Shuai Li* and Nicholas Hopper

Mailet: Instant Social Networking under Censorship

Abstract: Social media websites are blocked in many regimes where Internet censorship is applied. In this paper, we introduce *Mailet*, an unobservable transport proxy which enables the users to access social websites by email applications. Without assuming the Mailet servers are trustworthy, Mailet can support the services requiring privileges without having the complete credential. Particularly, the credential is split and distributed in two Mailet servers, and neither of them can recover the credential alone. To recover the credential in a TLS record message, we propose a highly efficient Galois/Counter Mode(GCM) based secure computation, which can enable the two servers to conceal their separate credential copies in the computation. We implemented a prototype for Twitter.com to demonstrate the usability and security of Mailet.

Keywords: censorship resistance, secure computation

DOI 10.1515/popets-2016-0011
Received 2015-08-31; revised 2015-12-02; accepted 2015-12-02.

1 Introduction

Social Media websites such as Twitter and Facebook have grown to play a prominent role not only in the social lives of their users, but as an important source of news about current events [21], communication hub in emergencies [27], and a coordination mechanism for social and political activism [26]. Correspondingly, governments in many countries have either permanently or temporarily blocked or threatened to block access to these sites; for example the *herdict.org* censorship data site reports at least 10 different countries blocking Facebook at some point in 2014 and 11 blocking Twitter. In response to this blocking, users in these countries often turn to circumvention tools; one survey of circumvention users found that accessing social media sites was the second most common reason for using these tools, with over 70% of respondents citing this intent [1].

In response to the popularity of some circumvention tools, several nation states have deployed technology that blocks access to these tools, through a combination of address blocking, protocol filtering based on deep packet inspection (DPI) and active probing to reduce false positives. This has led researchers to engage in an "arms race" of protocols and attacks for "unobservable transport." Steganographic "parrot" protocols [24, 28, 30] attempt to imitate protocols the censor will be reluctant to block, but many of these schemes were shown to suffer from imitation flaws [15]. Decoy Routing schemes [16, 20, 31, 32] use backbone routers as proxies to imitate connections to arbitrary unblocked hosts, but were shown to require impractically large and targeted deployments in order to avoid availability attacks [18, 25]. "Hide-within" systems [8, 17, 34] attempt to hide both the true destination and the covert nature of circumvention connections by tunnelling connections through popular services such as email, VoIP, and cloud storage. In addition to requiring high bandwidth overhead, these schemes were shown to suffer from detection and blocking attacks stemming from inconsistency between the data volume and loss tolerance of the proxy and cover applications [14].

As a result of this arms race, it is unclear whether there can be a single cover protocol that can handle arbitrary Internet content. An alternative strategy is to develop a small number of systems, each of which is difficult to detect or block when carrying a specific type of content. This strategy is supported by the finding that in China, most circumvention use is motivated by unfiltered search, unfiltered social media access, and video sharing sites [1]. An example of this strategy is Facet [22], which was designed to provide uncensored access to YouTube, Vine and Vimeo by playing the videos over an encrypted Skype call; since these videos are content-consistent the attacks on other "hide-within" schemes do not apply.

Given the important role of social media sites, finding an unobservable and difficult to block transport protocol for these systems is an important next step. Note that these services present several challenges not present

***Corresponding Author: Shuai Li:** University of Minnesota, E-mail: shuai@cs.umn.edu
Nicholas Hopper: University of Minnesota, E-mail: hopper@cs.umn.edu

in the social video context: they are not loss-tolerant, so they cannot rely on voice or video-based channels; they are authenticated, so the system should provide different privileges to different users; and the content provided is private, so it should be difficult for the circumvention to access or modify the content without the user's consent.

In this paper, we present **Mailet**, an unobservable transport which provides unfiltered social website access by using email applications. Mailet servers and clients exchange the text content of a social media website via email. Specifically, the client sends an email to an inbox accessed by the server with the specified service details included; and on behalf of the client, the Mailet server communicates with the social website and emails back the response text, if any. This design guarantees channel consistency and has no imitation flaws. This makes Mailet immune to existing attacks. In addition, Mailet has several desirable features:

Mailet is secure against untrustworthy proxies. The Mailet design enables Mailet servers to provide privileged services without learning the social media login credentials of the client, using a threshold trust approach. In Mailet, the client is allowed to split and distribute the credential to a set of Mailet servers, and each server holds a share of the secret. Without learning the other shares, a single server can not recover the credential alone. When presenting the credential to the social website, Mailet servers should combine the shares privately. However, existing social websites do not support decentralized or privacy-preserving authentication, and expect to interact with clients through a single TLS session. Mailet solves this problem by proposing a highly efficient Galois/Counter Mode (GCM) based secure computation to enable the servers to recover the credential without disclosing their separate credential copies to each other. This computed TLS message contains the complete credential and is consistent with the social website interface. Comparing with the conventional secure two-party computation (2PC), which enables two parties to evaluate a function without revealing their inputs to each other [19, 23, 33], our approach can achieve a speedup of 120. This high efficiency can make a normal Mailet server to support up to 200 simultaneous sessions with each service request being completed in about 1 second.

Mailet users do not require additional software. Users can access Mailet through any standard mail client that supports STARTTLS, or any standard webmail service not controlled by the censor. This resolves the secure distribution or bootstrapping problem common in circumvention software.

Mailet resists proxy enumeration. Users interact with Mailet by sending email to mailboxes on widely-used mail services. Even if the censor learns the IP address of the Mailet servers, blocking direct connections is futile because users and servers never directly communicate. Thus blocking access to Mailet involves preventing users from sending and receiving email from users of all of the popular email hosting services.

We have implemented the Mailet protocol for use with Twitter. We describe this implementation, which we release as open source software[1], and evaluate its performance and security as a circumvention tool both experimentally and analytically. We find its performance is adequate and hope that other organizations will be willing to deploy Mailet servers as well, providing stronger security against server collusion and providing users with a free and secure alternative circumvention tool.

Outline. The reminder of this paper is organized as follows. In section 2, the threat model and design goals are clarified. In section 3, the architecture of Mailet is presented, and section 4 introduces the decentralized credential. Section 5 gives implementation. Section 6 gives the security analysis, and in section 7 experimental results are presented. Section 8 gives the related works, and in section 9, limitations and future works are presented. Section 10 concludes this paper.

2 Threat Model and Design Goals

2.1 Threat Model

The censor's network. We assume a state-level censor that is trying to block inter-state connections related to social websites. The censor can block the social websites by static IP address filtering and keyword filtering, and furthermore, the censor is assumed to be capable of traffic analysis as a means of identifying the usage of circumvention tools. For example, the censor may monitor connections originating within its borders to the POP3, SMTP, and IMAP ports of popular email services, and attempt to identify the circumvention traffic if the traffic pattern deviates from normal email traffic. However, we assume that the censor cannot break the encryption used to protect traffic between email clients and servers, and that some email services based outside the censored network are not corrupted by the censor.

1 https://github.com/magicle/Mailet

Mailet Servers. Similar to other circumventions systems like Tor [11], Mailet users select two servers out of a larger set of possible servers, and rely on these servers not to collude. This mitigates corruption of the Mailet servers by ensuring that (if servers are selected at random), an adversary that corrupts fraction ρ of servers will have probability at most ρ^2 of corrupting both servers that represent a user. However, we must still ensure that if a single server is corrupted, the user is able to circumvent the censor without loss of privacy or availability.

When modeling the actions of corrupted Mailet servers when interacting with users and non-colluding servers, we distinguish between two security goals:

- *Credential Privacy*: Mailet's protocols are designed to ensure that a single server engaging in arbitrary malicious behavior may not compromise a user's social networking credentials, either in the form of passwords or other access tokens which are generated to enable a third-party application to represent the user. The existence of the third-party social media Apps stealing users' data [4] demonstrates the necessity of this attack model.
- *Interaction Integrity:* Since the pair of Mailet servers selected by a user must be capable of acting on her behalf, a malicious Mailet server might seek to modify or drop some of these actions. (For example, register additional applications in the user's name, selectively change the user's status or tweets, or deny service to the user) However, we assume that (similar to systems like Tor) if a server's attempts to engage in these behaviors are detected, the server will be removed from the system. Thus we model servers as *covert* adversaries [7] that can be deterred by mechanisms that detect malicious behavior with respect to integrity, and design Mailet's protocols to ensure that modification of the user's interactions are detected with constant, high probability.

We note that in contrast to a service like Tor, compromising the pair of servers representing a particular user leads directly to compromise of the user's social network credentials; this may motivate system organizers and users to take more care in selecting server operators than might otherwise be the case. Furthermore, we provide users with the mechanism to choose the servers that act on her behalf in case she has more detailed knowledge of the threat she faces.

2.2 Design Goals

The requirements the Mailet design should satisfy are:

- **Unblockable.** The usage of the Mailet system can not be detected by the censor, or blocked without causing significant loss of service to innocent email users. This requires that connections between Mailet server and client should have characteristics similar to that of ordinary email traffic.
- **Real-time Service.** The client can be served in real-time without intolerable delays, as long as ordinary email services also provide good performance.
- **Low Client and Service Overhead.** Neither the user nor the social media site should incur high bandwidth or computation costs when using Mailet.
- **Credential Security.** No single Mailet server should be able to recover the client's credentials even though this information is necessary for the system to complete the service.
- **No Deployment at Client Side.** It should not require the user to install any circumvention tools.

3 Mailet Design

The Mailet system uses email channels to facilitate communication between clients in a censored regime and social media websites. Instead of using email as a carrier for TCP/IP [34], Mailet directly transmits application-level content. This design reduces bandwidth requirements between clients and Mailet servers, while also increasing resistance to traffic analysis and differential channel attacks. The primary challenge associated with this design is protecting the clients' credential information. Social websites require a client to submit authentication credential in order to provide some privileged services, but cannot be relied on to provide a notion of limited delegation; thus for the Mailet system to represent a client it must have authentication credentials for that client's account. This poses a potential security threat to the client, since a malicious or compromised Mailet server could leak or otherwise misuse these credentials.

To protect the clients' credential information, Mailet is designed so that the system can *use* the credential even though no individual server will *know* it. In order to achieve this goal, we introduce a decentralized Mailet server structure, in which sub Mailet servers are controlled by separate parties, and clients distribute credential information among some of them, so that these

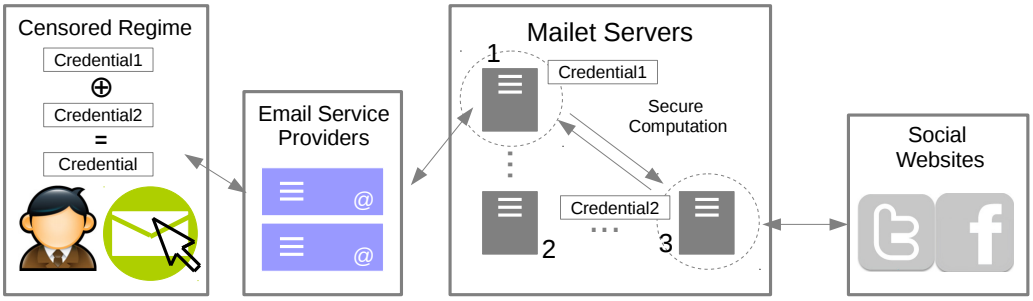


Fig. 1. Mailet System Architecture: the user in the censored regime splits its credential into two copies, which are then distributed to two Mailet servers. This protects the credential of the user, while still being able to fulfill the user's service requests after the two servers running secure computation to recover the credential privately.

servers can collectively represent the user while no individual server can access the credentials. In the following, we introduce the Mailet server design.

3.1 Architecture

Mailet servers and clients communicate with each other by email. Clients send commands and message to be posted to servers via email, and servers use email to deliver site contents to clients. In the system architecture, there are four types of entities:

- Mailet *Clients* are assumed to be able to access an uncompromised email service, but are assumed to have no direct access to the social website, due to censorship or surveillance.
- *Email Service Providers*. The Mailet server and client can have different email service providers, and these providers are assumed to be uncompromised by the censor. We argue that this is a reasonable assumption. Since Mailet servers are outside of the censored regime, they can reach uncompromised email service providers. For Mailet clients, the abundance of independent providers makes it difficult for a censor to compromise or filter all of them.
- Mailet *Servers*. The Mailet design includes multiple servers in order to protect the credential information of the clients. When a client first enrolls in Mailet, it distributes its credential information among two randomly selected servers, which collaboratively present the credential information to the social websites. Each server has its own (set of) email inbox(es) for communicating with clients.
- *Social Websites*. In our paper, we implement the Mailet system for Twitter. Social websites only accept direct connections via TLS, which cannot include third parties; thus Mailet's protocols must be designed to match this interface.

For connections requiring no credentials, such as searching tweets, the client can select a random Mailet server. For example, if the user wants to search for tweets, it can send an email with the command "searchtweet" plus the keyword to the server, which will search by the keyword and email the results back to the client. However, most commands require the user to prove its identity to the website by providing its credential. In these situations, Mailet servers use the decentralized credential mechanism described in the next section to allow transmission of the credential to the social website while preventing its recovery by any Mailet server.

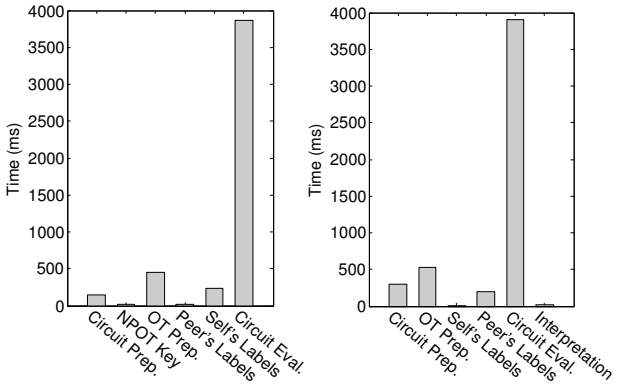
4 Decentralized Credential

Mailet clients need to share their credentials with Mailet servers when they require privileged services from social websites, so that the Mailet servers can present the credential to websites on behalf of their clients. While this concept of content proxy enhances Mailet in terms of unobservability and usability, it poses a potential threat to its clients' credentials if the Mailet servers are honest-but-curious or even malicious.

To protect the clients' credential, we propose the decentralized credential mechanism. Instead of "putting all the eggs in a single basket", Mailet design allows a

Category	Mailet Server A	Mailet Server B
OT Prep.	13.1	8.97
Label Transfer	0.79	0
OT Label Transfer	14.8	21.26
Circuit Evaluation	6631.13	0
Total	6659.82	30.23

Table 1. 2PC Downstream Bandwidth Consumption (KB)



(a) Mailet Server A **(b)** Mailet Server B

Fig. 2. 2PC Time Cost Breakdown: Mailet server A holds the cryptographic keys of the TLS session and a part of the TLS plaintext, while Mailet server B having the other part of the plaintext. The computation ends up with a valid TLS message having the client's genuine credential. This TLS message is then presented by server B to social websites to finish the service request. RC4-SHA is used as the cipher suite of the TLS session.

Mailet client to split its credential into two copies, and distribute them in two separate servers (which are randomly selected by the client). For each server, its holding of one copy cannot enable it to recover the original credential. This decentralized credential design effectively protects the client's credential.

A challenge in decentralized credential mechanism is how to *privately* combine and represent the credential to social websites. In other words, the Mailet servers should collaboratively recover the original credential, and include it in a TLS connection to the social website, while still preventing each other from learning the other copy. This task is usually regarded as a secure two-party computation problem: two parties (Mailet servers) holding separate secret inputs (the credential copies) evaluate a common function (a TLS record message) without disclosing their inputs to each other. However, since a TLS record message in Mailet is usually large (several hundred bytes), a standard two-party computation is too costly. We implemented a credential recovery by using an optimized 2PC algorithm, and the costs (time, CPU and memory usage) are shown in Figure 2 and Table 1.[2] The results show that 2PC has to take nearly 6 seconds to finish and consumes about 6 MB bandwidth between Mailet servers. In addition, it uses about 90% CPU and 9% memory for each computer with a Quad-Core processor and 4GB memory.

To overcome this challenge, we propose a novel GCM based Credential Recovery (GCM-CR) approach to *secretly* combine the decentralized credential without using the high overhead secure two-party computation. This design uses Galois/Counter Mode (GCM) cipher suite in the TLS connection, and takes the advantage of Encrypt-then-MAC (EtM) of GCM mode to compute a valid TLS record message. Since this scheme involves no 2PC, a valid TLS record can be computed efficiently. Comparing with the conventional 2PC, our approach can achieve a speedup of 120. In our context, the speedup is equal to $\frac{L_{2PC}}{L_{GCM-CR}}$, where L_{2PC} and L_{GCM-CR} are the latency of the traditional 2PC and the GCM-CR approach, respectively. Furthermore, we propose Checking-by-Sampling (CbS) mechanism to enhance the Mailet design against a malicious Mailet server crafting malicious messages on behalf of the client. Before delving into the details about decentralized credential, we first briefly introduce the Transport Layer Security (TLS) protocol.

4.1 TLS Protocol

Social media users connect to the website server by TLS protocol. A TLS handshake protocol is firstly executed by both sides to negotiate the cipher suite to use, exchange random numbers, and agree on a common premaster secret by public-key cryptography. Afterwards, under the negotiated cryptographic parameters, users communicate with the website server by encrypted TLS application data, which may include the user's credential or the message to post. Then for an attacker, it can neither learn the application data nor forge a malicious message on behalf of the user or the website server.

The TLS message type can be figured out by any third party, which enables a Mailet server to intercept

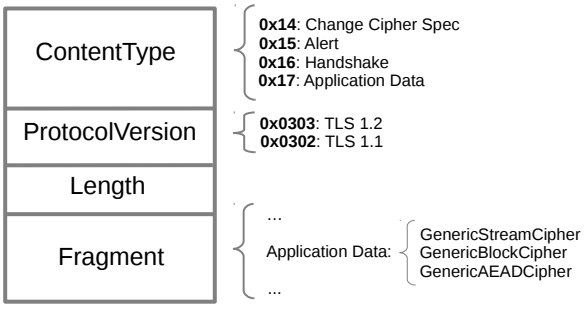


Fig. 3. TLS Record Format: the general case

2 We adopted the FastGC framework[19] and implemented the 2PC in 1600 lines of Java.

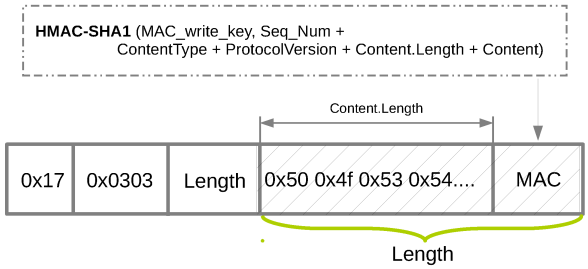


Fig. 4. Data Application Format: under stream cipher encryption with HMAC-SHA1 as the MAC algorithm

at the right time. The method is to examine the ContentType field. Figure 3 shows all the possible TLS message types and their ContentType field values. Besides of ContentType, a TLS message also contains ProtocolVersion field, Length field (the length of the Fragment in byte), and Fragment field. An example of the TLS format for application data under stream cipher encryption is shown in Figure 4. The content field is the encrypted application data, and the MAC field is the Message Authentication Code (MAC) for the application data, a sequence number, ProtocolVersion, ContentType, and length of the application data. The MAC is computed before the application data is encrypted. More details are given in the Appendix.

4.2 Credential Sharing and Recovering

Credential Sharing. Mailet's design incorporates multiple servers to store the client's credential information. For example, in Figure 1, the No. 1 and No. 3 servers are randomly chosen by the client to hold credential shares $Cred_1$ and $Cred_2$, separately. The credential generation method is as follows. First, the client generates a random string as the credential $Cred_1$, whose length is equal to that of the original credential. By XORing credential $Cred_1$ and the original credential, the client obtains the other credential $Cred_2$. Finally, the client distributes these two credentials to the chosen servers. Now neither of these two servers can recover the client's credential alone.

Credential Recovering. When the client initiates a command that requires credential use, the two Mailet servers S_1 and S_2 storing shares of the client's credential $Cred$ must collaborate to conduct a TLS session including $Cred$ with the social website server. S_1 and S_2 could recover $Cred$ simply by jointly computing $Cred_1 \oplus Cred_2$, but this would not be secure since then both servers would know $Cred$.

The approach in this section for privately generating the *valid* TLS record message relies on the *Initiator-Interceptor* structure. Particularly, we assign the servers asymmetric roles: one server is the *Initiator*, and the other is the *Interceptor*, shown in Figure 5. The Initiator initiates the client side of the TLS handshake with the social website server, using the Interceptor as a proxy to pass messages to the server, so that from the point of view of the social website, the connection originates from the Interceptor. At the conclusion of the handshake, the Initiator and the social website server share symmetric keys, so the Interceptor is unable to decrypt the traffic passed to the social website.

After the TLS handshake, the Initiator continues to forward TLS records through the Interceptor to initiate the application-level session; once the TLS record containing only $Cred_1$ arrives at the Interceptor, the Interceptor holds on this message, and regenerates a valid TLS record having $Cred = Cred_1 \oplus Cred_2$ with the help of the Initiator. Finally, the Interceptor sends the regenerated message instead of the original TLS message. The approach to regenerating a valid TLS record is described in the following.

4.3 GCM based Credential Recovery

We introduce the credential recovery without involving 2PC. The GCM-CR approach uses the cipher suites of GCM mode in the TLS session, and takes the advantage of GCM's stream cipher property and EtM feature.

Galois/Counter Mode (GCM). GCM is an operation mode for symmetric key block ciphers. It is an authenticated encryption algorithm, and can provide data confidentiality and integrity simultaneously. Due to its

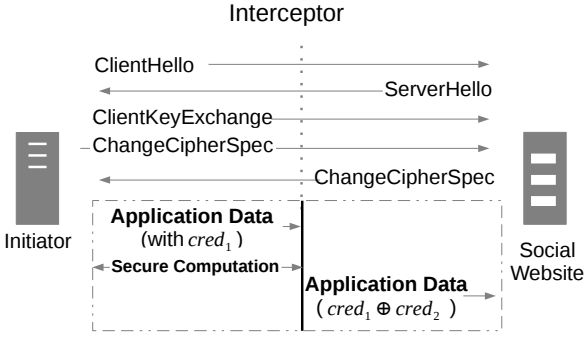


Fig. 5. Initiator and Interceptor Structure: the Interceptor intercepts the TLS application data from the Initiator to the social media website. By collaborating with the Initiator, it regenerates a valid TLS application data which has the genuine credential.

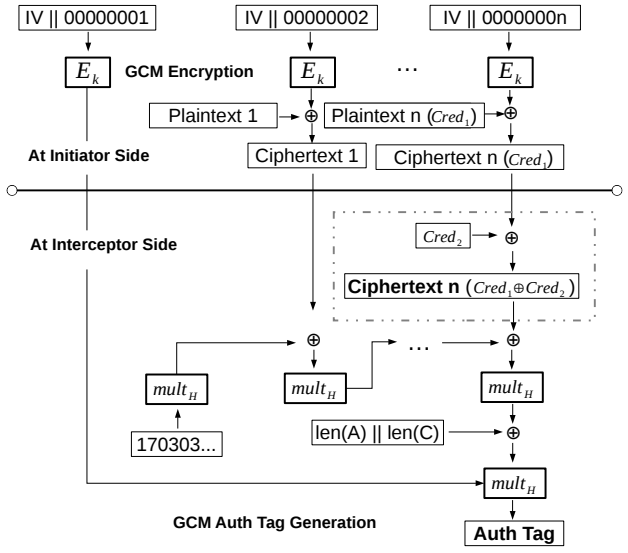


Fig. 6. Credential Recovery with GCM Mode: the Interceptor receives H and $E_k(IV||00000001)$ from the Initiator and creates a valid Auth Tag for the ciphertext including the genuine credential.

high speed with low cost and low latency, GCM mode has been included in TLS cipher suite list, and is widely implemented by popular website servers.

Recover the Credential in Ciphertext. For the encryption, GCM resembles the counter mode encryption, and turns a block cipher into a stream cipher. This makes the credential recovery in the ciphertext convenient. For the Initiator, it encrypts the TLS record message with $Cred_1$ in place of the original credential $Cred$, and passes this message to the Interceptor. The Interceptor can locate the credential $Cred_1$ with the help of the Initiator, and XOR $Cred_2$ with the ciphertext bytes in this location, so that when the website server decrypts the TLS message, the plaintext will contain $Cred$. Since the Interceptor does not know the symmetric key, it cannot recover $Cred$, while the Initiator does not see the completed record, and also cannot recover $Cred$. However, recovering the credential in the ciphertext/plaintext alone is not sufficient. The Interceptor has to generate a correct MAC to make the TLS record message valid.

Validate the TLS Record. In GCM mode, the plaintext is first encrypted, then an authentication tag is computed based on the ciphertext by a GHASH function. This Encryption-then-MAC (EtM) property enables the Interceptor, which has access to the ciphertext, to compute a valid authentication tag without having to do a 2PC with the Initiator. The authentication tag generation is shown in Figure 6. Note that the Initiator should share the H (the encryption of 128 bit 0s) and $E_k(IV||00000001)$ (the encryption of the first counter) with the Interceptor. This does not break the security of the cryptographic system. It leaks neither the TLS session key nor the Initiator's credential $Cred_1$.

Though GCM-CR can effectively protect the clients' credentials in the honest-but-curious attack model, it cannot prevent a malicious server from corrupting the protocols. Both Initiator and Interceptor can maliciously flip the bits in other fields of the TLS record message without being noticed by each other, so that they can change the App ID to be authorized or the message to be posted, etc.. In the following, we give solutions to prevent a Mailet server from crafting malicious messages.

4.4 Interaction Integrity

By applying the decentralized credential mechanism, a corrupted Mailet server is prevented from knowing the user's credential. However, this server might seek to modify the TLS message in a covert way to manipulate the outcomes of the protocol. An incorrect tweet may be posted, or a wrong third-party App is authorized because of this attack. In this section, we propose approaches to detect and prevent such attacks.

A Corrupted Mailet Interceptor. This Interceptor may flip the bits of the TLS ciphertext to manipulate a HTTP field value in the GCM-CR. To prevent such attacks, Mailet can randomize the order of fields in requests, and pad requests with separator strings of random length, making it difficult to predict the location of the desired field. In addition, an Initiator can screen the response from Twitter for a corrupted Interceptor. For example, when posting a tweet, the Initiator receives a response including the posted tweet ID, which allows it to retrieve the tweet. An inaccurate tweet indicates the presence of a corrupted Interceptor.

A Corrupted Mailet Initiator. Violating the interaction integrity is much easier when the corrupted Mailet server takes the role of the Initiator. This is because a malicious Initiator can craft any arbitrary plaintext of the TLS message in GCM-CR. In order to enhance Mailet design against a malicious Initiator, Checking-by-Sampling (CbS) mechanism is proposed. Instead of initiating a single TLS session with the website server, the Initiator starts n parallel sessions which are all passed through the Interceptor. For all these sessions, the Initiator cuts off the ciphertext of its credential copy

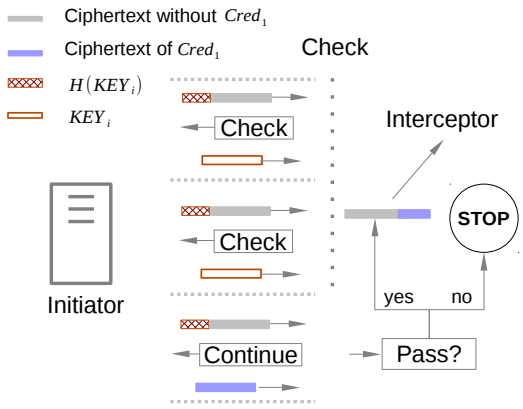


Fig. 7. Checking-by-Sampling: the Interceptor randomly chooses $n-1$ sessions to check the correctness of the non-credential HTTP fields. The commitment $H(KEY_i)$ is required to force the Initiator to provide the true TLS session key in the latter phase.

and passes only the rest of the TLS record messages with n commitments to corresponding TLS session keys. The commitment for each TLS session can be the cryptographic hash of the session key. For the Interceptor, it chooses $n-1$ out of n TLS sessions to open by requesting the Initiator to provide $n-1$ corresponding session keys. The Interceptor checks the commitments, and if malicious messages are detected, it stops the collaborative computation with the Initiator. Otherwise, the Initiator is believed to be honest and the only TLS session left is used for credential recovery. Before doing GCM-CR, the Initiator should complete this TLS record by providing the Interceptor with its credential ciphertext.

This enhancement can practically prevent an malicious Initiator. Even for a smart malicious Initiator, it only has $1/n$ probability to succeed for each session. Being malicious for multiple sessions, the malicious Initiator would be probably detected and reported.

5 Implementation

This paper implements the Mailet design for Twitter.com, and the implemented system can provide tweet posting, reading and searching as well as decentralized authorization for the client. We implemented the GCM-CR module in 1900 lines of Python, and the Mailet server in 1636 lines of Python. It is worth noting that our Mailet design is not limited to Twitter. Since many social websites (such as Facebook) share the similar authentication mechanism with Twitter, our implementation can be extended to support these websites.

5.1 Mailet Server Implementation

Interceptor in Mailet Design. We implemented a socket proxy as the Interceptor in Mailet. This proxy establishes an encrypted socket connection with the Initiator, and relays the socket level messages between the Initiator and the Twitter server. It identifies and intercepts the application data packet by inspecting the ContentType field of the encrypted TLS message, and does the GCM-CR approach with the Initiator. After the GCM-CR session, it sends the computed result to the Twitter server.

Initiator in Mailet Design. The Initiator adopts the TwitterAPI to complete the non-privileged services. TwitterAPI is a Python package to support the Twitter's REST and Streaming API with OAuth 1.0 and OAuth 2.0. We implemented the Initiator for reading and searching tweets. Note that these services do not need the user to provide its credential.

Session Cookie. For posting tweets and retweeting, the Twitter website only accepts session cookies as the authentication approach. Even the user is providing its credential such as its username and password in these services, the Twitter website server only responses with the session cookie which is expected to be used in authentication. This complicates the decentralized credential. In our implementation, the Interceptor XORs a random string in the auth_token field of the Twitter server's response, splitting the auth_token field on the fly. After the Initiator receives this packet, it decrypts without checking the MAC of this packet. At this point, the Initiator and the Interceptor holds separate auth_token copies. When posting a tweet or retweeting, the Mailet servers use GCM-CR to recover the original auth_token in the TLS record message.

In the cookie splitting, the Initiator can no longer check the MAC of the website's response containing the session cookie. This may make the Mailet design vulnerable to an external attacker. Fortunately, as the Initiator and Interceptor collaboratively represent a user, the Interceptor can do the MAC check instead. The approach is the same as GCM-CR. Besides splitting the session cookie, the Interceptor uses the original ciphertext and parameters H, $E_k(IV||00000001)$, which are passed by the Initiator, to compute the correct MAC. If the two MACs are not equal, the results from session cookie splitting would be dropped. Furthermore, it is worth noting that the connections between Initiator and Interceptor are secured by TLS, leaving an external attacker no chance to change the packet when this

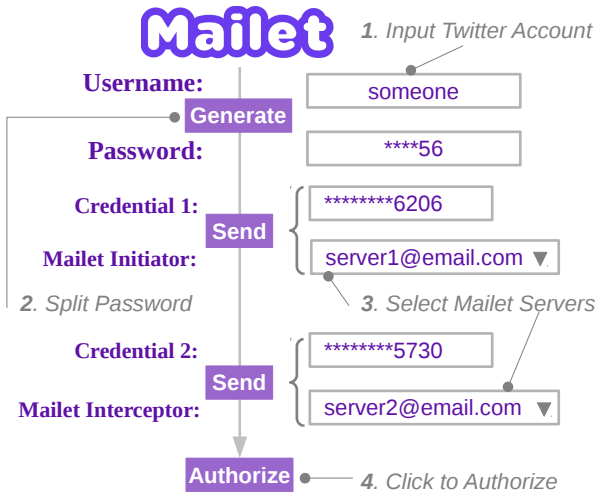


Fig. 8. Mailet Authorization GUI

packet is transmitted from the Interceptor to the Initiator. Therefore, we argue that checking MAC at the Interceptor protects against an external attacker.

TLS Session Information Extraction. In Mailet design, the Initiator should know the client write key, MAC client write key, and TLS plaintext messages in the TLS session. Since the Python `ssl` and `socket` adopt the OPENSSL library [5] to start the TLS session, we modified the "ssl/t1_enc.c" in OPENSSL to write the keys and the plaintexts in a file specified by the Initiator. We also defined the macro SSL_DEBUG, TLS_DEBUG, and KSSL_DEBUG to print out the TLS session information for debugging.

5.2 User Interface Design

This part introduces the user interface design in Mailet. We first introduce the Graphical User Interface (GUI) design for Mailet authorization.

Authorization GUI. The Mailet client can use the authorization GUI to authorize a Mailet server. The GUI is written in HTML and JavaScript, and is shipped to the client as an email attachment. The client can open the GUI in its web browsers, or uses the GUI directly in the email if the email application supports JavaScript. The authorization GUI is shown in Figure 8.

Firstly, the Mailet client inputs its Twitter username and password in the GUI; then it can split the password by clicking the "Generate" button. In the third step, the client chooses the Initiator and the Interceptor in the Mailet server pool, and clicks "Send" buttons to create autofilled emails to distribute the shares of the password. Last, the Mailet client clicks the "Authorize" button to send the authorization request by an autofilled email. Note that some buttons (such as "Generate" and "Send") can be combined to be a single button.

Email Interface. The Mailet client communicates with the Mailet servers by email messages. The Mailet implementation defines the interface for the email:

- The email subject consists of the Mailet commands and the parameters, with a colon separating the two fields. Take retweeting as an example. The client sends an email with the subject "*retweet:ID*", where *retweet* is the Mailet command for retweeting, and the *ID* is the parameter specifying which tweet the user is retweeting. All supported commands in the current implementation are listed in Table 2.
- The email body is defined to carry the text input of the client. It is used when the client posts, searches for, or replies to a tweet.

But a Mailet client does not have to access this interface by manually filling emails. Instead, it can use the Mailet autofilling feature, which is introduced in the following.

Mailet enables a client to autofill the email subject. The Mailet server encodes its emails in HTML language with navigation links for the client. For each link, the `mailto` URL scheme is exploited to autofill the email address and the subject. An example is shown in Figure 9. For each tweet, there is a retweet link created by the server to include the *retweet* command and the tweet's ID. When the client clicks this link, the subject will be autofilled.

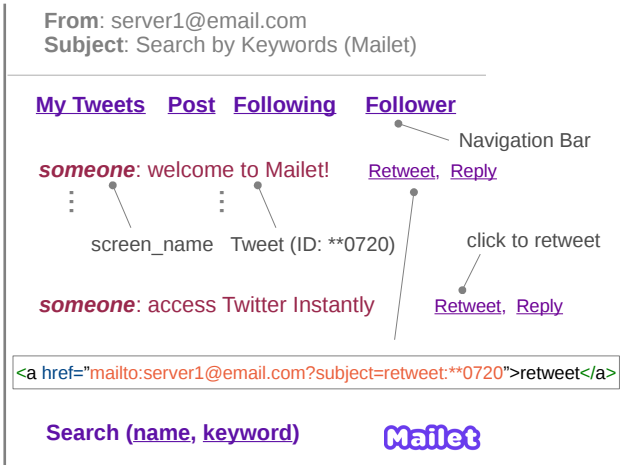


Fig. 9. Navigation Links for Email Autofilling

Email Subject	Email Body	Service
authorize	n/a	Authorization
password	Credential	Cred. Distribution
mytweet:count	n/a	My Tweets
posttweet	the tweet text	Post Tweets
searchbyname	screen_name	Search (Name)
searchbykey	Keywords	Search (Keyword)
deletetweet:ID	n/a	Delete Tweets
following:ID	n/a	Following List
follower:ID	n/a	Follower List
retweet:ID	n/a	Retweet
tweetreply:ID	reply text	Reply

Table 2. the Email Interface

Email Security. A recent work [12] provides the evidence that a state-level attacker, which is in control of the network, can strip out a client's STARTTLS command when this client connects to a mail server, forcing the email contents and attachments to be transmitted in plaintext. This attack would enable a censor to detect and block the usage of the Mailet service. One strategy to defeat such an attacker is to encrypt the emails (by PGP [9], for example). Another strategy is to exploit the pinning approach [13] to disable the attack.

6 Security Analysis

In this section, we informally analyze the security of the Mailet protocols.

6.1 Previous Attacks

Attacks by Imitation Flaws. Mailet uses email protocols and applications directly rather than imitating the email protocols as in parrot circumvention systems. As a consequence, there are no imitation flaws in Mailet, and the censor cannot detect Mailet by this attack.

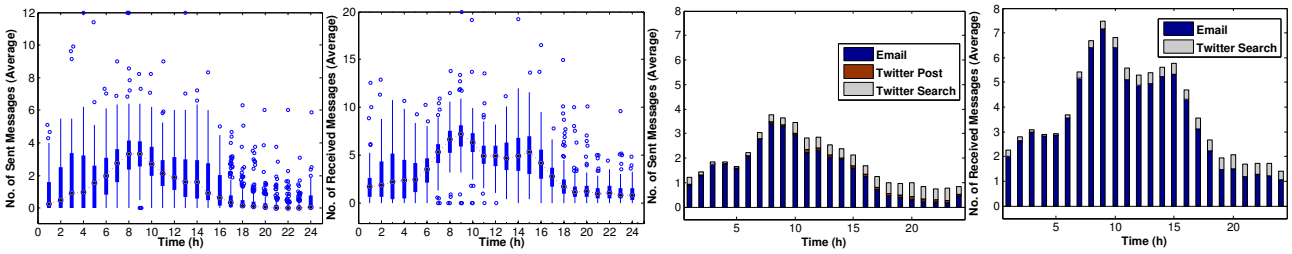
Attacks by Channel and Content Mismatch. Recall that Geddes *et al.* citecoveracks introduced the channel mismatch and content mismatch attacks. Channel mismatch attacks occur when the censor interferes with potential cover protocols to stall or block proxy connections. When the cover protocols can tolerate packet loss or packet delay, the censor's interference only has limited impact on the protocol. However, a proxy connection with the opposite channel feature may be stalled. Mailet exploits the fact that email and twitter conversations have similar loss tolerance and compatible delay tolerance to avoid this attack. Content mismatch occurs when the nature of the proxy traffic causes differences in the statistical properties of the cover channel compared to typical usage. In general, Mailet transactions and email conversations do not differ in terms of content, but may have variations in size and frequency of messages; we consider traffic analysis on these features further in section 6.3.

Denial of Service Attack. Since the email addresses of Mailet servers are public, the censor may try to disable the Mailet service by Denial of Service (DoS) attacks. To defend against the DoS attacks, a Mailet server can enforce usage limitations on each client ID, and it can also use CAPTCHA [6] or puzzles to defend against a Sybil identity attack.

6.2 Untrustworthy Mailet Servers

User's Credential Privacy. In general, under standard cryptographic assumptions on the security of the AES block cipher, credential splitting and detection of key reuse attacks are sufficient to prevent non-colluding malicious Mailet servers from gaining any information about a given user's credentials. One exception is that like most standard encryption protocols, the length of a credential is leaked; since credentials might include passwords, this could reduce the effort required to guess the user's credential by brute force. Fortunately, the Mailet client can conceal its credential length on purpose. Let's denote the length of the credential as θ. Instead of having the genuine credential as the input, a Mailet client can add $n - \theta$ ampersand separators to the end of the credential, padding to a length n sufficient to obscure the credential length, After Mailet servers recompute the credential in the TLS message, these added ampersands are taken as separators in the HTTP context and are ignored. In this way, the Mailet client can prevent the Mailet servers from learning the length of the credential.

Privacy of the User. The Mailet servers act as man-in-the-middle (MitM) when providing privileged access to a social media site. Thus, the servers have the chance to know what the user posted, or who she replied to, etc.. This may raise concerns about privacy violation. Fortunately, due to the open nature of Twitter, the messages or replies are not private in most situations. For example, everyone can see the tweets and replies of Twitter users without having any permission. Even for



(a) Normal Users (Send) **(b)** Normal Users (Receive) **(c)** Mailet Clients (Send) **(d)** Mailet Clients (Receive)

Fig. 10. the Daily Statistics for the Genuine Email Users and the Mailet Clients

the social media websites in which users' privacy is highlighted, we can protect a private field by splitting and distributing it to the two Mailet servers. By using GCM-CR, the field can be recovered secretly, as is adopted for protecting the credential. As a consequence, a single Mailet server cannot learn the private field alone. Except the private field and the credential, the Initiator and Interceptor can also learn other parts of the HTTP message. We argue that they only learn low level traffic parameters and do not violate the user's privacy.

Malicious Mailet Servers. We analyze the possible malicious application-level behaviors a Mailet server could launch and describe countermeasures that can detect these attacks, mitigating them in the context of covert adversaries.

Mailet servers can be malicious and may launch denial-of-service attacks. They may halt the collaborative credential recovery process, or corrupt credential copies in order to fail the service. Since there is no user input error involved in these processes however, they will result in failures that can be observed by the other Mailet server in a given interaction. Such attacks are thus easily detected.

A malicious server may attempt to corrupt the non-credential fields of a TLS message in order to post a selectively altered message, authorize a malicious App, or modify a user's social profile. The Checking-by-Sampling mechanism described in section 4.4 provides a high probability of detection if the Initiator attempts such an attack. The Interceptor may also attempt to maul the ciphertext of a request in order to carry out such an attack. To minimize the potential damage caused by this attack, Mailet randomizes the order of fields in requests, and pads requests with separator strings of random length. Unless the Interceptor predicts the exact position of the field it will be unable to create a semantically meaningful request; even if the request is semantically meaningful, the response sent to the Initiator will indicate an attack, since the outcome implied in the response will not match what is requested by the Initiator. Thus attacks by either server on the integrity of the social network interactions will be detected by the other server with high probability.

6.3 Traffic Analysis

The censor may try to detect Mailet connections by traffic analysis. For each communication session, since Mailet is content consistent, there is little chance for the censor to detect the Mailet session. The censor may also try to detect Mailet by the traffic pattern of an immediate reply email following the sent email. We argue that, for genuine email service, the auto reply feature also leads to such pattern. In addition, for many services such as post tweets and retweets, Mailet does not have this traffic pattern. Therefore, the detection is believed to have high false positives and negatives.

For multiple communication sessions, the censor may try to detect Mailet by inspecting the frequency of emails sent and received per day. In our experiment, we investigate the email patterns over a day for both the Mailet clients and the genuine email users. The experimental results show that, after the Mailet client uses its emails to access Mailet, the email patterns are still in the normal range of the genuine email users. This demonstrates that Mailet can defend against such attacks.

Experimental Results. This section measures the daily email patterns for the genuine email users and the Mailet clients. The datasets used in the experiment are introduced as follows.

Email Dataset. In this experiment, we use the Enron email dataset [3]. This dataset contains 517,425 emails from 151 users, and most of the users are senior management at Enron. However, there is no explanation about whether the dataset contains all the emails

for each user over the period. Therefore, we cannot summarize how many emails are sent or received daily on average.

Fortunately, this dataset still captures the probability distribution $p(u,t)$ over a day. Here, u denotes the user ID, and $p(u,t)$ represents what percentage of emails the user u sent in time interval t everyday. Besides, according to the email statistics report[2], an average business user sent 36 emails and received 85 emails per day in 2014. We then combine these two results to get how many emails are sent or received in each time interval in a day for a genuine email user.

Mailet Usage Dataset. This dataset includes the timestamps of the emails sent or received when a particular user utilizes the Mailet service. To simulate the Twitter users and capture the users' habits of posting, replying, and retweeting tweets, we randomly selected 100 Twitter users and collected their most recent 200 tweets. Specifically, we generated a random integer to be the Twitter user ID, and included this ID if it is valid and has more than 100 tweets (we excluded inactive users from our dataset). We also excluded the users who do not have time zone information. Last, the Coordinated Universal Time (UTC) was converted to the local time.

We also simulate the search habits of the Twitter users. According to[29], twitter users posted 340 million statuses and made 1.6 billion search queries per day. We use this ratio to obtain how many search requests are made by the user through a day. Here, we make an assumption that the way the user uses Twitter is stable. In other words, the ratio of posts to searches is consistent through a day. Though this dataset does not incorporate all the activities of the clients (such as authorization), it includes most of the activities of a typical Mailet client.

The daily statistics for the genuine email users is given in Figure 10. The time interval is set to 1 hour, and the number of the emails sent in each time interval is shown in boxplot. The central mark for each box is the median, and the edges of the box are the 25th and 75th percentiles. The whiskers reach the most extreme data points which are not considered to be outliers. The Figure 10(a) shows that the users from the Enron dataset sent more emails between 8am to 10am, and less emails during the night. The Figure 10(b) shows that the users received more emails than they sent, but the daily statistics of receiving emails is similar with the pattern shown in Figure 10(a).

The daily statistics for the Mailet client is also shown in Figure 10. In the experiment, we assume the email account of the Mailet client is also used for genuine email services, and we summarized how many emails an average Mailet user would send through a day. The figure shows that comparing with the normal emails, the number of emails used for the Mailet service are much less. In other words, the daily statistics of the client does not change significantly after the client uses the Mailet service. Considering the variations of the daily statistics for different users (which is shown in Figure 10(a)(b)), it is difficult for the censor to detect the Mailet session by inspecting the daily statistics. It is worth noting that though the Mailet clients send a bit more emails at midnight than genuine email users, which may make them detectable, it is believed that the bias of the email dataset leads to this problem. Since the emails in the dataset are from business email accounts, the dataset is biased in the sense of incorporating less midnight emails. Consequently, the censor is still believed to be unable to detect the Mailet session by email daily statistics.

7 Performance Analysis

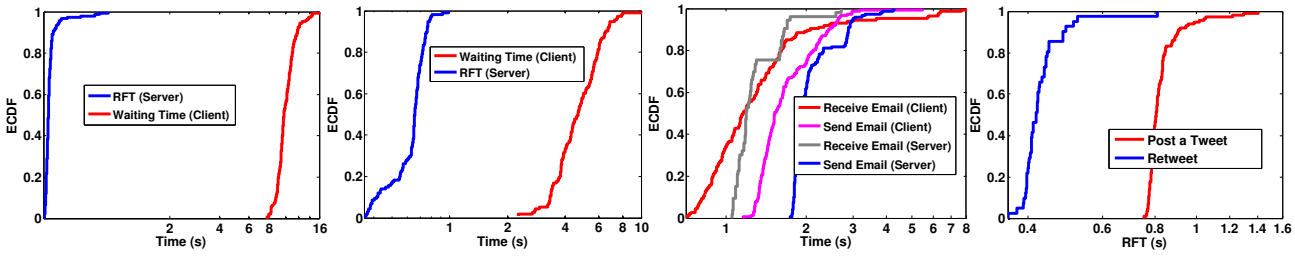
This section measures the performance of the Mailet system by its CPU, memory, and time cost, and demonstrates the feasibility of Mailet.

7.1 Mailet Server Overhead

This part evaluates the Mailet server's overhead by CPU and memory usage. Before giving the experimental results, we first introduce the experiment setup.

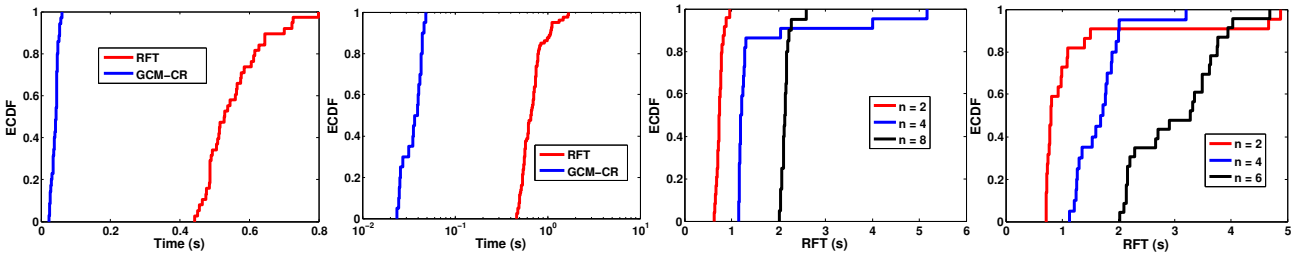
Experiment Setup. We started our experiment on two desktop computers A and B, which were connected by a local area network (LAN). The computers are Dell Precision T1500s, with the Ubuntu 14.04.1 LTS operating system. The processor for each computer is an Intel Core i7 CPU 860 (Quad-Core) 2.80GHz. The memory size for computer A is 8GB, and for computer B it is 4GB. They both have 8GB for swap space. The Initiator resided at B, and A was used as the Interceptor. The password in the credential is 10 bytes long, and the account is 17 bytes long.

We used a laptop as the Mailet user, which is a Lenovo Thinkpad T400 with an Intel T9900 processor and 6GB memory. This user sent service requests to the two Mailet servers over the Internet, and the services included authorization, posting a tweet, and retweeting. At the servers' sides, after receiving a request, they



(a) Retrieve Self's Own Tweets **(b)** Search by Keywords **(c)** Send/Receive Emails **(d)** Comparison: Twitter API

Fig. 11. Time for None-Privileged Mailet Services (s): (a) and (b) give the ECDF of a client's waiting time and the Mailet server's Request Fulfillment Time (RFT); (c) shows the ECDF of the email channel's delay; (d) represents the ECDF of the RFT when the authorized Mailet server fulfilled the privileged services by the Twitter APIs as a comparison with the GCM-CR based approach.



(a) Retweet **(b)** Post a Tweet **(c)** Post a Tweet with CbS **(d)** Retweet with CbS

Fig. 12. Time for Privileged Mailet Services (s): (a) and (b) show the ECDF of Mailet servers' Request Fulfillment Time (RFT); (c) and (d) measure the ECDF of RFT when CbS is adopted.

started to log the accumulative CPU and memory usages of the Mailet system until this request is fulfilled. The final measurement results are the average of multiple trials. In addition, the system cost with Check-by-Sampling mechanism was also logged as a comparison.

Evaluation Results. Table 3 gives a summary about the CPU and memory usage for the Initiator and Interceptor. In the table, the cost of the services (authorization, posting a tweet, and retweeting) is the cost of the Initiator and the Interceptor handling a single service request. Note that this measurement excludes the cost of the email clients. It shows that the whole Mailet system (including email clients) only consumes less than 5.3% of the CPU and at most 1.0% of the memory space. If the Mailet servers adopt the CbS mechanism to avoid a malicious Initiator, the overhead still remains low. Take retweet as an example. When no CbS approach is applied, the CPU usage is 2.0% for the Initiator and 1.8% for the Interceptor. When CbS is applied with $n = 8$, the CPU usage only rises to 3.15% and 2.0%. In both cases, the Initiator consumes 0.1% memory while the Interceptor occupies 0.2% memory. These results show the high efficiency of our GCM-CR approach.

7.2 Service Measurement

This part gives the performance analysis for Mailet. In the experiment, we measured the Mailet service by the following metrics:

- *Mailet User's Waiting Time.* This is the period of time between when a user makes a request and when this user receives the response from the Mailet servers. This time includes the time of sending and receiving the emails at both sides of the Mailet user and the server, and the time of the server fulfilling the request by the decentralized credential or Twitter API calls.
- *Request Fulfillment Time (RFT).* RFT is the period of time that the Mailet server needs to handle a request. For non-privileged services, RFT is the time that the server takes to complete the Twitter API call. For privileged services, RFT refers to the time cost of fulfilling a request by the decentralized credential mechanism. Note that RFT excludes the time cost of the email clients.
- *GCM-CR Time.* This refers to the period of time that the Interceptor spends on recovering the credential in ciphertext and calculating a valid authentication tag. Note that this time cost includes the

time spent on the Initiator's transmitting parameters such as H. For privileged services, the GCM-CR time is part of the RFT, and therefore it is shorter than the RFT.

For each kind of requests, we logged the above metrics and had 100 trials. For each metric, the final measurement is the average of the 100 results. The experimental results are shown in Figure 11 and Figure 12.

None-Privileged Service. Figure 11 (a) gives the time in retrieving the client's own tweets. Since collecting a user's tweets does not necessarily require the permission from this user, we implemented by using the Twitter API without having the user's credential. For the client, the waiting time is from 8 seconds to 16 seconds. On the server side, with 90% probability, the API calls can be completed in less than 0.2 seconds. These results show that the time cost of the email conversations contributes most to the client's waiting time.

Figure 11 (b) shows the time for the service of searching by keywords. The Mailet server is configured to return 20 search results in a session, and RFT is less than 1 second. At the client side, the waiting time is in the range of 3 seconds to 8 seconds. It is worth noting that the search service can be completed without the user's credential. Consequently we implemented this service by using the Twitter API.

Figure 11 (c) presents the time of sending/receiving emails for the Mailet client and the server. The x-coordinate is set in log scales, and the y-coordinate is the Empirical Cumulative Distribution Function (ECDF). Both the Mailet client and server have to spend at least 1 second sending or receiving emails with 50% probability.

To compare GCM-CR with the normal Twitter service access, Figure 11 (d) gives RFT for posting a tweet/retweeting by Twitter API at the server side when this server is authorized by the user. The server uses about 0.5 seconds to retweet, less than the time of its posting a tweet, which is in the range of 0.8 to 1.2. For GCM-CR, the server uses about 0.6 seconds to retweet, and 0.8 seconds to post a tweet. This comparison demonstrates the high efficiency of GCM-CR.

These experimental results show that the email conversations contribute the most to the client's waiting time, while most Twitter API calls can be completed by the Mailet server in less than 1 second. These results demonstrate that Mailet can provide quick access to non-privileged services of Twitter for its clients.

Privileged Service. The measurements for privileged services are given in Figure 12.

Figure 12 (a) (b) shows how fast GCM-CR can be completed, and how quickly the Mailet servers handle the services of posting a tweet or retweeting. It shows the GCM-CR time is under 0.05 seconds, which has a speedup of 120 when being compared with the conventional 2PC computation. In addition, the figures show that the Mailet servers can handle a post request in 0.8 seconds and a retweet request in 0.6 seconds on average.

Figure 12 (c) and (d) show the RFT when the Checking-by-Sampling strategy is applied. For the post request, when a larger n is adopted (which means the Initiator starts more TLS connections), the RFT is increased. If $n = 4$, it takes the Mailet servers about 1.2 seconds to complete a post request. For the case $n = 8$, the RFT is about 2 seconds. For the system deployer, it can tune the parameter n to make a trade-off between usability and security. It is worth noting that if long-term detection is applied in Mailet deployment, a small n (for example $n = 2$) is suitable.

The experimental results demonstrate the high performance of the Mailet design, while having small CPU and memory usages.

Initiator		Category	Interceptor	
CPU	MEM		CPU	MEM
0.4%	0.8%	Email Client	0.5%	0.2%
4.88%	0.2%	Authorization	1.71%	0.1%
2.25%	0.1%	Post a Tweet, n=1	1.3%	0.2%
4%	0.1%	Post a Tweet, n=8	1.7%	0.2%
2.0%	0.1%	Retweet, n=1	1.8%	0.2%
3.15%	0.1%	Retweet, n=8	2.0%	0.2%

Table 3. CPU and Memory Consumption for Mailet

8 Related Works

8.1 Parrot Circumvention Systems.

The parrot circumvention systems disguise their communications with the circumvention system user by emulating the well-known non-blocked protocols. SkypeMorph [24] disguises the communication between a Tor client and bridge as Skype Voice over Internet Protocol (VoIP). SkypeMorph starts the video call between the bridge and client as camouflage, then it drops the genuine connection and transmit Tor's TCP traffic over non-Skype UDP. A packetizer module is used to emulate the Skype UDP traffic. StegoTorus [30] obfuscates

the Tor protocol. It can choose HTTP request as a cover protocol and embeds hiddentexts in the URL and cookie fields. For HTTP responses, StegoTorus utilizes the attached files such as PDF and Flash to carry the hiddentexts. CensorSpoofer [28] is designed to provide an unblocked web browsing service by IP spoofing. Consider the upstream traffic of the web browsing is lightweight, it uses low capacity channels like emails for transmission. For downstream traffic, it directly sends the traffic to the user but with the faked IP source address to fool the censor. As a consequence, the IP address of the proxy is concealed in both upstream and downstream traffic.

Imitation Flaws. [15] shows the parrot systems fail to achieve the unobservability. Firstly, SkypeMorph and StegoTorus fail to imitate side channels, which are created by the genuine systems for traffic control, user login, etc. Secondly, StegoTorus and CensorSpoofer fail to imitate reactions. StegoTorus returns different error messages when the censor sends HTTP requests to it, and CensorSpoofer fails to properly select the spoofed IP address/port, making the address/port behave differently in responding to a censor's probe. Thirdly, SkypeMorph and StegoTorus are incorrect in imitation. Both SkypeMorph and StegoTorus wrongly imitate Skype UDP packets for lack of SoM field. Also StegoTorus generates incorrect PDF lacking the *xref table*.

8.2 Circumvention Systems without Imitation Flaw

FreeWave [17] is designed to circumvent the Internet censorship by hijacking the Skype protocol. The web browsing traffic between a FreeWave client and server is modulated into acoustic signals, so that it can be carried via Skype acoustic channels. Since FreeWave directly uses the genuine Skype VoIP service, it is claimed to avoid flaws in camouflage.

SWEET [34] is proposed to provide the unblockable web browsing service via email channels. In the infrastructure, the user in the censored regime encapsulates its web browsing traffic in emails, and sends these emails to the SWEET server. At the server side, it extracts the traffic from the emails, and forward the traffic to the expected destination. After receiving the replies from websites, the SWEET server embeds the responses in the emails and sends them back to the user. Since the email service provider does not collude with the censor, the SWEET is claimed unobservable due to the censor's disability in recognizing the SWEET session.

CloudTransport[8] uses the cloud storage service to circumvent the Internet censorship. It proposes a passive-rendezvous protocol, which enables the CloudTransport client and bridge to communicate through oblivious cloud systems.

Inevitable Inconsistency. Recent work [14] reveals even perfect emulation can not guarantee the proxy unobservability and unblockability. The failure roots in the inevitable content and channel inconsistencies between genuine and proxy protocols.

Content Inconsistency. In FreeWave, a modulated acoustic signal rather than human speech is transmitted over VoIP. This content inconsistency is proved sufficient for a censor to identify FreeWave connection by traffic analysis. For SWEET, it utilizes the email channels to transmit the network layer traffic, making the connections distinguishable from the genuine email users. For instance, the SWEET client and server have to exchange 8 emails on average in order to complete one web browsing request. Since this burst of emails is rare for genuine email users, it gives the censor the opportunity to identify the SWEET connections. Besides, this deficiency limits the SWEET user to at most 10 to 15 web browsing requests, far from satisfying the user's needs.

Channel Inconsistency. Both SkypeMorph and FreeWave require reliable channels to transmit Tor TCP packets or synchronization frames. But VoIP usually adopt unreliable UDP transmission. This inconsistency enables a censor to stall SkypeMorph and FreeWave by packet dropping, while having negligible impacts on the genuine VoIP service. For SWEET, such channel inconsistency also exists. The email channel is delay tolerant, but the SWEET infrastructure can hardly handle the delayed email delivery, which makes the HTTP/HTTPS connections timeout or stalls the handshake protocol. That means the sensor could delay the email delivery to interrupt the SWEET connection without severely affecting normal email users.

8.3 Circumvention Systems with Consistency

By realizing the design pitfalls in censorship circumvention system, Facet [22] is proposed as an unobservable transport which enables the users in the censored regime to watch YouTube, Vine, and Vimeo videos in real-time. Facet circumvents the Internet censorship by streaming social videos over Skype video calls. Specially, the Facet server creates a pair of audio and video emulators for

the user, and after receiving the video request, the server will start a Skype video call with the user and streams the requested video through the emulators. At the user side, it can watch the video in Skype session. The Facet design is claimed to have no imitation flaws, and be consistent with the Skype video calls. Also, Facet includes video and audio morphing to mitigate the censor's traffic analysis attack. The experiment demonstrates that If the censor wants to block 90% Facet connections, it has to block 40% genuine Skype calls. Though Facet is more secure against blockage, it can only support social video sites and can not protect the user from an untrustworthy server.

At last, a system named Tweetymail exists with no purpose of circumventing Internet censorship. This system also uses the email channels to transmit the tweets, and users have to authorize the Tweetymail server by the Twitter authorization routine before using the system. This renders the users in censored regimes unable to utilize the Tweetymail service, since these users cannot access Twitter website servers to complete the authorization. In addition, Tweetymail cannot prevent an untrustworthy server, which poses potential threats to the users' account safety.

9 Limitations and Future Works

Mailet is aimed at enabling users in the censored regimes to access social media websites. While we confess that Mailet cannot support uncensored generic Internet access, this limitation does not undermine its usability. The design method is to satisfy most of users' circumvention needs by a small set of unblockable transports, and Mailet is one of these transports which is for accessing social websites. Together with other transports (such as Facet), Mailet can fulfill most of users' needs. The future work includes the extension of the implementation to support other social websites, and the privacy enhancement for the Mailet users.

10 Conclusion

This paper presents Mailet, an unobservable transport which provides instant social networking under censorship. In Mailet, the client sends the Mailet server emails, which contain the request for accessing the social website. On behalf of the client, the Mailet server communicates with the social website server, and emails any website response back to the client. To prevent an untrustworthy Mailet server from misusing or leaking the credential of the client, the credential is split and distributed to some of the Mailet servers, and Mailet exploits GCM based secure computation to privately combine the shares. The experimental results demonstrate the security and practicality of Mailet. The Mailet implementation for other social websites (such as Facebook) is taken as the future work.

Acknowledgements

We would like to thank our shepherd, Martin Schmiedecker, and the anonymous reviewers for their insightful comments and suggestions. This work was partially supported by the NSF under grant 1314637.

References

[1] Collateral freedom: A snapshot of chinese internet users circumventing censorship, https://openitp.org/news-events/collateral-freedom-a-snapshot-of-chinese-users-circumventing-censorship.html.

[2] Email statistics report 2014-2018, http://www.radicati.com/wp/wp-content/uploads/2014/01/email-statistics-report-2014-2018-executive-summary.pdf.

[3] Enron dataset, https://www.cs.cmu.edu/ ./enron/.

[4] Google approves an app that steals all your data, http://www.technologyreview.com/.

[5] Openssl, https://www.openssl.org/.

[6] L. V. Ahn, M. Blum, N. J. Hopper, and J. Langford. Captcha: Using hard ai problems for security. In *Proceedings of the 22Nd International Conference on Theory and Applications of Cryptographic Techniques*, 2003.

[7] Y. Aumann and Y. Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In *Theory of Cryptography*, pages 137–156. Springer, 2007.

[8] C. Brubaker, A. Houmansadr, and V. Shmatikov. Cloudtransport: Using cloud storage for censorship-resistant networking. In *Proceedings of PETS'14*, 2014.

[9] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. Openpgp message format. RFC 4880, 2007.

[10] T. Dierks. The transport layer security (tls) protocol version 1.2. 2008.

[11] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of USENIX Security'04*, 2004.

[12] Z. Durumeric, D. Adrian, A. Mirian, J. Kasten, E. Bursztein, N. Lidzborski, K. Thomas, V. Eranti, M. Bailey, and J. A. Halderman. Neither snow nor rain nor mitm...: An empirical analysis of email delivery security. In *Proceedings of IMC'15*, 2015.

[13] C. Evans, C. Palmer, and R. Sleevi. Public key pinning extension for http. RFC 7469, 2015.
[14] J. Geddes, M. Schuchard, and N. Hopper. Cover your acks: Pitfalls of covert channel censorship circumvention. In *Proceedings of CCS'13*, 2013.
[15] A. Houmansadr, C. Brubaker, and V. Shmatikov. The parrot is dead: Observing unobservable network communications. In *Proceedings of IEEE Symposium on Security and Privacy'13*, 2013.
[16] A. Houmansadr, G. T. Nguyen, M. Caesar, and N. Borisov. Cirripede: Circumvention infrastructure using router redirection with plausible deniability. In *Proceedings of CCS'11*, 2011.
[17] A. Houmansadr, T. Riedl, N. Borisov, and A. Singer. I Want my Voice to be Heard: IP over Voice-over-IP for Unobservable Censorship Circumvention. In *Proceedings of NDSS'13*, 2013.
[18] A. Houmansadr, E. L. Wong, and V. Shmatikov. No direction home: The true cost of routing around decoys. In *Proceedings of NDSS'14*, 2014.
[19] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *Proceedings of USENIX Security'11*, 2011.
[20] J. Karlin, D. Ellard, A. W. Jackson, C. E. Jones, G. Lauer, D. P. Mankins, and W. T. Strayer. Decoy routing: Toward unblockable internet communication. In *Proceedings of FOCI'11*, 2011.
[21] H. Kwak, C. Lee, H. Park, and S. Moon. What is twitter, a social network or a news media? In *Proceedings of WWW'10*, 2010.
[22] S. Li, M. Schliep, and N. Hopper. Facet: Streaming over videoconferencing for censorship circumvention. In *Proceedings of WPES'14*, 2014.
[23] D. Malkhi, N. Nisan, B. Pinkas, Y. Sella, et al. Fairplay-secure two-party computation system. In *Proceedings of USENIX Security'04*, 2004.
[24] H. M. Moghaddam, B. Li, M. Derakhshani, and I. Goldberg. Skypemorph: Protocol obfuscation for Tor bridges. In *Proceedings of CCS'12*, 2012.
[25] M. Schuchard, J. Geddes, C. Thompson, and N. Hopper. Routing around decoys. In *Proceedings of CCS'12*, 2012.
[26] Z. Tufekci. Networked politics from tahrir to taksim: Is there a social media-fueled protest style? In *Digital Media and Learning Central*. http://dmlcentral.net/blog/zeynep-tufekci/networked-politics-tahrir-taksim-there-social-media-fueled-protest-style, June 2013.
[27] S. Vieweg, A. L. Hughes, K. Starbird, and L. Palen. Microblogging during two natural hazards events: What twitter may contribute to situational awareness. In *Proceedings of CHI'10*, 2010.
[28] Q. Wang, X. Gong, G. T. K. Nguyen, A. Houmansadr, and N. Borisov. Censorspoofer: Asymmetric communication using IP spoofing for censorship-resistant web browsing. In *Proceedings of CCS'12*, 2012.
[29] B. Warf. *Global geographies of the Internet*, volume 1. Springer Science & Business Media, 2012.
[30] Z. Weinberg, J. Wang, V. Yegneswaran, L. Briesemeister, S. Cheung, F. Wang, and D. Boneh. StegoTorus: A camouflage proxy for the Tor anonymity system. In *Proceedings of CCS'12*, 2012.
[31] E. Wustrow, C. M. Swanson, and J. A. Halderman. Tapdance: End-to-middle anticensorship without flow blocking. In *Proceedings of USENIX Security'14)*, 2014.
[32] E. Wustrow, S. Wolchok, I. Goldberg, and J. A. Halderman. Telex: Anticensorship in the network infrastructure. In *Proceedings of USENIX Security'11*, 2011.
[33] A. C. Yao. Protocols for secure computations. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 160–164. IEEE, 1982.
[34] W. Zhou, A. Houmansadr, M. Caesar, and N. Borisov. Sweet: Serving the web by exploiting email tunnels. In *Proceedings of HotPETs'13*, 2013.

Appendix

A. Transport Layer Security Protocol

The Transport Layer Security (TLS) protocol [10] provides communication security over the Internet. Instead of submitting its credential in a transparent protocol, the client initializes a TLS handshake with the social website server, and communicates by encrypted messages. This prevents attackers from sniffing, modifying, or injecting the traffic.

Protocol Overview. The following summarizes a typical handshake procedure for the TLS protocol:

- *Client Hello.* This is the first message sent by the client, which includes a client-generated random number, the cipher suites listing the cryptographic options supported by the client, and the protocol version number.
- *Server Hello.* The server's response messsage to the *Client Hello.* This message contains a server generated random number, and a single cipher suite selected by the server from the client cipher suites.
- *Server Certificate.* The Server sends a certificate message which is used by the agreed key exchange method for authentication.
- *Server Key Exchange Message.* This message is only sent by the server when the server certificate message does not provide sufficient data for the client to finish a handshake. For key exchange methods such as DHE_RSA, this message is needed.
- *Client Key Exchange Message.* The client sends this message to set the premaster secret, either by a RSA-encrypted secret or by the transmission of Diffie-Hellman parameters which allows the two sides to agree upon a common premaster secret.
- *Client Change Cipher Spec.* Just after the client key exchange message, the client sends the *change ci-*

pher spec message to the server, notifying that it will use the newly negotiated cryptographic parameters immediately. Following this message, a finished message is sent by the client under the new cryptographic parameters.

– *Server Change Cipher Spec.* In response, the server sends its *change cipher spec* message, and changes the cryptographic state into the new. Also, a finished message is sent by the server under the new cipher specification.

After completing the TLS handshake procedure, the server and the client can transmit the application data (including the client credential) to each other under the the negotiated cryptographic parameters.

TLS Record Format. The general format for all TLS messages is illustrated in Figure 3. The ProtocolVersion field (2 bytes) specifies the TLS version, and ContentType field (1 byte) indicates the type of the carried traffic. The possible types include handshake(22), application data (23), and change cipher spec (20), etc. The Length field (2 bytes) is the length of the fragment in byte, in which the application data resides. An example of the TLS format for application data under stream cipher encryption is shown in Figure 4. The content field contains the encrypted application data, and the MAC field is the Message Authentication Code (MAC) for the application data, a sequence number, ProtocolVersion, ContentType, and length of the application data. The sequence number is included in the MAC calculation, so that missing or duplicated messages are detectable. It is also worth noting that the MAC is computed before the encryption of the application data.

B. Comparisons with Similar Systems

There exist two proxy systems SWEET and Tweetymail also using the email channel for transmission. In this section, we compare Mailet with these two systems.

Accessibility.
Mailet: the application-level contents are directly exchanged between the user and the Mailet server, so that the user does not need to install any client-side software. This makes Mailet easy to be accessed.
SWEET: it requires the user to install the client-side program to tunnel through the email channels.
Tweetymail: Tweetymail is not available in censored regimes, because the users cannot finish the authorization, which requires them to access Twitter.com.

Usability.
Mailet: the high efficiency of GCM-CR enables the Mailet servers to complete a service request in less than 1 second. Besides, the user and the Mailet servers exchange at most 2 emails to complete the service. As a consequence, the total waiting time for the users is between 3 and 5 seconds.
SWEET: it has to exchange 8 emails on average to complete a service request. That means the total waiting time in SWEET is about 16 seconds on average.

Unblockability.
Mailet: since Mailet uses the email channel to transmit text contents, it can defend against the blockage exploiting the channel and content inconsistency.
SWEET: it uses the email channel to transmit the web browsing traffic (network layer packets), which makes it vulnerable to the censor's traffic analysis and proactive interference.

Security against Untrustworthy Servers.
Mailet: it adopts the decentralized credential to protect the user's account. A Mailet server alone cannot recover the user's credential. Besides, the CbS mechanism in Mailet design can mitigate the malicious attacks.
SWEET: it can defend against an untrustworthy server.
Tweetymail: an untrustworthy Tweetymail server may leak the authorized access tokens of the users, or directly use these tokens to attack the users' account.