert information in an undetectable and high-bandwidth manner. However, the registration protocol is prone to denial of service attacks and blocking.

In this paper, we address the main challenges to deployability that current decoy routing systems face. We provide an experimental analysis of affordable, off-the-shelf hardware that can be used by ASes in the deployment of decoy routing relay stations. To address the problem of route asymmetry, we leverage the fact that routes between specific clients and overt sites are very stable, meaning they pass through the same set of ASes in subsequent flows. We propose a "gossip" protocol that may be applied to all previously symmetric systems to make them work in an asymmetric setting. In keeping with the ideas presented in Waterfall, our approach emphasizes downstream traffic. We use extremely light-weight upstream stations (simple taps) to relay information to stations on the downstream half of the flow that incur a bandwidth overhead of only $1.0055\times$ the total bandwidth through upstream station. Our design provides a more secure alternative to Waterfall's registration protocol and requires fewer deployed *heavy-weight* relay stations that perform in-line blocking and intense computations than symmetric systems. We require as few as five heavy-weight stations for a highly connected, routing capable adversary such as China, as opposed to the hundreds of stations required by symmetric designs. Our contributions are as follows:

- We propose a new solution for routing asymmetry that is applicable to all previously symmetric systems and evaluate the overhead cost of deployment as well as its resistance to RAD attacks.
- We provide measurements on the impact relay station deployment would have on regular traffic through a participant ISP. We hope our results will convince an AS interested in deploying a relay station that their quality of service will not be largely affected by even the most complex of the recently proposed decoy routing systems.
- We present a possible vulnerability in decoy routing systems that modify and re-encrypt TLS application data and propose a solution that defends against an adversary capable of seeing traffic on both sides of the relay station. This adversary falls outside the decoy routing threat model for the censor, but a non-censoring adversary could exploit the vulnerability to decrypt or modify covert traffic.

In the next section, we discuss existing work in censorship circumvention. In Section 3 we propose our solution for handling asymmetric routes, followed by experimental results on relay station efficiency in Section 4 and a security analysis in Section 5. We end with a conclusion

and a discussion of future steps towards deployment in Section 6.

2 Censorship Circumvention

Nation-state censors filter Internet traffic before it leaves their area of influence. Past studies on Internet filtering have revealed a variety of techniques such as blocking access to specific IP addresses [2, 29], filtering DNS requests by the URL or keyword [29], or performing more sophisticated deep-packet inspection techniques to determine the usage of censorship resistance tools [34, 37]. Many Internet filtering techniques employed by censors have evolved in response to the development of the censorship circumvention systems. A notable example of this phenomenon is the interaction between Internet filtering in China and advances in the censorship resistance aspects of Tor [8].

Originally developed to provide anonymity for web browsing, Tor has been adopted by many for its usefulness in circumventing government censorship. By disguising which website a user is browsing, Tor prevents a censor from learning whether or not a user is accessing a blocked website. As such, many countries that censor web traffic began to block all access to Tor. Tschantz et al. [34] document the interplay between Tor and China's Great Firewall (GFW) with extensive empirical evidence taken from bug reports, correspondence with The Tor Project, and changes in the Tor protocol. In response to the blocking of publicly listed Tor relays, unlisted Tor relays called *bridges* began to be circulated privately, enabling their use for a short period of time before their discovery by censors [7]. When bridges are discovered by censors and subsequently blocked, new bridges are cycled into use. The GFW responded to the introduction of bridges by using more sophisticated deep-packet inspection techniques and exploiting unique patterns in the Tor protocol to differentiate Tor traffic from regular web browsing. This led to the development of pluggable transports [6, 9, 12, 28, 36, 38], designed to encapsulate and disguise the defining characteristics of Tor traffic.

The majority of pluggable transports take one of three different approaches to disguising Tor traffic: obfuscation, mimicry, or appropriation. The first approach aims to mask the defining characteristics of Tor traffic by making the connection look as random as possible [6, 38]. The success of this technique is grounded in the assumption that censors are unwilling to block traffic that they are unable to definitively classify as censorship resistance or contrary to their governance, as that would possibly lead to an increase in public unrest [10]. However, past precedent indicates that in critical times censors may be willing to take the risk; Aryan et al. recorded the blocking of undefined Internet protocols by the government of Iran during the 2013 presidential elections [2].

Mimicry aims to make connections indistinguishable from popular unblocked content or services, forcing censors to make a difficult decision: to either continue to expand their list of blocked sites to include popular services (thereby risking public unrest), or surrender their position. Many pluggable transports shape traffic or encapsulate it in messages that closely resemble protocols such as HTTP [9], Skype [28], or HTML [36]. The ultimatum presented to the censor rests entirely on the ability of these systems to mimic allowed sites and services more closely than the censor's ability to exploit minor differences. Houmansadr et al. [17] argue that the maintenance of near-perfect mimicry is extremely difficult; as advances in computing allow censors to classify large amounts of traffic more accurately, censorship resistors will see themselves on the losing side of this reactive battle.

While the cycling of bridges and use of pluggable transports has proven effective in many regions for providing access to Tor, there is a danger that after their discovery, censoring nations will start to punish users that have connected to IP addresses revealed to be entry points to the Tor network. Meek is a pluggable transport that *appropriates* connections to allowed sites and services. It disguises the IP address of bridges by hiding them behind popular services such as Google, Amazon Web Services, or Microsoft Azure using a technique called domain fronting [12]. A user makes a real connection to one of these large domains and accesses a proxy running inside their systems. Not only does this protect the user by making it impossible for a censor to link them to a specific IP address used to access Tor, it also leverages a powerful incentive for governments that do not control equivalent services not to block access to these powerful sites. For nations that do posses equivalence, the efficacy of this method diminishes.

Other circumvention systems appropriate allowed protocols and tunnel censorship resistance traffic through them [3, 19, 21, 27]. By using existing implementations of protocols such as Voice-over-IP, video streaming services, or email as a covert channel, these systems are not discernible by a censor due to differences in implementation as are systems that use mimicry. However, Geddes et al. [14] show that differences in the

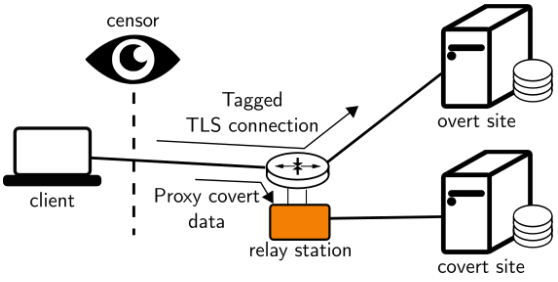


Fig. 1. An overview of a generic decoy routing system. A decoy routing session has two phases: the tagging phase, and the proxy phase. In the tagging phase, the client embeds a tag in a seemingly random channel (such as the ClientHello random nonce in a TLS handshake or the ciphertext of the TLS application data) to an overt, unblocked site. The relay station recognizes the tag and extracts from it the information necessary to compute the TLS master secret. In the proxy phase, the relay station decrypts TLS application data to receive upstream covert traffic from the client. It initiates a connection to the covert site and begins to proxy covert data between the client and the covert site. From the censor's point of view, the client is making a normal connection to, and receiving content from, the overt site.

typical traffic transported by the cover protocol and the tunnelled censorship resistance traffic can produce identifying characteristics that would allow a censor to block or identify their use.

2.1 Decoy routing

Decoy routing was originally proposed by three independent research groups in 2011 as a means to move censorship resistance systems from easily blocked endpoints to the middle of the network [18, 24, 40]. These original first-generation systems were followed by several second-generation systems, characterized by their main goal of improving the deployability and security of their predecessors [4, 11, 39]. Waterfall is a third-generation system, developed in response to a trade-off between asymmetry and security present in the second generation of proposals, that provides strong defenses against known attacks and addresses practical concerns surrounding the deployment of decoy routing systems [31]. The motivation of decoy routing was to level the playing field by fighting powerful nation-state censors with powerful nation-state defenses. The technique requires the cooperation and active participation of Internet Service Providers (ISPs) and Autonomous Systems (ASes) that own routers *outside* of regions that censor Internet traffic. Non-censoring jurisdictions would place decoy routers, or *relay stations*, at strategic points on the path between users in censoring regions and popular, unblocked sites. Users suffering from censorship could then make a connection to these unblocked *overt sites* and send a steganographic tag, recognizable by the deployed relay stations—and *only* by them—as a request to access censored content, appropriating the connection. From the censor's point of view, this tag and other features of the user's connection are identical to any other access to the overt site. We give an overview of a generic decoy routing system architecture in Figure 1.

The details of the steganographic tagging procedure vary across decoy routing systems. Telex [40], Curveball [24], Rebound [11], and Slitheen [4] place a tag in the random nonce of the ClientHello message in the TLS handshake with the overt site. This tag is recognizable only by a targeted deployed relay station and gives the relay station the information necessary to compute the TLS master secret for the session and man-in-the-middle the connection between the user and the overt site. Cirripede [18] and Waterfall [31] have registration protocols that are distinct from the connections that deliver covert content. TapDance [39] and Rebound use steganographic encodings to make their tagging protocol work for asymmetrically routed flows.

After the tagging/registration phase is complete, the decoy routing system begins to relay covert information to the client in the proxy phase of the decoy session. The details of this process also vary, and this phase is the aspect of decoy routing subject to the identifying characteristics of appropriation. It is at this point that the relay stations for most decoy routing systems abandon or sever the connection to the overt site. While the registration phase of early systems is provably indistinguishable from regular, non-decoy, connections to the overt site, traffic in the proxy phase takes the shape and characteristics of the *covert* site [11, 18, 24, 39, 40]. This makes these systems vulnerable to website fingerprinting [16, 35] or latency analysis attacks [32].

Slitheen and Waterfall address the weaknesses in the appropriation of the TLS connection to an overt site during the proxy phase by not severing or abandoning the connection to the overt site. They maintain the connection to the overt site, and send valid HTTP requests for real overt resources. The client loads overt websites in the exact manner that a regular user would, parsing HTTP responses and loading additional content with the help of an actual web browser, termed an overt user simulator (OUS). In Slitheen, covert content is delivered to the user by the relay station in the place of "leaf resources", or resources such as images or videos that would not prompt a browser to make additional connections for more resources. In Waterfall, all previously

Table 1. A comparison of the deployability features and security properties of existing systems. We indicate that a system has the property or feature listed on the left of the table with a filled circle ●. Systems that lack a feature or property are marked with an empty circle ○. Our proposed design to support asymmetric routes enables the deployment of lightweight upstream stations with no in-line blocking in addition to the original heavyweight downstream stations. (We denote the requirement for in-line blocking in only the downstream stations with the half-filled circle ◐.) This improvement to deployability provides the potential to thwart RAD attacks.

	Telex [40]	Telex + this work	Cirripede [18]	Curveball [24]	Curveball+this work	TapDance [39]	Rebound [11]	Slitheen [4]	Slitheen + this work	Waterfall [31]	Waterfall + this work
No in-line blocking	○	◐	○	○	◐	●	○	○	◐	◐	◐
Asymmetric	○	●	●	○	●	●	●	○	●	●	●
Defends against TCP replay attacks	●	●	●	●	●	○	●	●	●	●	●
Defends against latency analysis	○	○	○	○	○	○	●	●	●	●	●
Defends against website fingerprinting	○	○	○	○	○	○	○	●	●	●	●
RAD-resistant	○	●	○	○	●	●	○	○	●	●	●
DoS-resistant registration	●	●	●	●	●	●	●	●	●	○	●

cached resources are replaced to increase the amount of covert bandwidth available to the user.

In both systems, resources are replaced on a per-packet basis as they pass through the relay station, making decoy routing traffic identical to a regular access of the overt site. The pattern of connections to overt servers, packet sizes, and page load times are indistinguishable from a non-decoy session, removing the distinguishing characteristics that arise from appropriating the connection to the overt site. The task of the censor now falls on determining whether the access pattern of the overt sites themselves is done by a user simulator or a regular user, a problem that is more likely to be error-prone for a censor than existing website fingerprinting techniques.

2.2 Known challenges to deployment

Recently, a number of research groups have proposed solutions to the decoy router placement problem (DRP) that aim to maximize the coverage of overt sites available through decoy routing stations and minimize the number of decoy routers needed to successfully inhibit a censor's ability to evade decoy routers and block overt sites [5, 20, 30]. Sufficiently powerful censors can perform Routing Around Decoys (RAD) attacks by manipulating BGP and routing tables to send traffic to overt sites down paths that do not contain a deployed relay station [31, 32]. With enough deployed stations, these attacks become extremely difficult and expensive [20]. However, we have yet to see deployment on large ASes, let alone the widespread placement of relay stations in the middle of the network.

Wustrow et al. [39] were the first to closely examine deployment challenges, and developed TapDance as the result of discussions with ISPs about their reluctance to deploy existing systems. The resource requirements of relay stations and route asymmetry were cited as the most onerous to ISPs and practical usage of existing systems. Telex and Curveball both require the relay station to perform in-line flow blocking, severing the connection between the user and the overt site after the TLS handshake. This not only requires sophisticated and potentially expensive hardware, it also violates the terms of service many ISPs have with overt sites. Because TapDance does not perform in-line flow blocking, it does not have an impact on the quality of service of HTTPS traffic through the router of a deployed relay station. This has made the trial deployment of TapDance successful, at both a regional ISP and university network [13]. During the trial, the deployed TapDance stations were able to serve up to 3,000 clients while processing 40 Gb/s of regular ISP traffic. However, the deployability of TapDance is offset by security vulnerabilities that may lead to easy blocking by a nation-state censor, as we discuss in the next section.

To our knowledge, there have yet to be experiments on the resources needed by a relay station that performs in-line blocking to check steganographic tags and the impact these operations would have on the quality of service for all overt sites accessible through the deployed relay station. Tags need to be checked for every TLS connection, which now comprise over a third of all Internet traffic [1] and require the relay station to perform expensive public key operations. In Section 4, we provide an extensive analysis of the impact of checking Telex tags using specialized hardware. We chose Telex

tags as they are used by multiple systems, including Telex, Slitheen, and Rebound.

Another obstacle in the deployment of decoy routing systems is the prevalence of asymmetric flows. The upstream path from a user to an overt site may pass through a relay station, but the downstream path may take a different route and miss the relay station targeted by the user's tag. Of the seven existing decoy routing systems, only Cirripede, TapDance, Rebound, and Waterfall support asymmetric flows. With these systems, as long as the user's traffic passes through a relay station on the upstream (or downstream, in the case of Waterfall) path to the overt site, the relay station can effectively deliver covert content to the user. However, as we discuss in Section 3, second-generation asymmetric solutions have significant flaws that could allow a passive censor to identify their usage. While Waterfall has strong security properties, the registration protocol is prone to denial of service and blockage by a censor. Our solution presents an alternative to client registration as well as a solution to the relaying of upstream covert data from the client to the relay station that places less strain on overt sites.

For Telex, Curveball, and Slitheen, the relay station has to see both upstream and downstream traffic of a tagged session. Our solution can be applied to all previously symmetric systems to recognize and use asymmetric flows for the delivery of covert content. We use a gossip protocol for deployed relay stations to share information about potential steganographic tags. We provide an overview of the deployability features and security properties of existing systems in Table 1. The previously symmetric systems Telex, Curveball, and Slitheen are analyzed both in their original form and along with our improvements to support routing asymmetry.

3 Routing Asymmetry

Traffic between a client and an overt site often takes a different route, passing through different routers or ASes, in the upstream and downstream directions. Past studies have found somewhere between 80% and 90% of routes to be asymmetric [15, 22, 33]. This asymmetry becomes more prevalent in the centre of the network. John et al. [22] found that only about 10% of flows are symmetric in Tier-1 networks (i.e., the backbone of the Internet), while flows at the edge of the network are symmetric about 70% of the time. The ability of a decoy routing system to work in the presence of asymmetric flows enhances the system's deployability by increasing the effectiveness of deployed stations and lowering the number of relay stations that must be deployed to defend against routing-capable adversaries. Each individual relay station can intercept traffic meant for a larger number of overt sites. Four of the existing decoy routing systems accomodate routing asymmetry. Cirripede [18], TapDance [39], and Rebound [11] function properly if a user's traffic passes through a deployed relay station only in the upstream direction towards the overt site, but each has security issues or drawbacks, which we outline next. Waterfall [31] takes a different approach, placing relay stations only on the downstream path from the overt site to the user.

Cirripede accomplishes routing asymmetry by handling client registration (i.e., recognizing that a client wishes to begin a decoy routing session) solely through the passive observation of TCP SYN packets. These packets are sent from the client to the overt site at the start of every connection. After recording the ISNs from 12 of the client's TCP connections, they make a rule in their routing table to divert all traffic from the client's IP address to a service proxy for a fixed period of time. During this time, as long as the client's traffic passes through this router in the upstream direction towards any overt site, it will be redirected to a service proxy that will relay data to and from the client and a covert site. Downstream data from the covert site is sent directly from the service proxy to the client, eliminating any need for a relay station to be placed downstream.

On the usability side, a disadvantage of this approach is that all of a client's traffic will be redirected to the service proxy during the fixed time set by the relay station. If a client wishes to browse a site normally, they must wait for the duration of the decoy routing session to end. There is also a security vulnerability due to the fact that traffic between the user and the covert site does not follow the same downstream path it normally would in a connection to the overt site during the proxy phase. If the overt site and the covert site are significantly far apart, a censor could easily notice a significant difference in latency or in where the traffic enters their network to identify decoy routing sessions.

TapDance implements asymmetry by waiting for the client and overt site to complete the TLS handshake before initiating the tagging procedure. The first upstream HTTP GET request from the client contains a tag in the ciphertext that gives the relay station the client's public key and the encrypted TLS master secret for the session. After retrieving the TLS master secret, the relay station can decrypt upstream data from the client and establish a connection to the covert site. It

then sends covert data to the client directly, encrypting it with the TLS master secret and assuming the role of the overt server. Unfortunately, the non-blocking nature of TapDance and its inability to block or modify downstream traffic leaves the system vulnerable to active attacks by an adversarial censor. Because the relay station is sending traffic to the client on behalf of the overt site, the TCP sequence numbers for downstream data will differ from the overt site's TCP state. A censor can then replay a stale TCP packet to the overt site, prompting an acknowledgement that reveals the server's true state, inconsistent with what the censor has witnessed. TapDance also suffers from the same passive attack as Cirripede, that stems from the difference in the locations of the relay station and the overt site.

Rebound's asymmetric solution presents a different problem by making traffic vulnerable to attack from a passive adversary. Rebound's upstream-only relays receive necessary handshake information from the client in an encoding method similar to TapDance. After reconstructing the TLS master secret, the relay delivers covert content to the user by encrypting it and sending it as an invalid resource name to the overt server in an HTTP GET request. To maintain a consistent TCP state between the overt server and what a passive censor sees, a client must send a GET request with a length that matches the length of the downstream data she wishes to receive. This results in a nearly equal amount of upstream traffic and downstream traffic, which is a highly atypical traffic pattern for any type of web browsing activity. Furthermore, the ethical implications of sending several bad requests to overt sites makes this technique undesirable.

Waterfall places relay stations on the downstream path between the user and overt site, a technique that allows for much stronger security properties in the proxy phase of the decoy routing session as well as a defense against RAD attacks. The downstream-only asymmetry of Waterfall is made possible by the separate registration protocol between the client and the registration server. The client sends a registration package with a series of identifiers: one for each future decoy routing session the client wishes to establish. The identifiers contain all necessary upstream information a relay station would need to man-in-the-middle the TLS session with the overt site. The registration server disseminates this information to relay stations, which then attempt to decrypt TLS sessions whose client IP address are included in the list of registered clients, using the connection identifier information provided. This registration process provides a usability advantage over Cirripede: clients can choose which of their subsequent flows are to be decoy sessions and which are regular browsing sessions. However, an attacker could perform a denial of service attack against suspected clients by registering a series of identifiers in their name. It is unclear how such conflicts in registration would be solved. Furthermore, the connection between the client and the registration server could be censored, requiring the client to adopt a different censorship circumvention system to make this initial connection.

The proxying of covert information to the user in Waterfall is very similar to Slitheen: overt resources are replaced in a manner that perfectly imitates the loading of an overt site. However, upstream information is bounced off of the overt site to the downstream station in a manner similar to Rebound. They suggest several methods for bouncing covert data off of the overt server, including HTTP 404 messages and HTTP redirects, the latter of which are quite common in normal web-browsing behaviour.

In this section, we describe a solution to achieve asymmetry in previously symmetric decoy routing systems such as Telex, Curveball, or Slitheen, that maintain the security properties of these systems. Our solution can also be used as an alternative to the registration protocol of Waterfall and as an alternate way to relay upstream information to the downstream relay station, and maintains the same RAD-resistance as Waterfall due to the focus on the downstream half of the flow.

We position easily deployable, non-blocking relay stations (which are really just simple taps) in the upstream half of a connection from a user to an overt site to gossip ClientHello random nonces to possible downstream relay stations that may be able to recognize a tag. As this random nonce is the only upstream part of the TLS handshake a relay station needs to compute the TLS master secret, the downstream station only needs this small amount of gossipped information—and not necessarily in real time—to successfully use that and subsequent flows for decoy routing. During the proxy phase of the decoy routing session, these gossip stations also relay upstream information to nearby downstream stations.

3.1 Asymmetric Gossip Protocol

Our solution for asymmetric decoy routing takes a slightly different approach from existing solutions. We require the existence of a relay station in both the upstream half of the flow (on the path from the client to

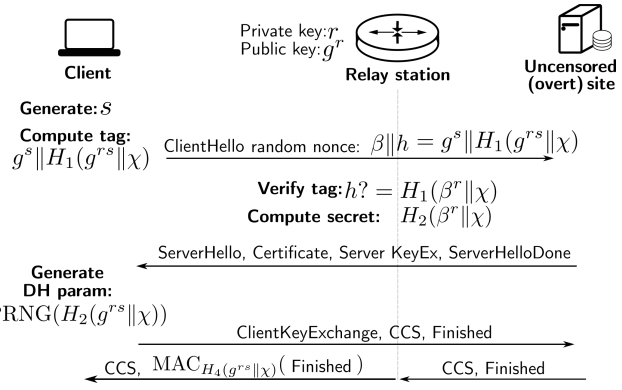


Fig. 2. Modified TLS handshake for tagged flows. We define the context string χ as `server_ip`$\|\rho$, where ρ = `ClientHello random`[0..3]. For a full comparison of the original TLSv1.2 protocol to these modifications, see Appendix A.

the overt site), and the downstream half (on the path from the overt site to the client). These relay stations do not need to be placed on the same router, or even in the same AS. The relay station in the upstream path requires only an extremely lightweight non-blocking network tap and aids in both registration and the proxying of upstream information. This tap removes Waterfall's need for a client to connect to a registration server, provides a DoS-resistant way to tag decoy sessions, and reduces the load on the overt site by directly sending upstream covert data to the downstream station.

Our focus on the downstream relay station comes from the fact that a relay station only needs to observe one upstream handshake message to compute the TLS master secret: the ClientHello message that contains the steganographic tag in its random nonce. However, the relay station needs to see multiple downstream handshake messages: the ServerHello, ServerKeyExchange, and (in the case of Slitheen), the downstream Finished message. To minimize the communication between two relay stations on either side of the flow, the upstream relay station "gossips" received ClientHello messages to other known relay stations, in an attempt to reach a relay station on the downstream path.

This approach spans multiple flows between a client and an overt site and therefore requires route *stability*, in which although each flow is routed asymmetrically, the routers traversed in each direction do not vary significantly between the same two endpoints. There is evidence that routes are highly stable; a 2009 study by Schwartz et al. [33] compared the routes taken between between over 10,000 sets of endpoints with an average of about 100 measurements for each pair over the course of four days. Analysis over this time frame is more than sufficient for the purposes of our system. They found that most pairs of endpoints had a dominant route, or one in which over half of the traffic between these pairs traversed. 25% of pairs had absolute stability where all traffic (although possibly asymmetric) always crossed the same routers. Furthermore, the number of distinct routes for endpoints that did experience variance was usually small: only about 20% of all endpoint pairs had over 20 distinct routes, and very few had over 60. This leads us to believe that there is a high probability that subsequent flows between the same client and overt site will cross the same downstream relay station, particularly in the short term due to load-balancing practices.

We first describe the tagging phase of our asymmetric solution using the modified TLS handshake used by Slitheen for tagging flows. We follow this with a discussion of asymmetry in the relay, or proxying, phase of the decoy routing session.

3.1.1 Asymmetric Tagging

We use the Slitheen tagging protocol as it is a slightly modified, updated version of Telex's tagging procedure, differing only in the handling of TLS Finished messages, as shown in Figure 2. This method varies only slightly from that used by Curveball, in which the client and the relay station share a symmetric secret that was exchanged out of band.

In Slitheen, a steganographic tag is placed in the last 28 bytes of the 32-byte random nonce of the TLS ClientHello handshake message. The first 4 bytes are reserved and usually randomly generated; however, some older implementations of TLS used them as a timestamp. Our solution uses an implementation that randomly computes these bytes, and we will refer to this random value as $\rho =$ `ClientHello random`[0..3] for ease of reference in the rest of the paper.

To enable the downstream relay station to compute the TLS master secret from the previous ClientHello random nonce and the server's TLS handshake messages, we make one further small modification to the tag. In Telex and Slitheen, the tag and the client's TLS key exchange parameters are computed from the client-relay shared secret as well as a context string χ, where $\chi =$ `server_ip`$\|\rho\|$`TLS_session_id`. In our asymmetric setup, the downstream relay station is responsible for intercepting the flow and may not have access to the TLS session id (as this may or may not be reflected in the ServerHello message, depending on the session's resumption status). Therefore, we use a different con-

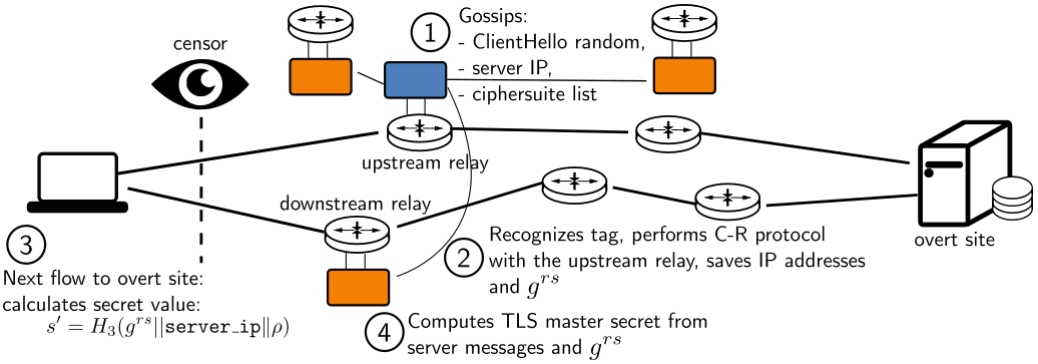


Fig. 3. Gossip protocol for symmetric flow tagging. The user tags a connection to the overt site for a relay station positioned in the *downstream* half of the flow. When a relay station sees a ClientHello message in an upstream flow for which it has not seen the downstream SYN|ACK packet, and does not recognize a tag in the random nonce, they gossip this nonce, along with the server IP address and the proposed list of ciphersuites, to nearby relay stations (1). Each of these relay stations checks the gossipped nonce for a tag using its own private key. If it was tagged for them, and they observe the downstream half of the flow, they perform a challenge-response protocol with the upstream station to receive the client IP address, and wait for a future TLS session between the client and the overt site to begin (2). When the user next makes a connection to the same overt site (3), they will generate the new client exponent as a hash of the previous client-relay shared secret $s' = H_3(g^{rs}\|\texttt{server\_ip}\|\rho)$. The downstream relay can reconstruct the client exponent and the new ClientHello random nonce themselves. After seeing the server's handshake messages, the relay can compute the TLS master secret (4) and replace downstream content. After the tagging phase, the upstream relay station will continue to send copies of the upstream data to the downstream server, however, this communication is not time critical.

text string $\chi = \texttt{server\_ip}\|\rho$ that depends only on the server's IP address and the first 4 bytes of the ClientHello random nonce. Removing the TLS session ID from the context string will not affect the security of the scheme as an adversarial censor is still unable to perform a tag replay attack. In this attack, a censor would observe a suspect flow and then initiate a TLS connection to the same overt site, reusing the suspect ClientHello random nonce in the hopes of observing their own decoy routing session. Such an attack would require the censor to generate the correct TLS session key matching the tag without the client or the relay private secret, which violates the security of public-key cryptography.

We give an overview of our asymmetric solution in Figure 3. A client begins an asymmetric decoy routing session by generating a random secret s and composing the steganographic tag $g^s\|H_1(g^{rs}\|\chi)$ for the ClientHello random nonce using the public key, g^r, of a relay station in the downstream half of the flow. Each relay station has its own public key and these are distributed along with the client-side software to the client. The client computes their secret exponent in the key exchange part of the TLS handshake from the client-relay shared secret, g^{rs} by seeding a secure pseudo-random number generator with $H_2(g^{rs}\|\chi)$. When a relay station in the upstream half of the flow receives a ClientHello message for which it does not recognize the tag, and for which it has not seen the SYN|ACK packet for the flow (indicating routing asymmetry), it gossips the ClientHello random nonce along with the flow's context information (i.e., the server IP address and ρ), as well as ciphersuite information, over encrypted connections to nearby relay stations that could possibly be on the downstream path of the flow. If a relay station receives this information and recognizes the tag, it performs a challenge-response protocol with the upstream station to prove that it recognized the tag and receive all further records in that flow. We discuss this challenge-response protocol further in Section 3.1.3. It then saves the client and server IP addresses in a table along with the client-relay shared secret g^{rs}, and waits for future connections from the client to the same overt site. We emphasize that this gossipped message does *not* need to be received by the downstream station before the overt site responds to the ClientHello message. If it is late, it simply acts as a registration step, allowing the downstream station to successfully use the *next* asymmetric connection between the client and the same overt site.

To compute the TLS master secret for a decoy routing session, the relay station needs three values: 1) the premaster secret, computed from the tag in the ClientHello random nonce and the server's public key in the ServerKeyExchange message, 2) the ClientHello random nonce, and 3) the ServerHello random nonce. In the event that the downstream relay station receives the tag and flow information before the overt site has sent the ServerHello message of the TLS handshake, it can proceed to compute the TLS master secret for the

current session. If the downstream station has missed the ServerHello message by the time the gossip protocol completes, it waits for the next connection from the client to the same overt site.

The next time a client makes a connection to the same overt site, the client computes the new secret exponent used to construct the steganographic tag as the hash of the previous client-relay shared secret and the IP address of the overt site: $s' = H_3(g^{rs} \| \texttt{server\_ip} \| \rho)$ where the first 4 bytes of the ClientHello random nonce, ρ, are generated from the previous shared secret g^{rs}.[1] They then place their tag, $g^{s'} \| H_1(g^{rs'} \| \chi)$, in the ClientHello random nonce of the new TLS session along with the deterministically generated first 4 bytes. When a downstream relay station receives the server handshake messages, they extract the ServerHello random nonce, ServerKeyExchange parameters, and compute the client's secret exponent and the ClientHello random nonce from the saved client-relay shared secret, g^{rs}, and the server IP address.

After computing the TLS master secret for the session, the relay station attempts to decrypt the downstream TLS Finished message. If the decryption is successful, it replaces the hash of the Finished message, `finished_hash` with $MAC_{H_4(g^{rs'} \| \chi)}(\texttt{finished\_hash})$. When the client receives the Finished message, they will compute the keyed MAC of the unmodified TLS Finished message and compare the result with the received value. If they received an unmodified Finished message, the flow was not successfully intercepted by a relay station. If they received the keyed MAC, they know the flow has been intercepted and a decoy routing session has begun.

3.1.2 Asymmetric Proxying

After the TLS handshake, the downstream relay station begins to proxy information between the client and a covert site. All three symmetric systems rely on upstream data from the client in order to establish a connection to a covert site and relay upstream data from the client to the covert site. We note that in this stage, the amount of upstream data from the client to the covert site is typically far less than the downstream covert data. To retrieve covert data from an upstream relay station, the downstream relay station will perform a challenge-response protocol with the upstream station, proving the session has been tagged for their private key and signalling that they wish to receive TLS application data from the upstream half of the flow. If successful, the upstream station will proceed to funnel upstream TLS records (over a point-to-point encrypted and authenticated connection) to the downstream station, which then decrypts the TLS records and proceeds in the usual manner. The sending of these upstream TLS records has no time constraints; they can arrive at the downstream station asynchronously with downstream data from the covert site or (in the case of Slitheen) the overt site. Any delay in the receipt of this data will not affect the security or correctness of the system, but only the latency experienced by the client in their browsing of covert content. The downstream station will make a connection to the covert site specified by the client and send the client's upstream covert data through this connection. Telex and Curveball will then deliver downstream covert data directly to the client, while Slitheen will insert it into downstream leaf resources.

3.1.3 Challenge-Response Protocol

We require the downstream relay station to perform a challenge-response protocol with the upstream gossip station in order to receive 1) the client information necessary to recognize future tagged flows, and 2) the upstream TLS records during the proxy phase of the decoy routing session. The reason for this requirement is to mitigate denial-of-service attacks on upstream stations and protect the privacy of both tagged and untagged traffic that passes through each upstream relay station. While the amount of additional data leakage in our gossip protocol is small (all ASes on the path between the client and the overt site have access to the same information), this prevents the usage of our decoy routing system by adversaries in expanding their ability to perform mass surveillance on Internet metadata.

First, to prove to the upstream station that they recognize one of the gossipped ClientHello tags, the downstream station uses the gossipped tag, context string information (i.e., the server IP and ρ), and ciphersuite information (i.e., the list of client-proposed cipher suites that the decoy routing system supports as well as valid elliptic curves if applicable) and computes all possible client key exchange parameters for those ciphersuites. Note that in current implementations of decoy routing systems this is at most six different sets of parameters. The downstream station then sends hashes of these

1 The OUS should therefore be a browser whose TLS implementation uses random data instead of a timestamp in that field.

Table 2. Estimates of the number of deployed downstream and upstream stations needed to evade censorship for China, a highly connected, routing-capable adversary. We use results from Houmansadr et al. [20] to estimate a necessary 880 upstream stations to resist RAD attacks and results from Nasr et al. [31] to estimate a necessary 5 downstream stations.

System	Heavy-weight stations	Light-weight stations
Symmetric designs [4, 18, 24, 40]	880	N/A
TapDance [39]	0	880
Waterfall [31]	5	0
Gossip protocol + any symmetric design	5	880

key exchange parameters to the upstream station. The upstream station compares these hashes with the hash of the key exchange parameters in the client's key exchange message. If one of them matches, they then send the connection information (i.e., the client IP address) to the downstream station so that they can recognize future tagged flows.

The downstream station must perform the above challenge-response protocol for each subsequent TLS session that the client sends to the overt site in order to receive the upstream TLS records. Because the downstream station can compute the key exchange parameters for future sessions ahead of time, they can send multiple sets of parameter hashes to the upstream station at once. Then, the upstream station can immediately forward upstream records as soon as the client sends a key exchange message whose parameters hash to a matching value.

3.2 Resistance to RAD attacks

The placement of decoy routers at ASes is critical for providing censorship resistance to users within censoring regions. Schuchard et al. [32] were the first to acknowledge that the number of decoy routers necessary to evade censorship in the presence of a routing-capable adversary is much greater than previous estimates. Since the introduction of RAD attacks, there have been many proposals for the optimal placement of decoy routers [5, 20, 30, 32]. Although it is unrealistic that all ASes will be willing to deploy our system, these proposals provide an idea for how many decoy routers will need to be deployed to provide censorship resistance for different nation-states. We draw on the findings of previous work to give an estimate on the number of heavy-weight downstream and light-weight gossip stations needed to resist censorship for China (a highly connected routing-capable adversary).

The placement of downstream decoy routers was investigated by Nasr et al. [31] in their analysis of Waterfall. They found that it is much more difficult and expensive for adversaries to route around downstream stations, and as a result fewer deployments were needed. Only one deployed decoy station impacts almost a quarter of the traffic from Chinese users, while 5 deployed stations impacts 78% of the routes.

It is much easier for an adversary to route around upstream decoy stations. We used the results from Houmansadr et al. [20] to estimate the number of gossip stations needed. Their results show that if decoy stations are placed at 3% of ASes (outside of China and its ring ASes), 40% of the Internet becomes unreachable for Chinese users, meaning it is not possible for China to avoid all deployed stations without cutting off access to 40% of the Internet. This requires the placement of roughly 880 gossip stations. Table 2 gives a comparison of the number of necessary deployments to previous systems. We note that while we require more deployments than both TapDance and Waterfall, our gossip stations are even more deployable than TapDance stations (which have been successfully deployed [13]): we require no intensive computations in our upstream stations to check for tagged flows.

Our asymmetric solution in this section provides a more secure alternative to previous proposals for the asymmetric deployment of decoy routing systems. Our methods can be easily integrated into Waterfall, providing a more secure alternative to client registration and a method for relaying upstream covert data in a manner that is kinder to overt sites. The tiered deployment made possible by our approach presents a cost-effective way for hesitant ISPs to participate in censorship resistance without the need for hardware that can perform in-line blocking or traffic replacement.

3.3 Bandwidth Overhead

The bandwidth overhead of the gossip protocol is small compared to the existing load of routers. We also note that the upstream stations do not need to perform in-line blocking, drastically lightening the load compared to previous symmetric systems. The overhead has three parts: (1) that induced by gossipping the ClientHello data that passes through the router to a set of known relay stations, (2) the challenge-response protocols between the upstream and downstream stations,

and (3) that of funnelling the upstream TLS application records of proven tagged flows to the downstream station. The gossipped data consists of the ClientHello random nonce, the server IP address, and the list of supported ciphersuites and supported elliptic curves. Its size is dependent on the number of ciphersuites supported by the client. Using Firefox, we measured the average gossip data size to the Alexa top 100 sites as 66 bytes. Note that there was almost no variation in the ciphersuites offered by the client in the version of Firefox we were using. Using traffic measurements from the Center for Applied Internet Data Analysis (CAIDA) [1] shown in Table 3 in the next section, we calculate the bandwidth overhead of gossipping ClientHello messages as

$$1 + \frac{66 \cdot 4430n}{125000 \cdot 2035.71} = 1 + 0.0011n$$

where n is the number of relay stations gossipped to. If, for example, we set $n = 5$, the number of downstream routers sufficient to defend against a highly connected adversary such as China, the overhead is only $1.0055\times$ the total bandwidth through the router of the deployed relay station. To give concrete numbers, for a router on a typical OC48 link of a large ISP that handles approximately 2 Gb/s of traffic, the router would have to transmit an extra 11 Mb/s of gossip data.

The challenge-response protocol requires the downstream station to send the upstream station a maximum of six 32-byte hashes for each TLS session, given current implementations of decoy routing systems. The upstream station responds with a 4-byte client IP address. The total amount of data exchanged for a single-session asymmetric decoy routing session is then 196 bytes. As the base rate of decoy routing flows is very low, this number is negligible in the calculation of the overhead.

To calculate the bandwidth of the proxy phase of the gossip protocol, we measured the average bandwidth of upstream TLS application data to the Alexa top 100 sites. Note that this data is only gossipped for flows that are tagged for a downstream station, which do not likely make up the majority of traffic through a relay station. In our CAIDA data set, the proportion of all data that is upstream data in TLS flows is 0.042. The overall bandwidth overhead is then $1 + 0.0011n + 0.042\beta$ where β is the base rate of tagged TLS flows. Note that for $\beta < 10^{-3}$, the overhead induced by copying upstream data is negligible, resulting in a total overhead of only a few percent.

To compare the bandwidth cost of relaying upstream information to the downstream relay during the proxy phase of the session to Waterfall, which has a similar requirement, our approach requires strictly less additional traffic. Our approach sends upstream records directly, while Waterfall requires them to be wrapped in appropriately sized HTTP GET requests. It is important to note that while our approach requires less traffic overhead, it does require more effort from the system to determine which bytes to forward, perform the challenge-response protocol, and tunnel upstream traffic, though the demands we place on the upstream station are less than those required by TapDance, which is already shown to be deployable [13]. Importantly, our approach places zero additional load on unsuspecting overt sites.

4 Relay station experiments

Wustrow et al. [39] found the main obstacle in convincing ISPs and ASes to deploy decoy routing systems to be the resource requirements of existing systems in checking tags and performing in-line blocking. By checking every TLS session for steganographic tags, the deployment of decoy routing systems also has the potential to affect the quality of service for all customers whose traffic traverses a relay station.

We performed several experiments to determine the impact a deployed decoy routing station would have on existing traffic in a real-world scenario. Note that these experiments measure the cost of the heavyweight *downstream* relay stations, of which fewer need to be deployed to defend against routing-capable adversaries. The cost of an upstream gossip station in terms of its impact on quality of service is nonexistant as the station does not perform in-line blocking of flows.

Our first set of tests aims to measure the effect tag checking would have on the quality of service for both TLS and non-TLS traffic of regular customers. We chose the Slitheen tagging procedure for our measurements, as the Slitheen modified TLS handshake requires the most effort from a deployed relay station and the Slitheen source code is freely available.[2]

For our tests, we used specialized (but off-the shelf) hardware capable of performing in-line blocking and efficient deep-packet inspection. Our reasons for doing so were that 1) only TapDance does not require in-line blocking, and this feature also introduces several vulnerabilities that an active attacker can exploit to easily differentiate decoy routing sessions, and 2) by showing

2 https://crysp.uwaterloo.ca/software/slitheen/

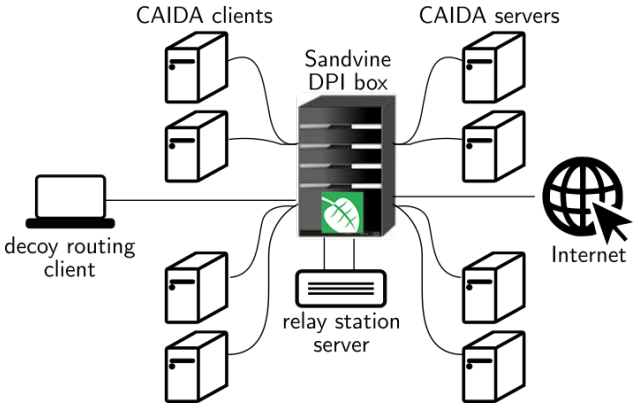


Fig. 4. The network topology of our experiments. Machines designated as CAIDA clients and CAIDA servers were dedicated to sending background traffic through the PTS, representative of traffic sent through an OC48 link of a large ISP according to statistics gathered from CAIDA [1].

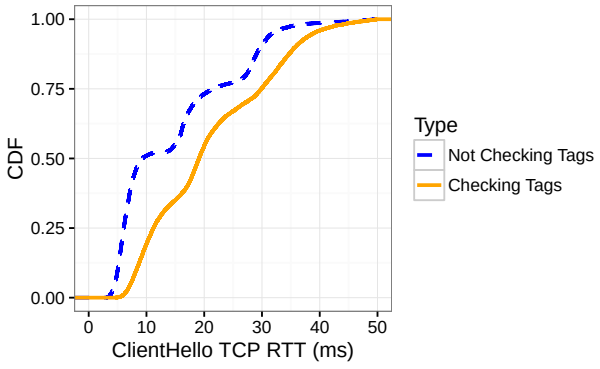


Fig. 5. CDFs comparing the TCP round trip time (RTT) for TLS ClientHello packets with tag checking on and off.

the capabilities of existing hardware to efficiently perform decoy routing tasks we can target existing users of this hardware as the first to deploy a decoy routing system. The relay station itself consists of two parts: a Sandvine Policy Traffic Switch (PTS) 22600 capable of performing deep-packet inspection and flow diversion, and a relay station server with two 10 Gb/s connections to the PTS. If a tagged flow is detected by the PTS, it is diverted to the relay station server. The relay station server and client machine each used 8 cores and 2 GB of RAM. The PTS is responsible for routing all traffic and checking the tags of TLS flows. If a flow is tagged for the relay station's public key, the PTS then diverts the flow through the relay station server, which performs the rest of the tagging procedure during the TLS handshake and handles the proxy phase of the decoy routing session. We provide an overview of our experimental set up in Figure 4.

For our tests, we gathered distribution statistics for Internet traffic from the Center for Applied Internet Data Analysis (CAIDA). We used the anonymized passive trace statistics through an OC48 link belonging to a large ISP in Chicago with a maximum load of 10 Gb/s. We calculated the average flows/s and average Mbit/s for 5 major types of flows: HTTPS, HTTP, DNS, generic TCP, and generic UDP, gathered over the course of an hour on April 6th, 2016 [1]. For each test, we sent a CAIDA-representative amount of traffic through the deployed relay station using four client machines and four server machines, on opposite sides of the PTS. To test the validity of our experiments, we measured the flow rate and bit rate for each type of flow at the client and server endpoints for 100 3-second captures. The results are given in Table 3.

Table 3. Distribution statistics (CAIDA / our experiments)

Flow type	Average flows/s	Average Mb/s
HTTPS	4.43k / 4,330 (± 90)	848.81 / 840 (± 20)
HTTP	4.07k / 4,040 (± 80)	814.51 / 800 (± 20)
DNS	2.26k / 2,200 (± 100)	N/A / N/A
TCP	792.74 / 790 (± 20)	246.52 / 250 (± 10)
UDP	527.53 / 530 (± 20)	125.87 / 128 (± 7)
Total	12.08k / 12,000 (± 100)	2035.71 / 2010 (± 40)

4.1 Impact on quality of service

Deployed relay stations must check every incoming TLS ClientHello random nonce for a steganographic tag. This requires a public key operation to compute the client-relay shared secret from the first 21 bytes of the nonce, and then a hash of the shared secret with a context string. This hash is then compared to the last 7 bytes of the 28-byte random nonce. These operations take time, and we sought to measure the latency they add to TLS flows.

To do so, we made 1000 sequential, untagged TLS handshakes to the Alexa top 1000 TLS sites and measured the time between when the client sent the ClientHello message to the time it took for the client to receive the TCP acknowledgement of the message. We tested two conditions: one where the PTS checked ClientHello messages for tags, and one in which the PTS did not check for tags. For each of these tests, we ran a CAIDA-representative amount of background traffic through the relay station, as described above. We present the results in Figure 5. Although deploying a relay station has an impact on the latency of ClientHello messages, the aver-

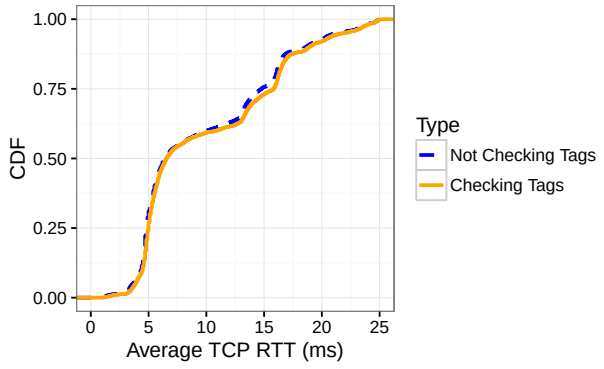
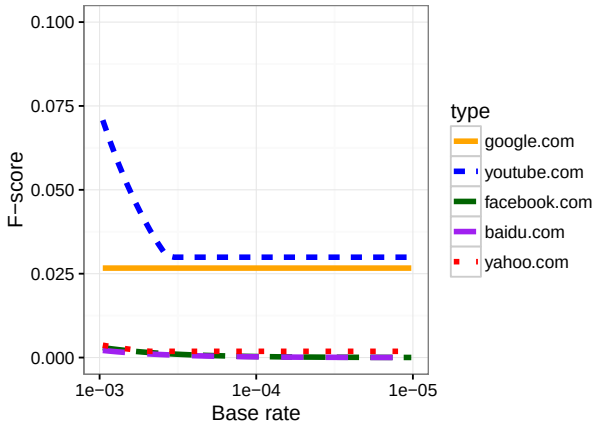
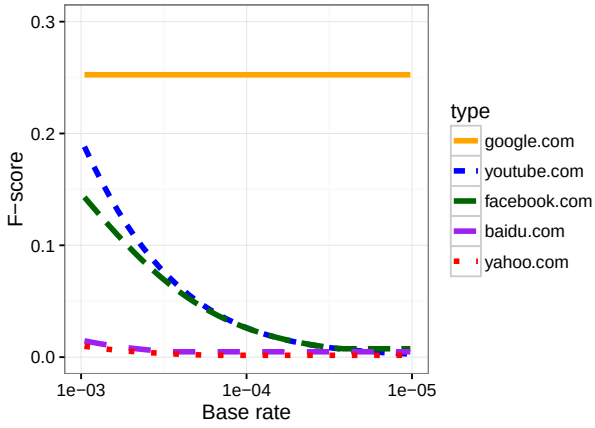


Fig. 6. CDFs comparing the average TCP round trip time (RTT) for non-TLS flows with tag checking on and off.



(a) The F-score of classifying flows based on TLS handshake time.



(b) The F-score of classifying flows based on the average TCP round trip time (RTT).

Fig. 7. The maximum F-score (i.e., the harmonic mean of precision and recall) a censor can achieve in classifying flows as decoy routing sessions or as regular accesses to the Alexa top 5 sites. Precision is dependent on the base rate of decoy routing sessions. As a result, the more prevalent decoy routing sessions are, the higher a censor's accuracy in classifying flows.

age additional latency is 7 ms, which is very low and falls within the standard deviation of each condition (10 ms).

In addition to measuring the impact a deployed relay station has on the quality of service for TLS flows, we also measured the impact it has on non-TLS flows and whether our equipment and software introduced any additional latency by performing deep-packet inspection to search for ClientHello messages. We performed a similar test as above, this time making 1000 HTTP connections to remote sites for each condition. For each connection, we calculated the average RTT of all TCP packets in the flow. The results are given as CDFs in Figure 6. The additional latency of deploying a relay station was 0.4 ms, which is very low, and falls within a standard deviation of each condition (10 ms). We note that at this time, the Slitheen tag checking and relay station code has not been optimized for quality of service. With further improvements, the results in this section for both TLS and non-TLS flows will likely show an even lower impact on the customers of participant ISPs.

Our results show that while the deployment of a full downstream relay station adds additional latency to flows due to checking for tags in ClientHello random nonces, but the latency introduced is quite small.

4.2 Defenses against latency analysis attacks

The security properties of Slitheen rely on the inability of a censor to detect additional latency added by the relay station in checking tags or replacing content from the overt site. We conducted tests to see whether the divert functionality of the PTS and the implementation of the relay station added enough latency to tagged flows to allow a censor to reliably classify them as decoy routing sessions. We simulated an attack in which the censor compiles a database of expected latencies for both decoy sessions and regular browsing sessions for each overt destination by making 100 connections to the top 5 Alexa sites for each condition. We then calculated the precision and recall an adversary could achieve in classifying flows as decoy routing or regular sessions.

We measured two different types of latency for each flow: the time it took to perform a full TLS handshake, and the average TCP acknowledgement time, or round-trip time (RTT) for application data. A censor will attempt to select a cut-off latency for each measurement type to identify decoy routing sessions. All flows with a higher latency than the cut-off value are classified as de-

coy routing sessions, while all flows with a lower latency are classified as a regular access to the overt site. We computed the CDFs of each type of latency for decoy routing sessions and regular accesses to each of the top 5 sites. From these CDFs, we can compute the true negative rate, τ, and false negative rate, ϕ, of an adversary for each possible latency cut-off point. We calculate the *precision* of the censor as:

$$\text{precision} = \frac{\beta(1-\phi)}{\beta(1-\phi)+(1-\beta)(1-\tau)}$$

where β is the base rate of the incidence of decoy routing sessions. A typical censor would try to maximize precision, thereby minimizing the number of regular accesses to the overt site that are mistakenly identified as decoy routing sessions and blocked. By maximizing *recall*, a censor ensures that they are identifying and blocking as many decoy routing sessions as possible. The recall of the censor is calculated as the true positive rate: $\text{recall} = 1 - \phi$.

A censor can achieve a precision of 1, indicating that they do not incorrectly classify any regular accesses to the overt site as decoy routing sessions. However, often this means the censor can only identify a very small portion of decoy routing sessions while the majority will continue undetected. They can also achieve a recall of 1 by simply classifying all connections as decoy routing sessions and blocking them. For most censors, both measures are important so we define a censor's accuracy in terms of its F-score, the harmonic mean of the precision and recall values. Precision and recall can be weighted differently according to the individual goals of the censor, but we consider the default equal weighting in this paper. For each value of β, the adversary will select a cut-off value that maximizes their F-score, given the latency distributions of each overt site. We plot the maximum F-score values based on the latency distributions for both the TLS and handshake time and the TCP RTT for five of the Alexa top sites in Figure 7.

The maximum accuracy a censor can achieve in identifying decoy routing sessions is very low for both types of latency. For the majority of sites, this value drops to almost 0 with a base rate of occurrence of decoy routing sessions of less than 10^{-4}, and meaning that if no more than one in every 10,000 connections to popular sites are decoy routing sessions, a censor is unable to reliably determine whether or not any given flow is carrying censorship resistance traffic. Even with a higher occurrence of decoy routing, the maximum F-score stays below 0.5 for most sites, making a reasonable censor that is unwilling to upset their population extremely wary of classifying and blocking potential decoy routing sessions. We note that some sites exhibit anomolous behaviour (e.g., google.com and youtube.com in their TCP RTTs and TLS handshake times, respectively). Such behaviour can be measured by the client, and those sites not selected as overt sites.

5 Security analysis and improvements

Our proposal to add asymmetry to previously symmetric decoy routing systems has two main advantages: it has better security properties than previously proposed asymmetric systems, and it provides a path for tiered deployment, creating a less expensive defense against routing-capable adversaries. We provide a comparison of the security properties of existing systems and our suggested improvements to previously symmetric systems in Table 1.

TapDance remains the only system capable of performing decoy routing without requiring a relay station to block or modify traffic. However, this feature comes at the cost of security. We believe a better route to deployment is by providing ISPs and ASes with experimental evidence of the impact a deployed relay station would have on customer traffic using existing hardware capable of performing tag checks efficiently and blocking or modifying tagged flows. By targeting ASes that already own this hardware or showing them a clear path to deployment, we are providing more evidence that decoy routing is an attainable option and moving towards real-world deployment. Furthermore, our asymmetric solution does not require in-line blocking for upstream relays, enabling more cautious potential participants to provide a stronger defense against RAD attacks.

5.1 Security of the Gossip Protocol

While the gossip protocol does not leak any additional information of tagged or untagged flows to an adversary, it does increase the number of routers that see traffic between the client and the overt site, possibly increasing the ability of a passive adversary to perform traffic analysis or surviellance attacks. However, gossipped messages do not significantly increase a censor's ability to detect the usage of censorship circumvention tools or attribute them to individual users. In this section, we discuss the impact of the gossip protocol on the security and privacy of both users of Slitheen and non-users

whose upstream handshake messages are gossipped to other relay stations.

Passive adversary: The gossip protocol requires an upstream station to send all seemingly untagged ClientHello messages, along with the upstream TLS application data to the downstream relay station. Note that the gossipping of ClientHello messages is done for all TLS handshakes that the upstream station does *not* recognize as tagged and includes both untagged and potentially tagged flows; therefore the gossipped messages do not expose censorship resistance traffic.

It is worth noting that the gossipping of application data to the downstream relay station only happens for tagged flows. A censor that can see traffic between relay stations could then perform a timing analysis attack to connect outgoing connections to gossipped messages. This is outside our threat model as we assume relay stations exist outside of the censor's area of influence and therefore probably do not cross through a censor's control. It is also practically difficult to correlate the TLS application records of any one flow to the encrypted traffic sent between two relay stations. Approximately 37% of flows are HTTPS [1], meaning that a censor observing traffic on even a small router would have to decide which of the thousands ($1/\beta$) of TLS sessions that data corresponded to.

Malicious relays: Our challenge-response protocol in Section 3.1.3 prevents a malicious downstream relay from lying about recognizing tags in order to induce extra load on the gossip station in a denial-of-service attack, or to increase their surveillance of flows outside of their usual field of view. However, precautions should be taken to prevent a censor from pretending to be an upstream station in order to use downstream stations as oracles to identify tagged flows. Downstream stations should maintain a list of approved and trusted upstream stations, as well as their public key information. This information can be updated by relay station operators as new upstream stations are deployed in much the same way as the client software maintains a list of public keys for trusted downstream stations.

5.2 Superencryption of application data

Severing or abandoning the connection with the overt site in the proxy phase of the decoy routing session introduces a vulnerability in which the server's state of the connection does not match the traffic that a censor sees. The censor can exploit this in most systems using a RAD attack or a regular TCP replay attack. Slitheen [4], Rebound [11], and Waterfall [31] avoid this vulnerability by interacting with the overt site throughout the proxy phase, modifying the contents of the TLS encrypted data to give covert data to the client. However, this process of modification introduces a new vulnerability to systems that use the same TLS master secret to re-encrypt the application-level data [4, 11].

During the re-encryption of new application data, if the relay station re-uses the same nonce, an adversary capable of seeing the data on both sides of the relay station could use it to decrypt or modify the data between the user and the covert site. Although this attack falls outside the usual threat model for decoy routing, in which we assume that the censor is unable to compare traffic on both sides of the relay station, this puts vulnerable users of censorship circumvention systems at risk. We describe the attack in more detail, and our solution of adding an extra layer of encryption around covert data, in Appendix B.

6 Conclusion and future work

As the Internet becomes more centralized and the capabilities of censors grow, so will their ability to filter Internet traffic with increasingly sophisticated methods. It is possible in the future that as censorship becomes more prevalent, so will the dangers of resisting government controls. There is a dire need for a censorship circumvention system that provides users with blocked content as well as hides their usage of the system. Decoy routing provides a promising solution to Internet censorship. Its strong security properties, and trend of realistically appropriating real, uncensored connections in the place of mimicry have the potential to end the cat-and-mouse game in favour of the resistor.

In this paper, we proposed a new approach to routing asymmetry that provides better security than previous asymmetric systems and a path to tiered deployment that allows for several lightweight, limited systems to surround a powerful censor, limiting the censor's ability to perform routing-based attacks. This work presents the next steps towards the deployment of decoy routing systems, however there is still much work to be done. With more efficient implementations of the relay station, the possible impact of deployment may be even less than what we measured with our limited improvements. We look at our results as a positive indication that decoy routing may prove to be practical in the future and may convince the owners of Internet routers to consider participating in censorship circumvention.

Acknowledgements

The work benefitted from the use of the CrySP RIPPLE Facility at the University of Waterloo. We especially thank Lori Paniak (University of Waterloo) and Dave Dolson (Sandvine) for their technical expertise. The authors thank the Ontario Graduate Scholarships Program for funding Bocovich, and Sandvine and NSERC for grant STPGP-463324.

References

[1] The CAIDA UCSD Statistical information for the CAIDA Anonymized Internet Traces. http://www.caida.org/data/passive/passive_trace_statistics.xml, 2016. Accessed 22-February-2017.

[2] Simurgh Aryan, Homa Aryan, and J. Alex Halderman. Internet censorship in Iran: A first look. In *3rd USENIX Workshop on Free and Open Communications on the Internet (FOCI)*, 2013.

[3] Diogo Barradas, Nuno Santos, and Luís Rodrigues. DeltaShaper: Enabling unobservable censorship-resistant TCP tunneling over videoconferencing streams. *Privacy Enhancing Technologies*, 2017(4):1–18, 2017.

[4] Cecylia Bocovich and Ian Goldberg. Slitheen: Perfectly imitated decoy routing through traffic replacement. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1702–1714. ACM, 2016.

[5] Jacopo Cesareo, Josh Karlin, Michael Schapira, and Jennifer Rexford. Optimizing the placement of implicit proxies. Technical report, Princeton, NJ, USA, 2012.

[6] Roger Dingledine. Obfsproxy: The next step in the censorship arms race. https://blog.torproject.org/blog/obfsproxy-next-step-censorship-arms-race, February 2012. [Online; accessed 29-February-2016].

[7] Roger Dingledine and Nick Mathewson. Design of a blocking-resistant anonymity system. Technical report, 2006.

[8] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *13th USENIX Security Symposium*, pages 303–320, 2004.

[9] Kevin P. Dyer, Scott E. Coull, and Thomas Shrimpton. Marionette: A programmable network-traffic obfuscation system. In *24th USENIX Security Symposium*, pages 367–382, 2015.

[10] Tariq Elahi, John A Doucette, Hadi Hosseini, Steven J Murdoch, and Ian Goldberg. A framework for the game-theoretic analysis of censorship resistance. *Proceedings on Privacy Enhancing Technologies*, 2016(4):83–101, 2016.

[11] D. Ellard, C. Jones, V. Manfredi, W.T. Strayer, B. Thapa, M. Van Welie, and A. Jackson. Rebound: Decoy routing on asymmetric routes via error messages. In *Local Computer Networks (LCN), 2015 IEEE 40th Conference on*, pages 91–99, October 2015.

[12] David Fifield, Chang Lan, Rod Hynes, Percy Wegmann, and Vern Paxson. Blocking-resistant communication through domain fronting. *Proceedings on Privacy Enhancing Technologies*, 2015(2):46–64, 2015.

[13] Sergey Frolov, Fred Douglas, Will Scott, Allison McDonald, Benjamin VanderSloot, Rod Hynes, Adam Kruger, Michalis Kallitsis, David G. Robinson, Steve Schultze, Nikita Borisov, Alex Halderman, and Eric Wustrow. An ISP-scale deployment of TapDance. In *7th USENIX Workshop on Free and Open Communications on the Internet (FOCI 17)*, Vancouver, BC, 2017. USENIX Association.

[14] John Geddes, Max Schuchard, and Nicholas Hopper. Cover your ACKs: Pitfalls of covert channel censorship circumvention. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, CCS '13, pages 361–372, New York, NY, USA, 2013. ACM.

[15] Yihua He, Michalis Faloutsos, Srikanth Krishnamurthy, and Bradley Huffaker. On routing asymmetry in the internet. In *Global Telecommunications Conference, 2005. GLOBECOM'05. IEEE*, volume 2, pages 6–pp. IEEE, 2006.

[16] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. Website fingerprinting: Attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, CCSW '09, pages 31–42, 2009.

[17] Amir Houmansadr, Chad Brubaker, and Vitaly Shmatikov. The parrot is dead: Observing unobservable network communications. In *2013 IEEE Symposium on Security and Privacy*, pages 65–79, May 2013.

[18] Amir Houmansadr, Giang T.K. Nguyen, Matthew Caesar, and Nikita Borisov. Cirripede: Circumvention infrastructure using router redirection with plausible deniability. In *18th ACM Conference on Computer and Communications Security*, CCS '11, pages 187–200, 2011.

[19] Amir Houmansadr, Thomas J Riedl, Nikita Borisov, and Andrew C Singer. I want my voice to be heard: IP over Voice-over-IP for unobservable censorship circumvention. In *2013 Network and Distributed System Security (NDSS) Symposium*, 2013.

[20] Amir Houmansadr, Edmund L Wong, and Vitaly Shmatikov. No direction home: The true cost of routing around decoys. In *2014 Network and Distributed System Security (NDSS) Symposium*, 2014.

[21] Amir Houmansadr, Wenxuan Zhou, Matthew Caesar, and Nikita Borisov. Sweet: Serving the web by exploiting email tunnels. *arXiv preprint arXiv:1211.3191*, 2012.

[22] Wolfgang John, Maurizio Dusi, and K. C. Claffy. Estimating routing symmetry on single links by passive flow measurements. In *Proceedings of the 6th International Wireless Communications and Mobile Computing Conference*, IWCMC '10, pages 473–478, New York, NY, USA, 2010. ACM.

[23] Antoine Joux. Authentication failures in NIST version of GCM. 2006.

[24] Josh Karlin, Daniel Ellard, Alden W Jackson, Christine E Jones, Greg Lauer, David P Mankins, and W Timothy Strayer. Decoy routing: Toward unblockable internet communication. In *USENIX workshop on free and open communications on the Internet*, 2011.

[25] Sanja Kelly, Mai Truong, Adrian Shahbaz, Madeline Earp, Jessica White, and Rose Dlougatch. Silencing the messenger: Communication apps under pressure. https:

//freedomhouse.org/report/freedom-net/freedom-net-2016, 2016. [Online; accessed 28-April-2017].

[26] David McGrew and John Viega. The galois/counter mode of operation (GCM). 2005.

[27] Richard McPherson, Amir Houmansadr, and Vitaly Shmatikov. Covertcast: Using live streaming to evade internet censorship. *Proceedings on Privacy Enhancing Technologies*, 2016(3):212–225, 2016.

[28] Hooman Mohajeri Moghaddam, Baiyu Li, Mohammad Derakhshani, and Ian Goldberg. Skypemorph: Protocol obfuscation for Tor bridges. In *2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 97–108, 2012.

[29] Zubair Nabi. The anatomy of web censorship in Pakistan. In *3rd USENIX Workshop on Free and Open Communications on the Internet (FOCI)*, 2013.

[30] Milad Nasr and Amir Houmansadr. Game of decoys: Optimal decoy routing through game theory. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1727–1738. ACM, 2016.

[31] Milad Nasr, Hadi Zolfaghari, and Amir Houmansadr. The waterfall of liberty: Decoy routing circumvention that resists routing attacks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 2037–2052, 2017.

[32] Max Schuchard, John Geddes, Christopher Thompson, and Nicholas Hopper. Routing around decoys. In *2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 85–96, 2012.

[33] Y. Schwartz, Y. Shavitt, and U. Weinsberg. On the diversity, stability and symmetry of end-to-end Internet routes. In *2010 INFOCOM IEEE Conference on Computer Communications Workshops*, pages 1–6, March 2010.

[34] M. C. Tschantz, S. Afroz, Anonymous, and V. Paxson. SoK: Towards grounding censorship circumvention in empiricism. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 914–933, May 2016.

[35] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. Effective attacks and provable defenses for website fingerprinting. In *Proceedings of the 23rd USENIX Conference on Security Symposium*, SEC'14, pages 143–157, 2014.

[36] 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 *2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 109–120, 2012.

[37] Philipp Winter and Stefan Lindskog. How the Great Firewall of China is blocking Tor. In *2nd USENIX Workshop on Free and Open Communications on the Internet (FOCI)*, 2012.

[38] Philipp Winter, Tobias Pulls, and Juergen Fuss. ScrambleSuit: A polymorphic network protocol to circumvent censorship. In *12th ACM Workshop on Workshop on Privacy in the Electronic Society*, WPES '13, pages 213–224, 2013.

[39] Eric Wustrow, Colleen M. Swanson, and J. Alex Halderman. TapDance: End-to-middle anticensorship without flow blocking. In *23rd USENIX Security Symposium*, pages 159–174, 2014.

[40] Eric Wustrow, Scott Wolchok, Ian Goldberg, and J. Alex Halderman. Telex: Anticensorship in the network infrastructure. In *20th USENIX Security Symposium*, 2011.

A Modifications to TLS

Most decoy routing systems require modifications to the TLSv1.2 handshake. In this section, we describe the modifications used by Slitheen [4] described in Section 3, and directly compare them to the original TLSv1.2 handshake. These are very similar to the modifications used in Telex [40] and Curveball [24].

Figure 8 gives an overview of a TLS handshake, with modifications shown in red. The modifications do not change the number or the size of the messages sent between the client and the server, only the contents of the messages. This is done in a way to avoid detection by the censor: only a party in possession of the client secret, the relay secret, or the TLS master secret can detect that modifications have been made.

The first modification happens in the generation of the ClientHello random nonce. This nonce is usually randomly generated and is used in the computation of the TLS master secret. In Slitheen, the last 28 bytes of this 32 byte nonce are replaced with a steganographic tag, $g^s \| H_1(g^{rs} \| \chi)$, where s is a secret generated by the client, g^r is the public key of a relay station, and χ = `server_ip||ClientHello random[0..3]` is a context string that consists of the server's IP address and the first 4 bytes of the ClientHello random nonce (in older versions of popular TLS libraries, this was often a timestamp; however, in newer versions it is typically generated randomly). The tag is recognizable only to the relay station with the private key r, and appears indistinguishable from random to any other observer.

The next modification is in the computation of values in the ClientKeyExchange message. Instead of randomly generating her private key exchange parameter, the client generates it from the previously generated tag. The private key is the result of feeding the client-relay shared secret $H_2(g^{rs} \| \chi)$ into a pseudo-random number generator. She then computes her public parameters in the ClientKeyExchange message from this private key. The relay station also has the ability to compute the private key, allowing it to later man-in-the-middle the TLS connection.

The last modification to the handshake is in the downstream Finished message, sent from the server to

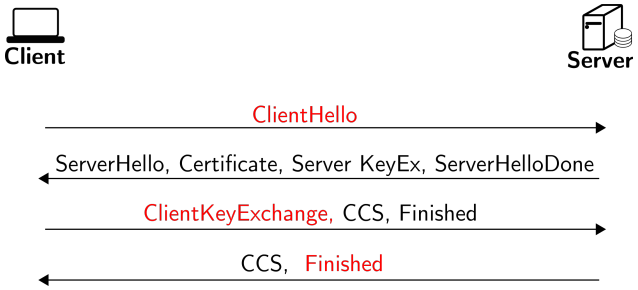


Fig. 8. An overview of the TLS handshake. Modified messages are shown in red.

the client. This message usually contains a hash of all previously seen handshake messages, `finished_hash`. The relay station intercepts this message and replaces it with a MAC that depends on the original Finished message and the client-relay shared secret, $MAC_{H_4(g^{rs}\|\chi)}(\texttt{finished\_hash})$. The purpose of this modification is to alert the client that the session can safely be used for decoy routing.

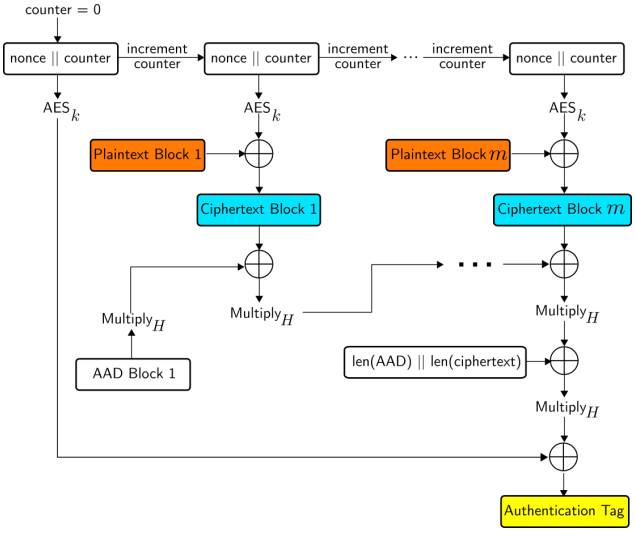


Fig. 9. AES in Galois Counter Mode (GCM) [26]. The counter is initialized to zero and incremented for each encryption step, in which the nonce is concatenated to the counter and encrypted with the key k. This encrypted block is XOR'd with a plaintext block (shown in orange) to produce the corresponding ciphertext block (shown in blue). Additional Authentication Data (AAD) and the ciphertext blocks are multiplied by the hash key $H = AES_k(0)$ and XOR'd together to produce the authentication tag (shown in yellow).

B Superencryption of covert data

Rebound [11], Slitheen [4], and Waterfall [31] require the modification and re-encryption of TLS application data that passes through the relay station. In Rebound and Slitheen, this data must be re-encrypted with the same TLS master secret. Some TLS modes of operation, such as AES-GCM (see Figure 9), rely on a public *nonce* in addition to the secret key. It is important to the security of AES-GCM that two different messages are never encrypted with the same nonce and the same key. However, many implementations of AES-GCM mode for TLS use sequential nonces for each message, and so when Slitheen or Rebound replaces message contents with covert data, it must reuse the nonce to avoid flagging to the observing censor that the censorship resistance system is in use. However, this presents a security problem as a third party capable of observing the ciphertext on both sides of the relay station can exploit patterns in the underlying plaintext to decrypt both ciphertext messages and modify the underlying plaintext without detection.

The original plaintext, P_1, will contain part of the HTTP response body of the original overt image, while the modified plaintext, P_2, will contain covert data to the user. The corresponding ciphertexts (limited to 1 block each for simplicity), seen by an observer, are computed as:

$$C_1 = E_k(n||0^{31}1) \oplus P_1$$

$$C_2 = E_k(n||0^{31}1) \oplus P_2$$

where E is AES encryption, and n is the nonce. The observer can then compute $C_1 \oplus C_2 = P_1 \oplus P_2$, and then exploit patterns in the underlying plaintexts to recover both P_1 and P_2. If the client was using Slitheen to browse a plaintext covert site, this two-time pad attack is trivial. In addition to breaking the client's confidentiality, an attacker can also modify the plaintext and compute the correct authentication tag [23]. Given the ciphertexts C_1 and C_2, as shown above, and the corresponding authentication tags (where A is one block of AAD for simplicity):

$$T_1 = ((A \cdot E_k(0) \oplus C_1) \cdot E_k(0) \oplus L) \cdot E_k(0) \oplus E_k(n||0^{32})$$

$$T_2 = ((A \cdot E_k(0) \oplus C_2) \cdot E_k(0) \oplus L) \cdot E_k(0) \oplus E_k(n||0^{32})$$

where $L = \texttt{len}(A)\|\texttt{len}(C)$ and multiplications are performed in $GF(2^{128})$.

The adversary can compute:

$$E_k(0) = \sqrt{\frac{T_1 \oplus T_2}{C_1 \oplus C_2}}$$

and from that, since the additional authentication data A is known:

$$E_k(n||0^{32}) = (((A \cdot E_k(0) \oplus C_1) \cdot E_k(0) \oplus L) \cdot E_k(0)) \oplus T_1.$$

This gives them everything necessary to compute their own tag for an arbitrary ciphertext C_3:

$$T_3 = ((A \cdot E_k(0) \oplus C_3) \cdot E_k(0) \oplus L) \cdot E_k(0) \oplus E_k(n||0^{32})$$

In the event that a user is browsing a covert site with TLS, the consequences of both of these attacks are mitigated. An adversary would be unable to decrypt the client's data, and any tampering would be detected in the TLS records sent between the client and the covert site. However, an adversary could use this to perform a targeted denial of service attack against decoy routing users. This attack is exceptionally damaging when the user of Slitheen is browsing a plaintext site, giving a third-party observer the ability to determine not only that the client is using Slitheen, but also *what covert data they are receiving.*

In Rebound, the same attack allows an adversary to determine that a change in the underlying plaintext has been made (and at what points the change has been made), but the randomness of the encrypted requests and responses between the client and relay station prevents an attacker from modifying the contents of the plaintext or discovering the nature of covert content.

To defend against this attack, we propose adding an additional layer of authenticated encryption under the ciphertext on the downstream side of the relay station, both in order to make it completely random, and to protect against modification. We chose this method rather than simply re-encrypting with a different, randomly generated, nonce as a censor could detect the usage of a non-sequential nonce. The keys for this "superencryption" step may be derived from the client ID in Slitheen, or the client-relay shared secret in Rebound by feeding it into a PRF.

Both the client and the relay station generate two superencryption keys: one to encrypt a 128-bit header and another to encrypt a variable-length covert data body. The header consists of an 4-byte counter and a 4-byte acknowledgement field. The counter is incremented for each chunk of covert data, and doubles as a mechanism for ordering covert data. In both Slitheen and Rebound, data to and from the covert site has the potential to arrive at the relay station and client, respectively, in a different order than it was written, as it can be distributed across multiple TCP connections to different overt sites. A counter allows each party to process the covert data in order, acknowledge the receipt of data, and retransmit data that was lost. The client and the server acknowledge received covert data in a similar style to TCP to prevent the loss of covert data chunks due to route-flapping or RAD attacks from a routing-capable adversary. The remainder of the header contains a 2-byte stream ID that indicates which connection to a covert server the data belongs to, the 2-byte length of the covert data chunk, and the 2-byte length of randomly generated padding. The last remaining 2 bytes of the header are padded with zeros. The covert data itself is encrypted using an authenticated encryption mode such as AES-GCM that is indistinguishable from random by an attacker.

Upon the receipt of a new chunk of covert content, the client or relay station will first decrypt the covert data header and extract the length of the covert data chunk. The client should verify that the counter is as expected and the padding at the end of the header exists. The client can then decrypt the covert data and send it to the client's browser.