Telex: Anticensorship in the Network Infrastructure

Eric Wustrow* Scott Wolchok* Ian Goldberg† J. Alex Halderman*

*The University of Michigan
{ewust, swolchok, jhalderm}@eecs.umich.edu

†University of Waterloo
iang@cs.uwaterloo.ca

Abstract

In this paper, we present Telex, a new approach to resisting state-level Internet censorship. Rather than attempting to win the cat-and-mouse game of finding open proxies, we leverage censors' unwillingness to completely block day-to-day Internet access. In effect, Telex converts innocuous, unblocked websites into proxies, *without* their explicit collaboration. We envision that friendly ISPs would deploy Telex stations on paths between censors' networks and popular, uncensored Internet destinations. Telex stations would monitor seemingly innocuous flows for a special "tag" and transparently divert them to a forbidden website or service instead. We propose a new cryptographic scheme based on elliptic curves for tagging TLS handshakes such that the tag is visible to a Telex station but not to a censor. In addition, we use our tagging scheme to build a protocol that allows clients to connect to Telex stations while resisting both passive and active attacks. We also present a proof-of-concept implementation that demonstrates the feasibility of our system.

1 Introduction

The events of the Arab Spring have vividly demonstrated the Internet's power to catalyze social change through the free exchange of ideas, news, and other information. The Internet poses such an existential threat to repressive regimes that some have completely disconnected from the global network during periods of intense political unrest, and many regimes are pursuing aggressive programs of Internet censorship using increasingly sophisticated techniques.

Today, the most widely-used tools for circumventing Internet censorship take the form of encrypted tunnels and proxies, such as Dynaweb [12], Instasurf [30], and Tor [10]. While these designs can be quite effective at sneaking client connections past the censor, these systems inevitably lead to a cat-and-mouse game in which the censor attempts to discover and block the services' IP addresses. For example, Tor has recently observed the blocking of entry nodes and directory servers in China and Iran [28]. Though Tor is used to skirt Internet censors in these countries, it was not originally designed for that application. While it may certainly achieve its original goal of anonymity for its users, it appears that Tor and proxies like it are ultimately not enough to circumvent aggressive censorship.

To overcome this problem, we propose*Telex*: an "end-to-middle" proxy with no IP address, located within the network infrastructure. Clients invoke the proxy by using public-key steganography to "tag" otherwise ordinary TLS sessions destined for uncensored websites. Its design is unique in several respects:

Architecture Previous designs have assumed that anticensorship services would be provided by hosts at the edge of the network, as the end-to-end principle requires. We propose instead to provide these services in the core infrastructure of the Internet, along paths between the censor's network and popular, nonblocked destinations. We argue that this will provide both lower latency and increased resistance to blocking.

Deployment Many systems attempt to combat state-level censorship using resources provided primarily by volunteers. Instead, we investigate a government-scale response based on the view that state-level censorship needs to be combated by state-level anticensorship.

Construction We show how a technique that the security and privacy literature most frequently associates with government surveillance—deep-packet inspection—can provide the foundation for a robust anticensorship system.

We expect that these design choices will be somewhat controversial, and we hope that they will lead to discussion about the future development of anticensorship systems.

Contributions and roadmap We propose using "end-to-middle" proxies built into the Internet's network infrastructure as a novel approach to resisting state-level censorship. We elaborate on this concept and sketch the design of our system in Section 2, and we discuss its relation to previous work in Section 3.

We develop a new steganographic tagging scheme based on elliptic curve cryptography, and we use it to construct a modified version of the TLS protocol that allows clients to connect to our proxy. We describe the tagging scheme in Section 4 and the protocol in Section 5. We analyze the protocol's security in Section 6.

We present a proof-of-concept implementation of our approach and protocols, and we support its feasibility through laboratory experiments and real-world tests. We describe our implementation in Section 7, and we evaluate its performance in Section 8.

Online resources For the most recent version of this paper, prototype source code, and a live demonstration, visit https://telex.cc.

2 Concept

Telex operates as what we term an "end-to-middle" proxy. Whereas in traditional end-to-end proxying the client connects to a server that relays data to a specified host, in end-to-middle proxying an intermediary along the path to a server redirects part of the connection payload to an alternative destination. One example of this mode of operation is Tor's leaky-pipe circuit topology [10] feature, which allows traffic to exit from the middle of a constructed Tor circuit rather than the end.

The Telex concept is to build end-to-middle proxying capabilities into the Internet's routing infrastructure. This would let clients invoke proxying by establishing connections to normal, pre-existing servers. By applying this idea to a widely used encrypted transport, such as TLS, and carefully avoiding observable deviations from the behavior of nonproxied connections, we can construct a service that allows users to robustly bypass network-level censorship without being detected.

In the remainder of this section, we define a threat model and goals for the Telex system. We then give a sketch of the design and discuss several practical considerations.

2.1 Threat model

Our adversary, "the censor", is a repressive state-level authority that desires to inhibit online access to information and communication of certain ideas. These desires are realized by IP and DNS blacklists as well as heuristics for blocking connections based on their observed content.

We note that the censor has some motivation for connecting to the Internet at all, such as the economic and social benefits of connectivity. Thus, the censor bears some cost from over-blocking. We assume that the censor follows a blacklist approach rather than a whitelist approach in blocking, allowing traffic to pass through unchanged unless it is explicitly banned.

Furthermore, we assume that the censor generally permits widespread cryptographic protocols, such as TLS, except when it has reason to believe a particular connection is being used for skirting censorship. We further assume that the censor is not subverting such protocols on a wide scale, such as by requiring a cryptographic backdoor or by issuing false TLS certificates using a country-wide CA. We believe this is reasonable, as blocking or subverting TLS on a wide scale would render most modern websites unusably insecure. Subversion in particular would result in an increased risk of large-scale fraud if the back door were compromised or abused by corrupt insiders.

The censor controls the infrastructure of the network within its jurisdiction ("the censor's network"), and it can potentially monitor, block, alter, and inject traffic anywhere within this region. However, these abilities are subject to realistic technical, economic, and political constraints.

In general, the censor does *not* control end hosts within its network, which operate under the direction of their users. We believe this assumption is reasonable based on the failure of recent attempts by national governments to mandate client-side filtering software, such as China's Green Dam Youth Escort [33]. The censor might target a small subset of users and seize control of their devices, either through overt compulsion or covert technical attacks. Protecting these users is beyond the scope of our system. However, the censor's targeting users on a wide scale might have unacceptable political costs.

The censor has very limited abilities outside its network. It does not control any external network infrastructure or any popular external websites the client may use when communicating with Telex stations. The censor can, of course, buy or rent hosting outside its network, but its use is largely subject to the policies of the provider and jurisdiction.

Some governments may choose to deny their citizens Internet connectivity altogether, or disconnect entirely in times of crisis. These are outside our threat model; the best approaches to censors like these likely involve different approaches than ours, and entail much steeper performance trade-offs. Instead, our goal is to make access to any part of the global Internet sufficient to access every part of it. In other words, we aim to make connecting to the global Internet an all-or-nothing proposition for national governments.

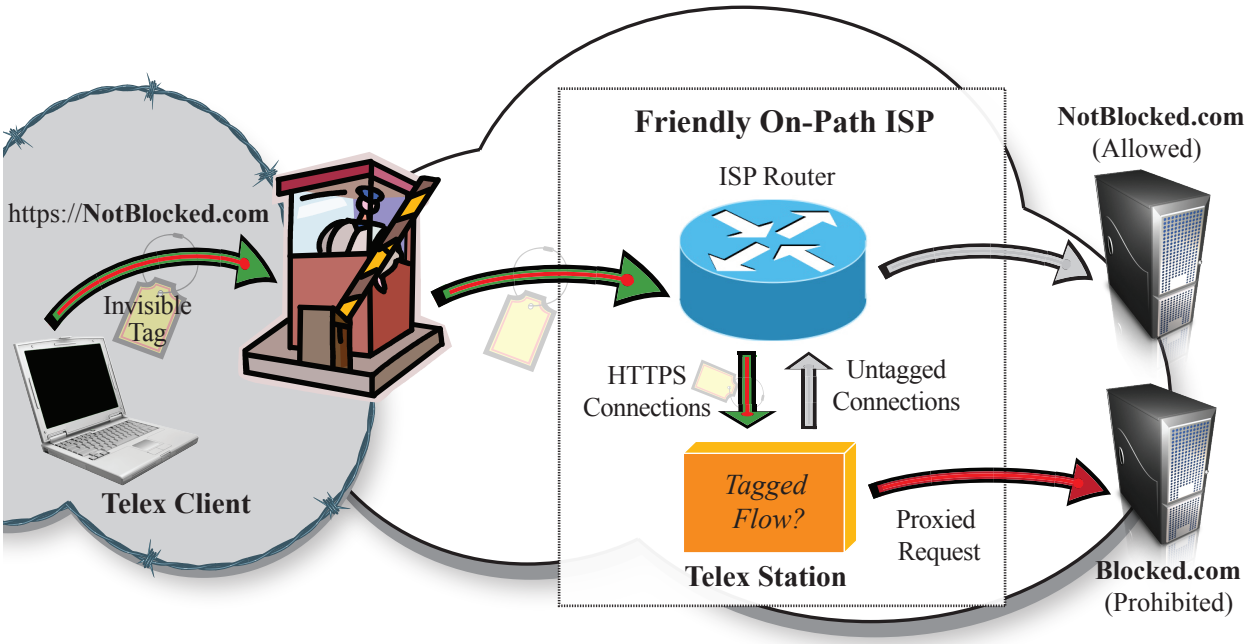

Figure 1: **Telex Concept** — This figure shows an example user connecting to Telex. The client makes a tagged connection to NotBlocked.com, which is passed by the censor's filter. When the request reaches a friendly on-path ISP, one of the ISP's routers forwards the request to the Telex station connected to its tap interface. Telex deciphers the tag, instructs the router to block the connection to NotBlocked.com. and diverts the connection to Blocked.com, as the user secretly requested. If the connection were not tagged, Telex would not intervene, and it would proceed to NotBlocked.com as normal.

2.2 Goals

Telex should satisfy the following properties:

Unblockable The censor should not be able to deny service to Telex without incurring unacceptable costs. In particular, we require that the censor cannot block Telex without blocking a large, primarily legitimate category of Internet traffic.

Confidential The censor should not be able to determine whether a user is using Telex or what content the user is accessing through the system.

Easy to deploy The consequences of system failure (or even normal operation) must not interfere with normal network operation (e.g., non-Telex connections) in order for deployment to be palatable to ISPs.

Transparent to users Using Telex should, possibly after a small startup procedure, closely resemble using an unfiltered Internet connection.

2.3 Design

To meet our goals and the constraints imposed by our threat model, we propose the design shown in Figure 1. As illustrated in the figure, a Telex connection proceeds as follows:

1. The user's client selects an appropriate website that is not on the censor's blacklist and unlikely to attract attention, which we represent by the domain NotBlocked.com.
2. The user connects to NotBlocked.com via HTTPS. Her Telex client[1] includes an invisible "tag," which looks like an expected random nonce to the censor, but can be cryptographically verified by the Telex station using its private key.
3. Somewhere along the route between the client and NotBlocked.com, the connection traverses an ISP that has agreed to attach a Telex station to one of its routers. The connection is forwarded to the station via a dedicated tap interface.
4. The station detects the tag and instructs the router to block the connection from passing through it, while still forwarding packets to the station through its dedicated tap. (Unlike a deployment based on transparent proxying, this configuration *fails open*: it tolerates the failure of the entire Telex system and so meets our goal of being easy to deploy.)
5. The Telex station diverts the flow to Blocked.com as

[1] We anticipate that client software will be distributed out of band, perhaps by sneakernet, among mutually trusting individuals within the censor's domain.

the user requested; it continues to actively forward packets from the client to Blocked.com and vice versa until one side terminates the connection. If the connection were untagged, it would pass through the ISP's router as normal.

We simplified the discussion above in an important point: we need to specify what protocol is to be used over the encrypted tunnel between the Telex client and the Telex station and how the client communicates its choice of Blocked.com. Layering IP atop the tunnel might seem to be a natural choice, yielding a country-wide VPN of sorts, but even a passive attacker may be able to differentiate VPN traffic patterns from those of a normal HTTPS connection. As a result, we primarily envision using Telex for protocols whose session behavior resembles that of HTTPS. For example, an HTTP or SOCKS proxy would be a useful application, or perhaps even a simple server that presented a list of entry points for another anticensorship system such as Tor [10]. In the remainder of this paper, we assume that the application is an HTTP proxy.

The precise placement of Telex stations is a second issue. Clearly, a chief objective of deployment is to cover as many paths between the censor and popular Internet destinations as possible so as to provide a large selection of sites to play the role of NotBlocked.com. We might accomplish this either by surrounding the censor with Telex stations or by placing them close to clusters of popular uncensored destinations. In the latter case, care should be taken not to reduce the size of the cluster such that the censor would only need to block a small number of otherwise desirable sites to render the station useless. Which precise method of deployment would be most effective and efficient is, in part, a geopolitical question.

A problem faced by existing anticensorship systems is providing sufficient incentives for deployment [6]. Whereas systems that require cooperation of uncensored websites create a risk that such sites might be blocked by censors in retaliation, our system requires no such participation. We envision that ISPs will willingly deploy Telex stations for a number of reasons, including idealism, goodwill, public relations, or financial incentives (e.g., tax credits) provided by governments. At worst, the consequences to ISPs for participation would be depeering, but depeering a large ISP would have a greater impact on overall network performance than blocking a single website.

Discovery of Telex stations is a third issue. With wide enough deployment, clients could pick HTTPS servers at random. However, this behavior might divulge clients' usage of Telex, because real users don't actually visit HTTPS sites randomly. A better approach would be to opportunistically discover Telex stations by tagging flows during the course of the user's normal browsing. When a station is eventually discovered, it could provide a more comprehensive map of popular sites (where popularity is as measured with data from other Telex users) such that a Telex station is likely to be on the path between the user and the site. Even with only partial deployment, users would almost certainly discover a Telex station eventually.

3 Previous Work

There is a rich literature on anonymous and censorship-resistant communication, going back three decades [7]. One of the first systems explicitly proposed for combating wide-scale censorship was Infranet [13], where participating websites would discreetly provide censored content in response to steganographic requests. Infranet's designers dismissed the use of TLS because, at the time, it was not widely deployed and would be easily blocked. We observe that this aspect of Internet use has substantially changed since 2002. Unlike Infranet, Telex does not require the cooperation of unblocked websites—a significant impediment to deployment—which participate in our system only as oblivious cover destinations.

A variety of systems provide low-latency censorship resistance through VPNs or encrypted tunnels to proxies. These systems rely on servers at the edge of the network, which censors constantly try to find and block (via IP). By far, the best studied of these systems is Tor [10], which also attempts to make strong anonymity guarantees by establishing a multi-hop encrypted tunnel. Traditionally, users connect to Tor via a limited set of "entry nodes," which provide an obvious target for censors. In response, Tor has implemented bridges [27], which are a variation on Feamster et al.'s keyspace hopping [14], in which each client is told only a small subset of addresses of available proxies. While bridges provide an extra layer of protection, the arms race remains: Chinese censors now learn and block a large fraction of bridge nodes [9], possibly by using a Sybil attack [11] against the bridge address distribution system. Like Telex, Tor adopts a pragmatic threat model that emphasizes performance; it wraps connections using TLS and does not strongly protect against traffic analysis and end-to-end timing attacks [22]. Unlike Tor, we separate the problem of censorship resistance from that of anonymous communication and concentrate on resisting blocking; users who require increased anonymity can use Telex as a gateway to the Tor network.

The most widely-used anticensorship tools today are also among those that make the fewest security promises. Pragmatic systems such as Dynaweb [12] and Ultrasurf [30] that employ simple encrypted tunnels with large numbers of entry points are popular, and, so far, have managed to stay one step ahead of many censors. However, we worry that such systems will not be able to withstand continued research and development on the part of censors (e.g., Sybil attacks for proxy IP discovery). We aim

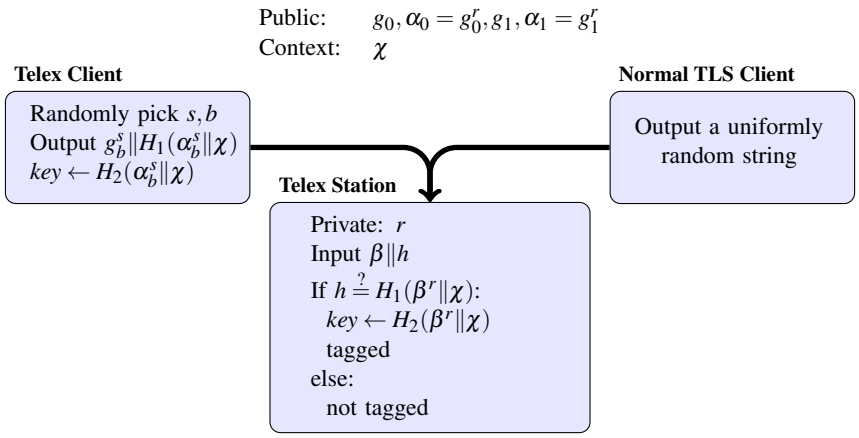

Figure 2: **Tag creation and detection** — Telex intercepts TLS connections that contain a steganographic tag in the ClientHello message's nonce field (normally a uniformly random string). The Telex client generates the tag using public parameters (shown above), but it can only be recognized by using the private key r embedded in the Telex station.

to provide similar or better performance by adopting a single-hop tunnel and locating proxies in the middle of the network, where they are not susceptible to IP-based blocking.

4 Tagging

In this section, we describe how we implement the invisible tag for TLS connections, which only Telex stations can recognize. We present an overview here, while the details and a security argument appear in Appendix A. Figure 2 depicts the tagging scheme.

Our tags must have two properties: they must be *short*, and they must be *indistinguishable* from a uniformly random string to anyone without the private key. Someone *with* the private key should be able to examine a random-looking value and efficiently decide whether the tag is present; if so, a shared secret key is derived for use later in the protocol.

The structure of the Telex tagging system is based on Diffie-Hellman: there is a generator g of a group of prime order. Telex has a private key r and publishes a public key $\alpha = g^r$. The system uses two cryptographically secure hash functions H_1 and H_2, each salted by the current *context string* χ (see Section 5). To construct a tag, the client picks a random private key s, and computes g^s and $\alpha^s = g^{rs}$. If $\|$ denotes concatenation, the tag is then $g^s \| H_1(g^{rs} \| \chi)$, and the derived shared secret key is $H_2(g^{rs} \| \chi)$.

Diffie-Hellman can be implemented in many different groups, but in order to keep the tags both short and secure, we must use elliptic curve groups. Then we must ensure that, in whatever bit representation we use to transmit group elements g^s, they are indistinguishable from uniformly random strings of the same size. This turns out to be quite tricky, for three reasons:

- First, it is easy to tell whether a given (x, y) is a point on a (public) elliptic curve. Most random strings will not appear to be such a point. To work around this, we only transmit the x-coordinates of the elliptic curve points.
- Second, it is the case that these x-coordinates are taken modulo a prime p. Valid tags will never contain an x-coordinate larger than p, so we must ensure that random strings of the same length as p are extremely unlikely to represent a value larger than p. To accomplish this, we select a value of p that is only slightly less than a power of 2.
- Finally, it turns out that for any given elliptic curve, only about half of the numbers mod p are x-coordinates of points on the curve. This is undesirable, as no purported tag with an x-coordinate not corresponding to a curve point can possibly be valid. (Conversely, if a given client is observed using only x-coordinates corresponding to curve points, it is very likely using Telex.) To solve this, we use *two* elliptic curves: the original curve and a related one called the "twist". These curves have the property that every number mod p is the x-coordinate of a point on either the original curve or the twist. We will now need two generators: g_0 for the original curve, and g_1 for the twist, along with the corresponding public keys $\alpha_0 = g_0^r$ and $\alpha_1 = g_1^r$. Clients pick one pair (g_b, α_b) uniformly at random when constructing tags.

When Telex receives a candidate tag, it divides it into two parts as $\beta \| h$, according to the fixed lengths of group elements and hashes. It also determines the current con-

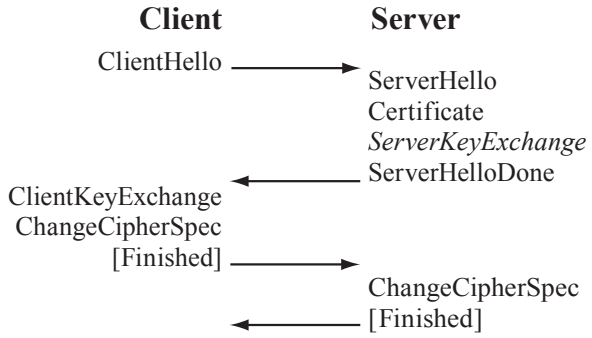

Figure 3: **TLS Handshake** — The client and server exchange messages to establish a shared master_secret, from which they derive cipher and MAC keys. The handshake ends with each side sending a Finished message, encrypted with the negotiated keys, that includes an integrity check on the entire handshake. The ServerKeyExchange message may be omitted, depending on the key exchange method in use.

text string χ. If this is a valid tag, β will be g_b^s and h will be $H_1(g_b^{rs}\|\chi)$ for some s and b. If this is not a valid tag, β and h will both be random. Thus, Telex simply checks whether $h \stackrel{?}{=} H_1(\beta^r\|\chi)$. This will always be true for valid tags, and will be true only with probability $2^{-\ell_{H_1}}$ for invalid tags, where ℓ_{H_1} is the bit length of the outputs of H_1. If it is true, Telex computes the shared secret key as $H_2(\beta^r\|\chi)$.

5 Protocol

In this section, we briefly describe the Transport Layer Security (TLS) protocol [8] and then we explain our modifications to it.

5.1 Overview of TLS

TLS provides a secure channel between a client and a server, and consists of two sub-protocols: the handshake protocol and the record protocol. The handshake protocol provides a mechanism for establishing a secure channel and its parameters, including shared secret generation and authentication. The record protocol provides a secure channel based on parameters established from the handshake protocol.

During the TLS handshake, the client and server agree on a cipher suite they will use to communicate, the server authenticates itself to the client using asymmetric certificates (such as RSA), and cryptographic parameters are shared between the server and client by means of a key exchange algorithm. While TLS supports several key exchange algorithms, in this paper, we will focus on the Diffie-Hellman key exchange.

Figure 3 provides an outline of the TLS handshake. We describe each of these messages in detail below:

ClientHello contains a 32-byte nonce, a session identifier (0 if a session is not being resumed), and a list of supported cipher suites. The nonce consists of a 4-byte Unix timestamp, followed by a 28-byte random value.

ServerHello contains a 32-byte nonce formed identically to that in the ClientHello as well as the server's choice of one of the client's listed cipher suites.

Certificate contains the X.509 certificate chain of the server, and authenticates the server to the client.

ServerKeyExchange provides the parameters for the Diffie-Hellman key exchange. These parameters include a generator g, a large prime modulus p_{DH}, a server public key, and a signature. As per the Diffie-Hellman key exchange, the server public key is generated by computing $g^{s_{priv}} \bmod p_{DH}$, where s_{priv} is a large random number generated by the server. The signature consists of the RSA signature (using the server's certificate private key) over the MD5 and SHA-1 hashes of the client and server nonces, and previous Diffie-Hellman parameters.

ServerHelloDone is an empty record, used to update the TLS state on the receiving (i.e., client) end.

ClientKeyExchange contains the client's Diffie-Hellman parameter (the client public key generated by $g^{c_{priv}} \bmod p_{DH}$).

ChangeCipherSpec alerts the server that the client's records will now be encrypted using the agreed upon shared secret. The client finishes its half of the handshake protocol with an encrypted Finished message, which verifies the cipher spec change worked by encrypting a hash of all previous handshake messages.

5.2 Telex handshake

The Telex handshake has two main goals: first, the censor should not be able to distinguish it from a normal TLS handshake; second, it should position the Telex station as a man-in-the-middle on the secure channel. We now describe how the Telex handshake deviates from a standard TLS handshake.

Client setup The client selects an uncensored HTTPS server located outside the censor's network (canonically, https://NotBlocked.com) and resolves its hostname to find `server_ip`. This server may be completely oblivious to the anticensorship system. The client refers to its database of Telex stations' public keys to select the appropriate key $P = (\alpha_0, \alpha_1)$ for this session. We leave the details of selecting the server and public key for future work.

ClientHello message The client generates a fresh tag τ by applying the algorithm specified in Section 4, using public key P and a context string composed of `server_ip`$\|$`UNIX_timestamp`$\|$`TLS_session_id`.

This yields a 224-bit tag τ and a 128-bit shared secret key k_{sh}. The client initiates a TCP connection to `server_ip` and starts the TLS handshake. As in normal TLS, the client sends a ClientHello message, but, in place of the 224-bit random value, it sends τ.

(Briefly, the tag construction ensures that the Telex station can use its private key to efficiently recognize τ as a valid tag and derive the shared secret key k_{sh}, and that, without the private key, the distribution of τ values is indistinguishable from uniform; see Section 4.)

If the path from the client to `server_ip` passes through a link that a Telex station is monitoring, the station observes the TCP handshake and ClientHello message. It extracts the nonce and applies the tag detection algorithm specified in Section 4 using the same context string and its private key. If the nonce is a genuine tag created with the correct key and context string, the Telex station learns k_{sh} and continues to monitor the handshake. Otherwise, with overwhelming probability, it rejects the tag and stops observing the connection.

Certificate validation The server responds by sending its X.509 certificate and, if necessary, key exchange values. The client verifies the certificate using the CA certificates trusted by the user's browser. It additionally checks the CA at the root of the certificate chain against a whitelist of CAs trusted by the anticensorship service. If the certificate is invalid or the root CA is not on the whitelist, the client proceeds with the handshake but aborts its Telex invocation by strictly following the TLS specification and sending an innocuous application-layer request (e.g., `GET / HTTP/1.1` for HTTPS).[2]

Key exchange At this point in the handshake, the client participates in the key exchange to compute a master secret shared with the server. We modify the key exchange in order to "leak" the negotiated key to the Telex station. Several key exchange algorithms are available. For example, in RSA key exchange, the client generates a random 46-byte master key and encrypts it using the server's public key. Alternatively, the client and server can participate in a Diffie-Hellman key exchange to derive the master secret.

The Telex client, rather than generating its key exchange values at random, seeds a secure PRG with k_{sh} and uses its output for whatever randomness is required in the key exchange algorithm (e.g., the Diffie-Hellman exponent). If a Telex station has been monitoring the connection to this point, it will know all the inputs to the client's key exchange procedure: it will have observed the server's key exchange parameter and computed the client's PRG seed k_{sh}. Using this information, the Telex station simulates the client and simultaneously derives the same master secret.

Handshake completion If a Telex station is listening, it attempts to decrypt each side's Finished message. The station should be able to use the master secret to decrypt them correctly and verify that the hashes match its observations of the handshake. If either hash is incorrect, the Telex station stops observing the connection. Otherwise, it switches roles from a passive observer to a man-in-the-middle. It forges a TCP RST packet from the client to NotBlocked.com, blocks subsequent messages from either side from reaching the remote end of the connection, and assumes the server's role in the unbroken TCP/TLS connection with the client.

Session resumption Once a client and server have established a session, TLS allows them to quickly resume or duplicate the connection using an abbreviated handshake. Our protocol can support this too, allowing the Telex station to continue its role as a man-in-the-middle.

The station remembers `key` and `session_id` by the server, for sessions it successfully joined. A client attempts to resume the session on a new connection by sending a ClientHello message containing the `session_id` and a fresh tag τ', which Telex can observe and verify if it is present. If the server agrees to resume the session, it responds with a ServerHello message and a Finished message encrypted with the original master secret. The client then sends its own Finished message encrypted in the same way, which confirms that it knows the original master secret. The Telex station checks that it can decrypt and verify these messages correctly, then switches into a man-in-the-middle role again.

6 Security Analysis

In this section, we analyze Telex's security under the threat model described in Section 2.1.

6.1 Passive attacks

First, we consider a passive censor who is able to observe arbitrary traffic within its network. For this censor to detect that a client is using Telex, it must be able to distinguish normal TLS flows from Telex flows.

Telex deviates from a normal TLS handshake in the client's nonce (sent in the ClientHello message) and in the client's key exchange parameters. In Section 4, we showed that an attacker cannot distinguish a Telex tag from a truly random string with more than a negligible advantage. This means that a client's tagged nonce (using Telex) is indistinguishable from a normal TLS random nonce. Likewise, the Telex-generated key exchange parameters are the output of a secure PRG; they are not

[2] Both the additional root CA whitelist and the browser list need to be checked; the censor may control a CA that is commonly whitelisted by browsers, and the root CA whitelist may contain entries that are trusted by one browser but not another.

distinguishable from truly random strings as a direct result of the security of the PRG.

During the TLS record protocol, symmetric cryptography is used between the Telex station and the client. A censor will be unable to determine the contents of this encrypted channel, as in normal TLS, and will thus be unable to distinguish between a Telex session and a normal TLS session from the cryptographic payload alone.

Stream cipher weakness TLS supports several stream cipher modes for encrypting data sent over the connection. Normally, the key stream is used once per session, to avoid vulnerability to a reused key attack. However, the Telex station and NotBlocked.com use the same shared secret when sending data to the client, so the same key stream is used to encrypt two different plaintexts. An attacker (possibly different from the censor) with the ability to receive both of the resulting ciphertexts can simply XOR them together to obtain the equivalent of the plaintexts XORed together. To mitigate this issue, Telex sends a TCP RST to NotBlocked.com to quickly stop it from returning data. In addition, our implementation uses a block cipher in CBC mode, for which TLS helps mitigate these issues further by providing for the communication of a random per-record IV.

We note that an adversary in position to carry out this attack (such as one surrounding the Telex station) already has the ability to detect the client's usage of Telex, as well as the contents of the connection from Telex to Blocked.com.

Traffic analysis A sophisticated adversary might attempt to detect a use of Telex by detecting anomalous patterns in connection count, packet size, and timing. Previous work shows how these characteristics can be used to fingerprint and identify specific websites being retrieved over TLS [18]. However, this kind of attack would be well beyond the level of sophistication observed in current censors [16]. We outline a possible defense against traffic analysis in Section 9.

6.2 Active attacks

Our threat model also allows the censor to attempt a variety of active attacks against Telex. The system provides strong defenses against the most practical of these attacks.

Traffic manipulation The censor might attempt to modify messages between the client and the Telex station, but Telex inherits defenses against this from TLS. For example, if the attacker modifies any of the parameters in the handshake messages, the client and Telex station will each detect this when they check the MACs in the Finished messages, which are protected by the shared secret of the TLS connection. Telex will then not intercept the connection, and the NotBlocked.com server will respond with a TLS error. Widescale manipulation of TLS handshakes or payloads would disrupt Telex; however, it would also interfere with the normal operation of TLS websites.

Tag replay The censor might attempt to use various replay attacks to detect Telex usage. The most basic of these attacks is for the censor to initiate its own Telex connection and reuse the nonce from a suspect connection; if this connection receives Telex service, the censor can conclude that the nonce was tagged and the original connection was a Telex request.

Our protocol prevents this by requiring the client to prove to the Telex station that it knows the shared secret associated with the tagged nonce. We achieve this by using the shared secret to derive the key exchange parameter, as described in Section 5. In particular, consider the encrypted Finished message that terminates the TLS handshake. This message must be encrypted using the freshly negotiated key (or else the TLS server will hang up), so it cannot simply be replayed. Second, the key exchange parameter in use must match the shared secret in the tagged nonce, or the Telex station will not be able to verify the MAC on the Finished message. Together, these requirements imply that the client must know the shared secret.

Handshake replay This property of proving knowledge of the shared secret is only valid if the server provides fresh key exchange parameters. An attacker may circumvent this protection by replaying traffic in *both* directions across the Telex station. This attack will cause a visible difference in the first ApplicationData message received at the client, provided that either 1) Blocked.com's response is not completely static (e.g., it sets a session cookie) or 2) the original connection being replayed was an *unsuccessful* Telex connection. In either case, the new ApplicationData message will be fresh data from Blocked.com.

A partial defense against this attack is to enforce freshness of the timestamps used in both halves of the TLS handshake and prohibit nonce reuse within the window of acceptable timestamps. However, this defense fails in the case where the original connection being replayed was an unsuccessful attempt to initiate a Telex connection, because the Telex station did not see the first use of the nonce. As a further defense, we note that NotBlocked.com will likely not accept replayed packets, and the Telex station can implement measures to detect attempts to prevent replayed packets from reaching NotBlocked.com.

Ciphertext comparison The attacker is able to detect the use of Telex if they are able to receive the unaltered traffic from NotBlocked.com, in addition to the traffic they forward to the client. Though they will not be able to decrypt either of the messages, they will be able to see

that the ciphertexts differ, and from this conclude that a client is using Telex. Normally, Telex blocks the traffic between NotBlocked.com and the client after the TLS handshake to prevent this type of attack.

However, it is possible for an attacker to use DNS hijacking for this purpose. The attacker hijacks the DNS entry for NotBlocked.com to point to an attacker-controlled host. The client's path to this host passes through Telex, and the attacker simply forwards traffic from this host to NotBlocked.com. Thus, the attacker is able to observe the ciphertext traffic on both sides of the Telex station, and therefore able to determine when it modifies the traffic.

Should censors actually implement this attack, we can modify Telex stations in the following way to help detect DNS hijacking until DNSSEC is widely adopted. When it observes a tagged connection to a particular server IP, the station performs a DNS lookup based on the common name observed in the X.509 certificate. This DNS lookup returns a list of IP addresses. If the server IP for the tagged connection appears in this list, the Telex station will respond to the client and proxy the connection. Otherwise, the station will not deviate from the TLS protocol, as it is possible that the censor is hijacking DNS. This may lead to false negatives, as DNS is not globally consistent for many sites, but as long as the censor has not compromised the DNS chain that the station uses, there will be no false positives. For popular sites, we could also add a whitelisted cache of IP addresses.

Since the censor controls part of the network between the client and the Telex station, it could also try to redirect the connection by other means, such as transparently proxying the connection to a censor-controlled host. In these cases, the destination IP address observed by Telex will be different from the one specified by the client. Thus, the context strings constructed by the client and Telex will differ, and Telex will not recognize the connection as tagged. This attack offers the adversary an expensive denial of service attack, but it does not allow the attacker to detect attempted use of Telex.

Denial of service A censor may attempt to deny service from Telex in two ways. First, it may attempt to exhaust Telex's bandwidth to proxy to Blocked.com. Second, it may attempt to exhaust a Telex station's tag detection capabilities by creating a large amount of ClientHello messages for the station to check. Both methods are overt attacks that may cause unwanted political backlash on the censor or even provoke an international incident. To combat the first attack, we can implement a client puzzle [20], where Telex issues a computationally intensive puzzle the client must solve before we allow proxy service. The client puzzle should be outsourced [32] to avoid additional latency that might distinguish Telex handshakes from normal TLS handshakes. To combat the second attack, we can implement our tag checking in hardware to increase throughput if necessary.

7 Implementation

To demonstrate the feasibility of Telex, we implemented a proof-of-concept client and station. While we believe these prototypes are useful models for research and experimentation, we emphasize that they may not provide the performance or security of a more polished production implementation, and should be used accordingly.

7.1 Client

Our prototype client program, which we refer to as `telex_client`, is designed to allow any program that uses TCP sockets to connect to the Telex service without modification. It is written in approximately 1200 lines of C (including 500 lines of shared TLS utility code) and uses libevent to manage multiple connections. The user initializes `telex_client` by specifying a local port and a remote TLS server that is not blocked by the censor (e.g. NotBlocked.com). Once `telex_client` launches, it begins by listening on the specified local TCP socket. Each time a program connects to this socket, `telex_client` initiates a TLS connection to the unblocked server specified previously. Following the Telex-TLS handshake protocol (see Section 5.2), `telex_client` inserts a tag, generated using the scheme described in Section 4, into the ClientHello nonce. We modified OpenSSL to accept supplied values for the nonce as well as the client's Diffie-Hellman exponent. We supply this 1024-bit value as the output of a secure pseudorandom generator with input k_{sh} associated with the previously generated tag. These changes required us to modify fewer than 20 lines of code in OpenSSL 1.0.0.

7.2 Station

Our prototype Telex station uses a modular design to provide a basis for scaling the system to high-speed links and to ensure reliability. In particular, it fails safely: simple failures of the components will not impact non-Telex TLS traffic. The implementation is divided into three components, which are responsible for diversion, recognition, and proxying of network flows.

Diversion The first component consists of a router at the ISP hosting the Telex station. It is configured to allow the Telex station to passively monitor TLS packets (e.g., TCP port 443) via a tap interface. Normally, the router will also forward the packets towards their destination, but the recognition and relay components can selectively command it to not forward traffic for particular flows. This allows the other components to selectively manipulate packets and then reinject them into the network. In

our implementation, the router is a Linux system that uses the iptables and ipset [19] utilities for flow blocking.

Recognition During the TLS handshake, the Telex station recognizes tagged connections by inspecting the ClientHello nonces. In our implementation, the recognition subsystem reconstructs the TCP connection using the Bro Network Intrusion Detection System [23]. Bro reconstructs the application-layer stream and provides an event-based framework for processing packets. We used the Bro scripting language for packet processing (approximately 300 lines), and we added new Bro built-in functions using C++ (approximately 450 lines).

When the Bro script recognizes a TLS ClientHello message, it checks the client nonce to see whether it is tagged. (The tag checking logic is a C implementation of the algorithm described in Section 4.) If the nonce is tagged, we extract the shared secret associated with the tag and create an entry for the connection in a table indexed by flow. All future event handlers test whether the flow triggering the event is contained in this table, and do nothing if it is not.

The Bro script then instructs the diversion component (via a persistent TCP connection) to block the associated flow. As this does not affect the tap, our script still receives the associated packets, and the script is responsible for actively forwarding them until the TLS Finished messages are observed. This allows the Bro script to inspect each packet before forwarding it, while ensuring that any delays in processing will not cause a packet that should be blocked to make it through the router (e.g., a TLS ApplicationData packet from NotBlocked.com to the client). To derive the TLS shared secret from the key exchange, our Bro script also stores the necessary parameters from the TLS ServerKeyExchange message in the connection table.

Once it observes the server's TLS Finished handshake message, our Bro script stops forwarding packets between the client and the server (thus atomically severing traffic flow between them) and sends the connection state, which includes the TCP-level state (sequence number, TCP options, windows, etc.), the key exchange parameters, and the shared secret k_{sh} to the proxy service component. Our proof-of-concept implementation handles only the TCP timestamp, selective acknowledgements (SACK), and window scaling options, but other options could be handled similarly. Likewise, we currently only support TLS's Diffie-Hellman key exchange, but RSA and other key exchange methods could also be supported.

Proxy service The proxy service component plays the role of the TLS server and connects the client to blocked websites. Our implementation consists of a user space process called `telex_relay` and an associated kernel module, which are responsible for decapsulating TLS connection data and passing it to a local Squid proxy [25].

The `telex_relay` process is responsible for relaying data from the client to the Squid proxy, in effect spoofing the server side of the connection. We defer forwarding of the last TLS Finished message until `telex_relay` has initialized its connection state in order to ensure that all application data is observed. We implement this delay by including the packet containing TLS Finished message in the state sent from our Bro script and leaving the task of forwarding the packet to its destination to `telex_relay`, thus avoiding further synchronization between the components.

Similarly to `telex_client`, `telex_relay` is written in about 1250 lines of C (again including shared TLS utility code) and uses libevent to manage multiple connections. It reuses our modifications to OpenSSL in order to substitute our shared secret for OpenSSL's shared secret. We implement relaying of packets between the client and the Telex service straightforwardly, by registering event handlers to read from one party and write to the other using the usual `send` and `recv` system calls on the one hand and `SSL_read` and `SSL_write` on the other.

To avoid easy detection, the relay's TCP implementation must appear similar to that of the original TLS server. Ideally, `telex_relay` would simply `bind(2)` to the address of the original server and set the `IP_TRANSPARENT` socket option, which, in conjunction with appropriate firewall and routing rules for transparent proxying [29], would cause its socket to function normally despite being bound to a non-local address. This would cause the relay's TCP implementation to be identical to that of the operating system that hosts it. However, the TCP handshake has already happened by the time our Bro script redirects the connection to `telex_relay`, so we need a method of communicating the state negotiated during the handshake to the TCP implementation. Accordingly, we modified the Linux 2.6.37 kernel to add a `fake_accept` ioctl that allows a userspace application to create a seemingly connected socket with arbitrary TCP state, including endpoint addresses, ports, sequence numbers, timestamps, and windows.

8 Evaluation

In this section, we evaluate the feasibility of our Telex proxy prototype based on measurements of its performance.

8.1 Model deployment

We used a small model deployment consisting of three machines connected in a hub-and-spoke topology. Our simulated router is the hub of our deployment, and the two machines connected are the Telex station, and a web server serving pages over HTTPS and HTTP. The Telex

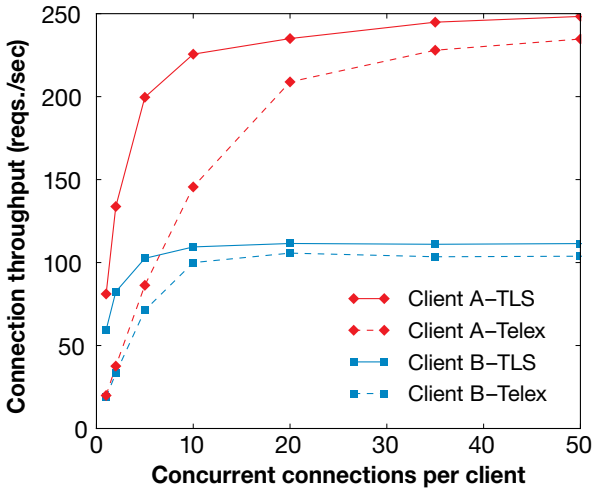

Figure 4: **Client Request Throughput** — We measured the rate at which two client machines could complete HTTP requests for a 1 kB page over a laboratory network, using either TLS or our Telex prototype. The prototype's performance was competitive with that of unmodified TLS.

station has a 2.93 GHz Intel Core 2 Duo E7500 processor and 2 GB of RAM. The server has a 4-core, 2.26 GHz Intel Xeon E55200 processor and 11 GB of RAM. The router has a 3.40 GHz Intel Pentium D processor and 1 GB of RAM. All of the machines in our deployment and tests are running Ubuntu Server 10.10 and are interconnected using Gigabit Ethernet.

8.2 Tagging performance

We evaluated our tagging implementation by generating and verifying tags in bulk using a single CPU core on the Telex station. We performed ten trials, each of which processed a batch of 100,000 tags. The mean time to generate a batch was 18.24 seconds with a standard deviation of 0.016 seconds, and the mean time to verify a batch was 9.03 seconds with a standard deviation of 0.0083 seconds. This corresponds to a throughput of approximately 5482 tags generated per second and 11074 tags verified per second. As our TLS throughput experiments show, tag verification appears very unlikely to be a bottleneck in our system.

8.3 Telex-TLS performance

To compare the overhead of Telex, we used our model deployment with two additional clients connected to the router. Our primary client machine (client A) has a 2.93 GHz Intel Core 2 Duo E7500 processor and 2 GB of RAM. The secondary client machine (client B) has a 3.40 GHz Intel Pentium D processor and 2 GB of RAM. For our control, we used the Apache benchmark `ab` [1] to have each of the clients simultaneously download a static 1-kilobyte page over HTTPS. To compare to Telex, we then configured `ab` to download the same page through the `telex_client`. Because the Telex tunnel itself is encrypted with TLS, we configured `ab` to use HTTP, not HTTPS, in this latter case. For the NotBlocked.com used by `telex_client`, we used our server on port 443 (HTTPS) and for Blocked.com, we used our same server on port 80 (HTTP).

We modified `ab` to ensure that only successful connections were counted in throughput numbers and to override its use of OpenSSL's `SSL_OP_ALL` option. This option originally caused `ab` to send fewer packets than a default configuration of OpenSSL, allowing the TLS control to perform artificially better at the cost of decreased security.

We used `ab` to perform batches of 1000 connections (`ab -n 1000`); in each batch, we configured it to use a variable number of concurrent connections. We repeated each trial on our two clients (client A and client B) to get a mean connection throughput for each client.

The results are shown in Figure 4; the performance of the Telex tunnel lags behind that of TLS at low concurrency, but catches up at higher concurrencies. The observered performance is consistent with Telex introducing higher latency but similar throughput, which we posit is due to Telex's additional processing and network delay (e.g., execution of the `fake_accept` ioctl). Both Telex and TLS exhibit diminishing returns from more than 10 concurrent requests, and both start to plateau at 30 concurrent requests. Manual inspection of client machines' CPU utilization confirms that the tests are CPU bound by 50 concurrent connections.

8.4 Real-world experience

To test functionality on a real censor's network, we ran a Telex client on a PlanetLab [24] node located in Beijing and attempted connections to each of the Alexa top 100 websites [2] using our model Telex station located at the University of Michigan. As a control, we first loaded these sites without using Telex and noted apparent censorship behavior for 17 of them, including 4 from the top 10: facebook.com, youtube.com, blogspot.com and twitter.com. The blocking techniques we observed included forged RST packets, false DNS results, and destination IP black holes, which are consistent with previous findings [15]. We successfully loaded all 100 sites using Telex. We also compared the time taken to load the 83 unblocked sites with and without Telex. While this metric was difficult to measure accurately due to varying network conditions, we observed a median overhead of approximately 60%.

To approximate the user experience of a client in China, we configured a web browser on a machine in Michigan

to proxy its connections over an SSH tunnel to our Telex client running in Beijing. Though each request traveled from Ann Arbor to China and back before being forwarded to its destination website (a detour of at least 32,000 km), we were able to browse the Internet uncensored, and even to watch streaming YouTube videos.

Anecdotally, three of the authors have used Telex for their daily Web browsing for about two months, from various locations in the United States, with acceptable stability and little noticeable performance degradation. The system received additional stress testing because an early version of the Telex client did not restrict incoming connections to the local host, and, as a result, one of the authors' computers was enlisted by others as an open proxy. Given the amount of malicious activity we observed before the issue was corrected, our prototype deployment appears to be robust enough to handle small-scale everyday use.

9 Future Work

Maturing Telex from our current proof-of-concept to a large-scale production deployment will require substantial work. In this section, we identify four areas for future improvement.

Traffic shaping An advanced censor may be able to distinguish Telex activity from normal TLS connections by analyzing traffic characteristics such as the packet and document sizes and packet timing. We conjecture that this would be difficult to do on a large scale due to the large variety of sites that can serve as NotBlocked and the disruptive impact of false positives. Nevertheless, in future work we plan to adapt techniques from prior work [18] to defend Telex against such analysis. In particular, we anticipate using a dynamic padding scheme to mimic the traffic characteristics of NotBlocked.com. Briefly, for every client request meant for Blocked.com, the Telex station would generate a real request to NotBlocked.com and use the reply from NotBlocked.com to restrict the timing and length of the reply from Blocked.com (assuming the Blocked.com reply arrived earlier). If the NotBlocked.com data arrived first, the station would send padding as a reply to the client, including a command to send a second "request" if necessary to ensure that the apparent document length, packet size, and round trip time remained consistent with that of NotBlocked.com.

Server mimicry Different service implementations and TCP stacks are easily distinguished by their observable behavior [21, Chapter 8]. This presents a substantial challenge for Telex: to avoid detection when the NotBlocked.com server and the Telex station run different software, a production implementation of Telex would need to accurately mimic the characteristics of many common server configurations. Our prototype implementation does not attempt this, and we have noted a variety of ways that it deviates from TLS servers we have tested. These deviations include properties at the IP layer (e.g. stale IP ID fields), the TCP layer (e.g. incorrect congestion windows, which is detectable by early acknowledgements), and the TLS layer (e.g. different compression methods and extensions provided by our more recent OpenSSL version). While these specific examples may themselves be trivial to fix, convincingly mimicking a diverse population of sites will likely require substantial engineering effort. One approach would be for the Telex station to maintain a set of userspace implementations of popular TCP stacks and use the appropriate one to masquerade as NotBlocked.com.

Station scalability Widescale Telex deployment will likely require Telex stations to scale to thousands of concurrent connections, which is beyond the capacity of our prototype. We plan to investigate techniques for adapting station components to run on multiple distributed machines. Clustering techniques [31] developed for increasing the scalability of the Bro IDS may be applicable.

Station placement Telex raises a number of questions related to Internet topography. How many ISPs would need to participate to provide global coverage? Short of this, where should stations be placed to optimally cover a particular censor's network? We leave accurate deployment modelling for future work.

Furthermore, we currently make the optimistic assumption that all packets for the client's connection to NotBlocked.com pass through some particular Telex station, but this might not be the case if there are asymmetric routes or other complications. Does this assumption hold widely enough for Telex to be practically deployed? If not, the system could be enhanced in future work to support cooperation among Telex stations on different paths, or to support multi-headed stations consisting of several routers in different locations diverting traffic to common recognition and relay components.

10 Conclusion

In this paper, we introduced Telex, a new concept in censorship resistance. By moving anticensorship service from the edge of the network into the core network infrastructure, Telex has the potential to provide both greater resistance to blocking and higher performance than existing approaches. We proposed a protocol for steganographically implementing Telex on top of TLS, and we supported its feasibility with a proof-of-concept implementation. Scaling up to a production implementation will require substantial engineering effort and close partnerships with ISPs, and we acknowledge that worldwide

deployment seems unlikely without government participation. However, Internet access increasingly promises to empower citizens of repressive governments like never before, and we expect censorship-resistant communication to play a growing part in foreign policy.

Acknowledgments

We are grateful to the anonymous reviewers for their constructive feedback, and to Matthew Green for shepherding the work to publication. We also wish to thank Michael Bailey, Drew Dean, Robert K. Dick, Roger Dingledine, Ed Felten, Manish Kirir, Z. Morley Mao, Nadia Heninger, Peter Honeyman, Brent Waters, Florian Westphal, and Xueyang Xu for thoughtful discussions. Ian Goldberg gratefully acknowledges the funding support of NSERC and MITACS.

References

[1] ab: Apache benchmark. http://httpd.apache.org/docs/2.0/programipsets/ab.html.

[2] Alexa top sites. http://www.alexa.com/topsites.

[3] BERNSTEIN, D. J. Curve25519: New Diffie-Hellman speed records. *Public Key Cryptography-PKC 2006* (2006), 207–228.

[4] BONEH, D. The decision Diffie-Hellman problem. *Algorithmic Number Theory* (1998), 48–63.

[5] BONEH, D. J., GENTRY, C., LYNN, B., AND SHACHAM, H. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In *Proceedings of Advances in Cryptography — Eurocrypt 2003* (May 2003), pp. 416–432.

[6] DANEZIS, G., AND ANDERSON, R. The economics of censorship resistance. In *Proceedings of the 3rd Annual Workship on Economics and Information Security (WEIS04)* (May 2004).

[7] DANEZIS, G., AND DIAZ, C. Survey of anonymous communication channels. *Computer Communications 33* (Mar. 2010).

[8] DIERKS, T., AND RESCORLA, E. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), Aug. 2008. Updated by RFCs 5746, 5878.

[9] DINGLEDINE, R. Strategies for getting more bridge addresses. https://blog.torproject.org/blog/strategies-getting-more-bridge-addresses, May 2011.

[10] DINGLEDINE, R., MATHEWSON, N., AND SYVERSON, P. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium* (Aug. 2004).

[11] DOUCEUR, J. R. The Sybil attack. In *Proc. International Workshop on Peer-to-Peer Systems (IPTPS)* (2002), pp. 251–260.

[12] Dynaweb proxy. http://www.dit-inc.us/dynaweb.

[13] FEAMSTER, N., BALAZINSKA, M., HARFST, G., BALAKRISHNAN, H., AND KARGER, D. Infranet: Circumventing web censorship and surveillance. In *Proceedings of the 11th USENIX Security Symposium* (Aug. 2002).

[14] FEAMSTER, N., BALAZINSKA, M., WANG, W., BALAKRISHNAN, H., AND KARGER, D. Thwarting web censorship with untrusted messenger discovery. In *Privacy Enhancing Technologies* (2003), Springer, pp. 125–140.

[15] GLOBAL INTERNET FREEDOM CONSORTIUM. The Great Firewall revealed. http://www.internetfreedom.org/files/WhitePaper/ChinaGreatFirewallRevealed.pdf.

[16] GLOBAL INTERNET FREEDOM CONSORTIUM. Defeat internet censorship: Overview of advanced technologies and products. http://www.internetfreedom.org/archive/Defeat_Internet_Censorship_White_Paper.pdf, Nov. 2007.

[17] GOH, E.-J., BONEH, D., GOLLE, P., AND PINKAS, B. The Design and Implementation of Protocol-based Hidden Key Recovery. In *Proceedings of the 6th Information Security Conference* (Oct. 2003), pp. 165–179.

[18] HINTZ, A. Fingerprinting websites using traffic analysis. In *Privacy Enhancing Technologies*, R. Dingledine and P. Syverson, Eds., vol. 2482 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2003, pp. 229–233.

[19] IP sets. http://ipset.netfilter.org/.

[20] JUELS, A., AND BRAINARD, J. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *Proceedings of the 1999 Network and Distributed System Security Symposium (NDSS)* (Feb. 1999).

[21] LYON, G. *Nmap Network Scanning*. Nmap Project, 2009, ch. Chapter 8: Remote OS Detection.

[22] MURDOCH, S. J., AND DANEZIS, G. Low-cost traffic analysis of tor. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy (SP05)* (2005).

[23] PAXSON, V. Bro: A system for detecting network intruders in real time. *Computer Networks 31* (1999), 2435–2463.

[24] PETERSON, L., ANDERSON, T., CULLER, D., AND ROSCOE, T. A Blueprint for Introducing Disruptive Technology into the Internet. In *Proceedings of HotNets–I* (Princeton, New Jersey, October 2002).

[25] Squid HTTP proxy. http://www.squid-cache.org/.

[26] STEIN, W., AND JOYNER, D. Sage: System for algebra and geometry experimentation. *ACM SIGSAM Bulletin 39*, 2 (2005), 61–64.

[27] THE TOR PROJECT. Tor: Bridges. https://www.torproject.org/docs/bridges.

[28] THE TOR PROJECT. New blocking activity from Iran. https://blog.torproject.org/blog/new-blocking-activity-iran, Jan. 2011.

[29] Transparent proxy support documentation. http://lxr.linux.no/#linux+v2.6.37/Documentation/networking/tproxy.txt, Jan. 2011.

[30] UltraSurf proxy. http://www.ultrareach.com/.

[31] VALLENTIN, M., SOMMER, R., LEE, J., LERES, C., PAXON, V., AND TIERNEY, B. The NIDS cluster: Scalable, stateful network intrusion detection on commodity hardware. In *Proc. 10th International Conference on Recent Advances in Intrusion Detection (RAID '07)* (Sept. 2007), pp. 107–126.

[32] WATERS, B., JUELS, A., HALDERMAN, J. A., AND FELTEN, E. W. New client puzzle outsourcing techniques for DoS resistance. In *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS 2004)* (Oct. 2004).

[33] WOLCHOK, S., YAO., R., AND HALDERMAN, J. A. Analysis of the Green Dam Censorware System. *Computer Science and Engineering Division, University of Michigan 18* (2009).

A Tagging Details

Our system uses an elliptic curve E defined over a field of prime order p. We choose p to be 3 mod 4, so that -1 will be a quadratic nonresidue mod p. (z is a *quadratic residue* mod p if there exists an integer y such that $y^2 \equiv z \bmod p$. Otherwise, z is a *quadratic nonresidue* mod p. Half of

the non-zero elements mod p are quadratic residues, and half are nonresidues.) Let ℓ_p be the bit length of p, and ensure that $2^{\ell_p} - p < \sqrt{p}$. The curve E is defined by the equation $y^2 = x^3 - 3x + b \bmod p$ for a particular value of b.

For some values of $x \in \mathbb{F}_p$, $z = x^3 - 3x + b$ will be a quadratic residue mod p; for those values, $y = z^{\frac{p+1}{4}}$ will be a square root of z and (x, y) will be on the elliptic curve E.

The other values of x will never occur as the x-coordinate of a point on the elliptic curve E; however, for those values of x, $-z$ *will* be a quadratic residue, $y = z^{\frac{p+1}{4}}$ will be a square root of $-z$, and (x, y) will be a point on the "twist" curve E' defined by $-y^2 = x^3 - 3x + b$. We choose a value of b such that both E and E' have prime order over $\mathbb{F}_p$. It is a fact about elliptic curves that the orders o and o' of E and E' will satisfy $o = p + 1 - t$ and $o' = p + 1 + t$, for some $|t| \leq 2\sqrt{p}$.

Define a function $\phi : \{0,1\}^{\ell_p} \times \{0,1\}^{\ell_p} \to \{0,1\}^{\ell_p}$, such that $\phi(r,x)$ is the point multiplication on the elliptic curve (E or E') which contains a point X with x-coordinate x. To compute $\phi(r,x)$, consider r and x as integers expressed as little-endian strings. x will be the x-coordinate of a point $X = (x, y)$ on one of the curves. On that curve, compute $R = r \cdot X$, and output the x-coordinate of R, expressed as a little-endian string. If R is the point at infinity (which happens if and only if r is a multiple of the curve order), $\phi(r,x)$ is undefined. We note that this is the same function (albeit over different curves) as was used by Bernstein in Curve25519 [3].

The tagging protocol is as follows:

Setup Telex selects arbitrary generators of E and E' and publishes their x-coordinates as little-endian strings g_0 and g_1. Since E and E' have prime order, any non-identity element is a generator of those groups. Telex selects a random private key $r \in \{0,1\}^{\ell_p}$, and publishes $\alpha_0 = \phi(r, g_0)$ and $\alpha_1 = \phi(r, g_1)$. If either of those values is undefined because r is a multiple of either group order (this happens with probability less than $2^{2-\ell_p}$), a different value for r can be selected. Telex also publishes hash functions $H_1 : \{0,1\}^* \to \{0,1\}^{\ell_{H_1}}$ and $H_2 : \{0,1\}^* \to \{0,1\}^{\ell_{H_2}}$.

Client tag generation Given a *context string* χ, the client selects a random $s \in \{0,1\}^{\ell_p}$ and a random bit $b \in \{0,1\}$. The client computes $\beta = \phi(s, g_b)$ and $k = \phi(s, \alpha_b)$. (The bit b selects whether the client will be using E or E'.) In the extremely unlikely event (probability approximately $2^{1-\ell_p}$) that s is a multiple of the group order, $\phi(s, \alpha_b)$ will be undefined, and the client can select a different s. The client publishes the tag $\beta \| H_1(k \| \chi)$ and stores the shared secret key $H_2(k \| \chi)$ for later use. Again viewing ϕ as point multiplication, we can see that the generation of the value k is just elliptic curve Diffie-Hellman; we will exploit this fact in the security argument below.

Telex tag inspection Given a context string χ and a purported $(\ell_p + \ell_{H_1})$-bit tag, the Telex station parses the tag as $\beta \| h$ where β is ℓ_p bits and h is ℓ_{H_1} bits. It computes $k' = \phi(r, \beta)$ and $h' = H_1(k' \| \chi)$. If $h = h'$, the Telex station accepts the tag as valid, and outputs $H_2(k' \| \chi)$ as the shared secret key for later use. Otherwise, it rejects the tag as invalid.

A.1 Parameter selection

In our implementation, we use $p = 2^{168} - 2^8 - 1$ (and so $\ell_p = 168$). Using sage version 4.5.2 [26], we searched for an appropriate value of b by randomly selecting candidate values of b until the orders of E and E' both turned out to be prime. This search took only a few minutes on an 8-core computer, and yielded the value $b =$ 1143018135415191678211954030708980203438788563291 74. The curve E has order $p + 1 - t$ and the twist E' has order $p + 1 + t$ (both of which are prime) for $t = -259041875058586799467181 03$. g_0 is the 168-bit little-endian representation of the number 2, and g_1 is likewise of the number 0. The hash functions H_1 and H_2 are both based on the SHA256 hash function; we select $\ell_{H_1} = 56$ and $\ell_{H_2} = 128$, and set H_1 to be the first 56 bits of the SHA256 output, and H_2 to be the last 128 bits of the SHA256 output. The resulting tag length is $\ell_p + \ell_{H_1} = 224$ bits, which is the size of the random portion of a TLS `ClientHello` message.

Choosing $\ell_p = 168$ requires an adversary (under the usual security assumptions for elliptic curves) to perform 2^{84} computations in order to break the tagging scheme by recovering the private key from the public key (and thus violating the DDH assumption below). While we believe this is sufficient, there are a number of methods we can use to guard against even more powerful adversaries. The first is that the key strength ($2^{\ell_p/2}$) can be traded off against the rate of false positives ($2^{-\ell_{H_1}}$) under the restriction that $\ell_p + \ell_{H_1} = 224$. There are also other places [17] one can hide random-looking bits in a TLS session, to increase from the 224 bits we use to hide our tag. Next, we can limit the utility of expending massive effort to recover the Telex private key by having multiple keys that may correspond to time, source, and/or destination. These public keys could be bundled with the Telex client code. Depending on the duration each public key is used, time-based keys would have to be refetched periodically. As an example, a system that switches public keys every hour could bundle 1 million keys, enough to last for over 114 years, in only 42 MB of space.

A.2 Security argument

We must argue that an adversary, given g_0, g_1, α_0, α_1, and a candidate tag τ, cannot determine whether τ was

an output from the above client tag generation algorithm or was just a $(\ell_p + \ell_{H_1})$-bit string generated uniformly at random by a standard TLS client. Parsing τ as $\beta \| h$, we claim that the distribution of β values is only negligibly different from a uniform distribution of ℓ_p-bit values, and also that, under reasonable cryptographic assumptions, given β, an adversary cannot distinguish the correct value of h that would appear in a valid tag from a random ℓ_{H_1}-bit value.

To see the former, consider the distribution of possible values of $\beta = \phi(s, g_0)$ as s ranges over $\{0,1\}^{\ell_p}$. Treating s as a number, this distribution is only negligibly different from that resulting from the range $1 \leq s < o$, where o is the order of E. The latter is the distribution of x-coordinates of a uniformly selected (non-infinity) point of E. Let L_0 be the set of values $x \in \mathbb{F}_p$ such that $x^3 - 3x + b$ is a quadratic residue. Then every value in L_0 appears as the x-coordinate of two points of E, except possibly for up to 3 points whose y-coordinates are 0, which appear only once each. The previous distribution is then only negligibly different from the uniform distribution on L_0. If L_1 is the set of values $x \in \mathbb{F}_p$ such that $x^3 - 3x + b$ is a quadratic nonresidue, then the same argument shows that the distribution of possible values of $\beta = \phi(s, g_1)$ is only negligibly different from a uniform distribution on L_1. The required distribution of β is then negligibly different from the result of selecting a uniform element of L_b where b is a uniform random bit. Since the sizes of L_0 and L_1 are negligibly different, and L_0 and L_1 are disjoint, and the size of $L_0 \cup L_1$ is p, which is negligibly different from 2^{ℓ_p} (as we chose p to be only slightly smaller than a power of 2), our result follows.

To see the latter, we require the Decision (Co-)Diffie-Hellman (DDH and DCoDH) assumptions [4, 5]: that no adversary, given the points P and rP, can distinguish the distributions $\{(Q, rQ)\}$ and $\{(Q, r'Q)\}$ with non-negligible advantage, where P and Q are points on either E or E' and r and r' are selected uniformly at random from their respective domains (or, as above, from $[0, 2^{\ell_p})$). If P are Q are on the same curve, this is DDH; if one is on E and one on E', this is DCoDH. We also need an assumption on the properties of H_1; namely, that for any χ and any bit b, the distribution $\{H_1(\phi(s, \alpha_b) \| \chi)\}$ over all s is indistinguishable from the uniform distribution on ℓ_{H_1}-bit strings. (This is of course true if H_1 is modelled as a random oracle, but seems likely to be true for our SHA256-based H_1 as well.)

An adversary that can distinguish $\{(\phi(s, g_b), H_1(\phi(s, \alpha_b) \| \chi))\}$ from $\{(\phi(s, g_b), \$\}$ (where \$ are uniform ℓ_{H_1}-bit values) can also distinguish $\{(\phi(s, g_b), H_1(\phi(s, \alpha_b) \| \chi))\}$ from $\{(\phi(s, g_b), H_1(\phi(s', \alpha_b) \| \chi))\}$ by our assumption on H_1. He can then distinguish $\{(\phi(s, g_b), \phi(s, \alpha_b))\}$ from $\{(\phi(s, g_b), \phi(s', \alpha_b))\}$ by taking hashes, and $\{(sG_b, sA_b)\}$ from $\{(sG_b, s'A_b)\}$ by taking x-coordinates, where G_b is the elliptic curve point with x-coordinate g_b and A_b is the elliptic curve point with x-coordinate α_b. Writing $Q = sG_b$ and $r' = s's^{-1}$, and noting that $A_b = rG_b$, this is the same as distinguishing the distributions $\{(Q, rQ)\}$ and $\{(Q, r'Q)\}$, given G_b and $A_b = rG_b$, which is impossible by the DDH assumption. Care must also be taken to ensure that the adversary's knowledge of (G_{1-b}, A_{1-b}) does not aid him, but this can also be seen to be true by DCoDH.

In summary, under the DDH and DCoDH assumptions on E and E' and a random-looking-output assumption on H_1, an adversary who does not know Telex's private key r cannot distinguish valid tags from uniformly generated $(\ell_p + \ell_{H_1})$-bit strings with more than a negligible advantage.