Privacy in Centralized Systems

by

Theodore Ian Martiny

B.S., Virginia Commonwealth University, 2012

M.S., University of Pittsburgh, 2015

M.S., University of Colorado, 2017

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

2022

Committee Members:

Eric Wustrow, Chair

Dirk Grunwald

Sangtae Ha

John Black

Eric Keller

Martiny, Theodore Ian (Ph.D., Computer Science)

Privacy in Centralized Systems

Thesis directed by Assistant Professor Eric Wustrow

The vast majority of online services are run using a centralized infrastructure. The centralized nature of these services allow the provider to have absolute control over the content as well as any profit generated by that content. Centralized services often have servers distributed across the world for user reliability, though they function as centralized systems: their functionality is identical across their network and the data they collect is available to the service as a whole, not only the server a user interacts with. Users of these services, and the data they generate, are completely at the whim of the provider; companies often offer vague promises of security and privacy of the data collected from their users which the users cannot verify themselves. Moving these services to a decentralized system (where each server acts independently of others) could address these issues but decentralized systems often face severe scalability issues as well as having cumbersome requirements on users such as needing specialized software to access the service.

This dissertation demonstrates that user privacy can be inherently built into centralized systems using cryptographic protocols. Centralized systems can offer their services with minimal user information allowing services **actually** geared towards privacy to have it be a core functionality of their service. A service that doesn't have access to users data cannot abuse or leak it.

Proof of Censorship utilizes Private Information Retrieval to allow content providers (such as Twitter) to be cryptographically auditable over whether content has been modified or removed. In 2018 Signal (a secure end-to-end messenger) introduced Sealed Sender in an attempt to hide the sender of encrypted messages. **Improving Signal's Sealed Sender** strengths Sealed Sender by guaranteeing that privacy cryptographically using blind RSA signatures. **Mind the IP Gap** measures how countries utilize their centralized censorship apparatus to restrict content to users through DNS manipulation.

Acknowledgements

I am grateful to my committee and most importantly my advisor Eric Wustrow for all their support throughout my education. I'm deeply thankful to my parents for supporting me throughout my whole educational journey, and gently reminding me exactly how many years I've been in school. I couldn't have done this without the support of Kim who is always there.

Contents

Chapter

Tables

Table

Figures

Figure

Chapter 1

Introduction

A **centralized system** is one in which the core functionality of the system is dictated by a single body that single-handedly determines the use and behavior of the system. Any privacy guarantees are at the sole discretion of the system. Common examples of centralized systems of relevance to this dissertation are Twitter, Signal (a secure end-to-end encrypted messenger), and centralized censorship systems. In each of these cases the centralized systems have multiple servers spread across their areas of operation, around the world for global services such as Twitter and Signal and spread throughout a censored region in the case of censorship systems. However these servers function as a whole, often running the exact same code, and sharing the data they collect with the centralized system of use.

Users of centralized systems have no inherent guarantee of **privacy** for their data. How their data is collected and used is at the sole discretion of the service. Services often offer vague claims as to their privacy guarantees for users: social media sites collect a mass amount of information on its users, extending past the content provided to the site; Facebook collects information on which people users interact with, including who you communicate with and which content you interact most often with [3]. Similarly Twitter collects metadata information on its users including browser information and operating system [9].

With the current state of the internet functioning primarily under centralized systems that function on the data that users generate or inherently provide **can privacy exist in centralized systems?**. This dissertation answers in the affirmative: privacy can be cultivated in these

centralized systems by utilizing cryptographic protocols.

I present here multiple avenues of research providing mechanisms for centralized systems to offer services with privacy. These services are built upon cryptographic protocols which limit information shared between users and the services and have the smallest possible surface area allowing these centralized systems to offer their services while fully enabling user privacy.

In order to develop these tools which guarantee user privacy there must be a basis of what information is necessary for a service to function as well as how potential bad acting Centralized Systems abuse the service they offer to restrict access to the content they offer.

The concept of user privacy is inherently linked to the concept of service provider censorship. A service with no user privacy can censor a users's access to content through that lack of privacy. By guaranteeing user privacy we restrict the possibility of a service censoring content to users.

This dissertation showcases three topics of enhancing user privacy in centralized systems examining three different avenues of censorship. First we examine content provider censorship, where a service that provides access to content censors that access. I present a solution to this problem applicable for content providers that wish to demonstrate their commitment to user privacy. Second I examine metadata based privacy, where a service provider might censor access to content not based on the content itself but based on the metadata around that content such as who a message is sent from. I submit a cryptographic method to achieve privacy around the metadata associated with end-to-end encrypted messages, strengthening Sealed Sender, a technology geared towards privacy that is vulnerable to a simple Statistical Disclosure Attack. Finally I showcase a measurement study of DNS censorship over IPv6 to learn how countries censor access to content differently over IPv6 compared to IPv4 motivating avenues of censorship circumvention.

1.1 Content Provider Censorship

In 2016, Twitter received over 11,000 removal requests from government and police agencies worldwide, and complied with 33% of them [159]. In one example from August 2016, Twitter complied with a Brazilian government request to remove a tweet comparing then-mayoral candidate

Rafael Greca to a fictional villain from the 1992 film Batman Returns [105]. Between May 6 and May 19, 2021 Facebook removed nearly 500 pro-Palestine posts without any public explanation [132].

Services can be compelled (politically or financially) to censor content for some subset of users which makes the censorship less overt: the content is available for some users but not others. Can centralized systems be designed to ensure they are transparent and auditable in terms of the content they remove?

Chapter 2: **Proof of Censorship: Enabling centralized censorship-resistant content providers** answers this question affirmatively. Content providers can be inherently auditable in order to be held accountable to the content they remove access to.

Using Private Information Retrieval (PIR) a content provider is unaware of the content a user is requesting thereby inhibiting its ability to censor access to a subset of content. The provider must censor access to the service as a whole or not at all.

Proof of Censorship extends this functionality by guaranteeing the content has not been modified. This is done by providing a cryptographic proof the content has not been modified since upload time. This further enables services to purposely remove content against its policies, users receive proof that the content has been removed (or modified) that the service can publicly announce.

1.2 Metadata Privacy

In addition to the content that users provide to services they also inherently provide metadata associated with their content. Metadata is generally only loosely based on the content it is related to. In the context of secure end-to-end-encrypted messengers, such as Signal, the content of messages is completely hidden from Signal and any other man in the middle. However, there is metadata associated with encrypted messages, e.g., the sender and receiver of the message, the time the message was sent, and in a more granular fashion, the size of the message. Even centralized services that aim to protect content using end-to-end encryption may receive sensitive metadata. For example, encrypted messaging apps' servers might learn who is communicating, even if the

content is protected.

While leaking metadata may appear reasonable when compared to revealing the contents of the messages, observing metadata can have serious consequences. Consider that Alice may be a whistleblower communicating with a journalist [112] or a survivor of domestic abuse seeking confidential support [75]. In these cases, merely knowing **to whom** Alice is communicating combined with other contextual information is often enough to infer conversation content without reading the messages themselves. Former NSA and CIA director Michael Hayden succinctly illustrated this importance of metadata when he said the US government "kill[s] people based on metadata" [86].

In 2018 Signal, a secure end-to-end encrypted messenger, introduced Sealed Sender[106], a protocol that provides a step in the correct direction towards hiding metadata.

In Chapter 3: **Improving Signal's Sealed Sender** I demonstrate that the Sealed Sender protocol is vulnerable to a Statistical Disclosure Attack (SDA), which would allow a malicious or compelled Signal (or other secure end-to-end encrypted messengers that implement Sealed Sender) to link these One-Way Anonymous messages due in part to delivery receipts: a feature of most online messengers which immediately alert a user when their message has been delivered.

Further, I develop a more private protocol by formalizing Sealed Sender **conversations**, as opposed to individual messages, and decoupling a user's long term identity (such as Alice) from the origin of messages, thereby creating a cryptographically private One-Way Anonymous conversation. This framework is extended to Two-Way Anonymous messages where the messaging service need not know either the sender nor the receiver of a message.

1.3 Measurement of censorship in existing systems

Internet censorship is a global problem that affects over half the world's population. Censors rely on sophisticated network middleboxes to inspect and block traffic. A core component of Internet censorship is DNS blocking, and prior work has extensively studied how DNS censorship occurs, both for specific countries [126, 79] and globally [90, 46, 134, 147].

Nation-state level censorship systems function as centralized systems. Similar to global ser-

vices that have many servers located across the world, a nation-state censorship system may have many middleboxes distributed across their region to perform censorship, but the policies they follow on the whole are centralized and limit the autonomy of the users in those countries. Chapter 4: **Mind the IP Gap: Measuring the impact of IPv6 on DNS censorship** works to study centralized censorship systems globally.

IPv6 is becoming more widely deployed, with nearly 35% of current Internet traffic being served over native IPv6 connections [12]. It may appear that censors fall into a bifurcation of either censoring over IPv6 or not, but in practice we see that is not the case, we find a large range of how well censors block DNS requests over IPv6 compared to IPv4.

By studying censorship in IPv6 we create opportunities for censorship tool development to exploit the censorship techniques used by individual countries, allowing citizens access to content they would not otherwise be able to reach.

In Chapter 4: **Mind the IP Gap: Measuring the impact of IPv6 on DNS censorship** I present the results of our measurement of IPv6 DNS censorship compared to IPv4. We find that censors show a clear bias towards censoring in IPv4, and towards censoring native records (`A` records in IPv4 and `AAAA` records in IPv6). We examine statistically significant differences we discover at the country level by resolver and domain. We find that Thailand, Myanmar, Bangladesh, Pakistan, and Iran all present significant and consistent discrepancies across all resolvers and domains suggesting avenues for successful censorship circumvention opportunities in those countries.

1.4 Thesis Structure

In this thesis I provide three projects geared towards enhancing privacy in centralized systems:

(1) **Proof of Censorship** — this work showcases content provider censorship and how a service dedicated to being transparent about content it modifies or removes can implement a service which provides cryptographic promises that content has not been censored.

(2) **Improving Signal's Sealed Sender** — Signal's Sealed Sender provided a first step to-

wards user privacy but ultimately is vulnerable to a simple Statistical Disclosure Attack allowing Signal (if compelled or malicious) to determine who a user is messaging. This work provides a cryptographically private mechanism to ensure any centralized messaging service can offer its service without learning metadata information about which users are messaging each other.

(3) **Mind the IP Gap** — this work measures how countries' centralized censorship systems block access to the internet through DNS resolution over the IPv4 and IPv6 internet allowing future work to implement systems to allow censored citizens to circumvent countries DNS censorship techniques.

Chapter 2

Proof of Censorship: Enabling centralized censorship-resistant content providers

2.1 Introduction

Online censorship is done in many ways. In addition to blocking access to websites, censors also use legal means to remove information from content providers directly, such as through laws like the DMCA [39] in the United States. In the second half of 2016, Twitter received over 5,000 removal requests from government and police agencies worldwide, and complied with 19% of them [159]. In one example from August 2016, Twitter complied with a Brazilian government request to remove a tweet comparing then-mayoral candidate Rafael Greca to a fictional villain from the 1992 film Batman Returns [105].

This style of censorship can be significantly less apparent than overt website blocking. Because the service remains accessible, users are unaware of the content that may be secretly withheld from them. While some content providers (including Twitter) publish transparency reports containing overall statistics or examples of removed content [159, 61, 69], these reports are difficult to confirm or refute: How can users know if the transparency reports themselves have not been similarly censored?

While decentralized or distributed tools provide a way to make content robust against censorship, these techniques do not fully solve the problem. In a distributed censorship-resistant scheme, users must download and use specialized software which may itself be blocked by a censor. Due to this requirement, these tools are only utilized by well-informed users, while the majority of people

rely exclusively on centralized content providers. Thus, there are clear advantages to providing censorship resistance in a centralized content storage context.

One idea proposed to incentivize censorship-resistant content providers to not delete data is to use **proof of retrievability** (PoR) to enable content providers to prove that they still retain a copy of a file. In PoR, providers respond to requests or challenges for specific subsets of a file they have agreed to store. A provider responds with the subset, proving (over many challenges) that they likely still store the entire contents of the file. With enough successful PoR challenge responses, a client can reconstruct the file, meaning that a provider that gives valid PoR responses for a file necessarily provides access to it. While this is a useful starting point, there are two major problems with this approach.

First, the content provider can always choose to not respond to a PoR request. Note that this is different from providing an invalid response to a request. By simply leaving a connection open and providing no response, the service never provides any incriminating evidence that they are not providing access to the file. While other clients can see this suspicious behavior, the provider could choose to respond correctly to some subset of users, seeding confusion and distrust over claims of censorship.

Second, the content provider can cease this behavior at any time, with no evidence that they ever censored a file. For example, a provider could wait until a certain number of users realized a file was removed or journalists started to ask questions before they reverted the decision to censor a file. Such a provider could pretend they never censored a file, and those that observed that they did would have no transferable proof. This allows a provider to censor files at will, and restore them only when convenient.

In this paper, we describe a way that a content provider can be held accountable for the files that they censor. We do this via a **proof of censorship** that a content provider unavoidably produces should they censor a file. Furthermore, these proofs are transferable, meaning that once one has been produced, others can verify the cryptographic proof and see that it attests to a

censored file. Even if the content provider restores the original file, this proof still maintains the evidence of previous censorship.

To construct our proof of censorship, we use private information retrieval (PIR) which hides what information a client is requesting from a server. At a high level, servers commit to storing (encrypted) files by signing hashes of those files on upload. To download a file, a client performs a PIR query, which the server responds to, signing both the PIR query and the server's response. Because the server does not know what file a client is requesting, it cannot selectively answer queries. To censor a file, the provider must return garbage data. Upon decryption of the PIR response, the client can confirm the file has been censored, and can reveal its PIR queries and the signed response to the world. Anyone with this proof (and the content provider's long-term public key) can verify that the content provider has removed the file.

Our proofs of censorship are compact and relatively efficient to verify: in the event of censorship, for a reasonable size database, the proof is on the order of a few megabytes, and takes approximately 50 milliseconds to verify on a laptop computer.

2.2 Background

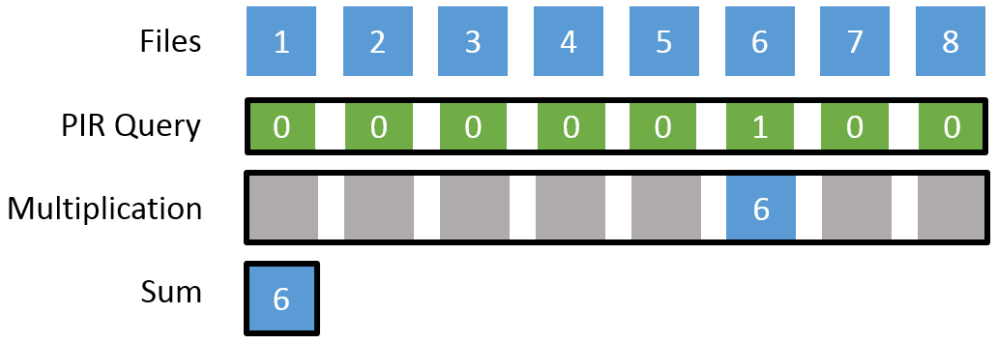

Figure 2.1: **Vector-Matrix PIR** — Private Information Retrieval involves a client making a query against a database. In Vector-Matrix PIR the client encrypts a vector of query elements and the server then multiplies the query elements against the database blocks homomorphically, using homomorphic addition to produce a single sum. This results in an encrypted value corresponding to the element where the client encrypted 1 (instead of 0), which is sent back to the client for decryption.

To describe our proof of censorship, we first provide background on private information

retrieval (PIR) [92]. PIR allows a client to request data from a server without revealing to the server what data is being requested. A naive way to accomplish this is for the client to download the entire database from the server, then select its data locally. This method is bandwidth inefficient, however, and does not scale to large datasets.

There are two settings for PIR. In **information-theoretic** PIR (ITPIR), a set of N servers share the database, with clients making requests to all of them and combining their responses to obtain the requested files. If at least one of the servers is not colluding with the others, the client's request remains private. In **computational** PIR (CPIR), a single server stores the database, and clients use homomorphic-encrypted requests to retrieve data. As long as the computational hardness assumptions on the underlying cryptography hold, the server cannot determine what information the client requested. As we are focused on a single centralized content provider, in this paper, we only consider and describe CPIR.

Many CPIR schemes fall into the "matrix-vector" model [73], which is illustrated in Figure 2.1. In this model, CPIR uses a (IND-CPA-Secure, additive) homomorphic encryption that allows operations done on ciphertext to correspond to operations done on the corresponding plaintext. As an example, consider an additively homomorphic encryption function $E()$ with corresponding decryption function $D()$ that allows addition of ciphertext and multiplication by a constant; i.e,:

$$D(E(a) + E(b)) = a + b$$

and

$$D(E(a) \cdot c) = a \cdot c.$$

To perform a PIR query, a client constructs a vector of encrypted 0s and 1s. For example, if the server has n blocks, and the client wishes to retrieve block x from the server where $0 \leq x < n$, the client sends a query vector Q to the server, comprised of $q_{i \neq x} = E(0)$, and $q_x = E(1)$. Note that the IND-CPA security of the encryption scheme implies that the server cannot determine which q_i is the encryption of 1.

With Q, the server (homomorphically) multiplies each q_i with its corresponding data block d_i in the database. The server then takes the homomorphic sum over the result. When $i \neq x$, the multiplication by $E(0)$ will ensure the corresponding block does not contribute to the later-decrypted sum. The response is sent back to and decrypted by the client, resulting in $D(\sum_i q_i \cdot d_i) = D(E(1) \cdot d_x) = 1 \cdot d_x$.

The PIR library that we use (XPIR [14]) provides two optimizations: recursion and aggregation. Recursion allows clients to send fewer query elements to the server, by breaking the database into a d-dimensional cube, and having the query select the row/column vectors. For example with a database of 100 records, it can be broken into a 10×10 table of elements. The client sends 10 queries, which the server copies and applies to each row effectively selecting a singular column. Then, a separate 10 queries can be used to select a single element from that column. Thus the client sends a total of 20 query elements, as opposed to 100 (with no recursion).

Aggregation allows a client and server to pack multiple plaintext files into a single element. For example, if the ciphertext allows for an absorption of 768 bytes, but files are only 128 bytes, the database could utilize aggregation, fitting 6 files into each block. Clients then select the block of 6 files that contains their requested file, and locally discard the other 5. With aggregation, the client sends fewer request elements to the server, resulting in smaller queries.

2.3 Threat Model

We consider a setting consisting of a single centralized content provider with multiple clients who upload and download content. Clients upload files to the provider, and distribute tickets that enable others to download the file. The information in a ticket can be summarized in a URL to be provided to others. This allows clients to easily share tickets in the same way they would distribute online content to others. In this model, we wish to detect a censoring content provider while preventing malicious clients from making false accusations. We achieve two properties:

Non-repudiation of Targeted Censorship A provider cannot selectively censor a file while re-

sponding to queries from honest clients without producing a transferable proof of their misdeed.

Unforgeability Against a non-censoring content provider, an attacker cannot forge a proof that a file was censored.

We note that our threat model does not prohibit a provider that chooses to delete or remove access to all of its files: a provider can always shut down entirely, or refuse to sign statements with its private key.

2.4 System Design

In this section, we describe the details of our proof of censorship. A proof of censorship has two parts: a commitment from the server that it will store and distribute a file, and second, a later proof that it failed to uphold that commitment. The first part is obtained during file upload, where the server returns a signed and timestamped **ticket** that efficiently represents the commitment to store the file, while the second part is obtained when a client downloads the file the server is attempting to censor. We assume the server has a widely published long-term public signing key.

We begin our description assuming that a file fits in a single block that can be requested in a single PIR query to the server. In Section 2.4.2, we describe how to efficiently extend this idea to provide proofs of censorship for files that span multiple blocks.

2.4.1 Proof of Censorship Construction

Ticket Construction On upload, a client will upload an encrypted block that represents the file. The server chooses an index for the block to be stored, and provides a signature over the data and the index. For block data B_i stored at index i, the server hashes the block data to obtain $H(B_i)$. The server then signs the tuple containing a timestamp t, the index i, and the block hash $H(B_i)$ using its long-term private key. This signature, along with t, i, and $H(B_i)$ are sent to the client in the form of a **ticket**.

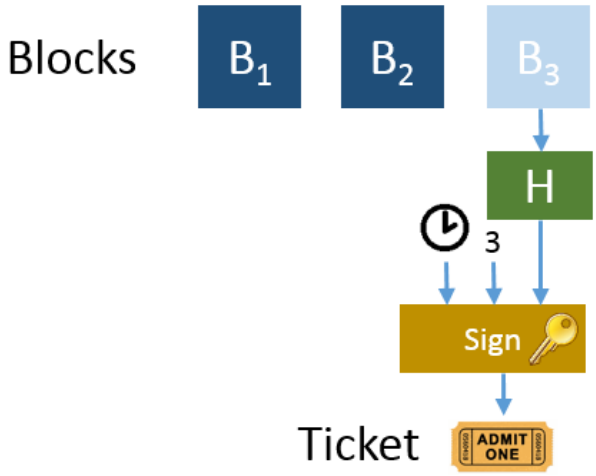

Figure 2.2: **Ticket Creation** — When a file is uploaded, the server produces a signed ticket that commits the server to storing the block of data. The server signs a timestamp, the index where the data is stored, and a hash of the data, and returns this to the client for local storage.

This ticket can be encoded into a URL that can be distributed easily to other users in the form of:

$$\texttt{https://<service>/\#}H(B_i)/i/t/sig$$

where *sig* is the server's signature of the ticket. This gives the client all of the information it needs to verify as well as a simple way to distribute the ticket. Additionally the sensitive information is provided to the client in such a manner that it is not passed to the server, as values after a # symbol are only handled locally in browsers.

The client then verifies that the ticket is well formed and valid. First, the client hashes the data it has uploaded to obtain $H(B)$. Then, it ensures that the timestamp t it received from the server is a recent timestamp[1] . Finally, the client checks the signature of the ticket, using t, i, and the client-computed $H(B)$. If the signature is valid, the client stores the ticket for this block. If any checks fail, the client can attempt to re-upload their file until they receive a well-formed ticket. Figure 2.2 illustrates how tickets are generated during file upload.

File Download To download a file, a client creates a PIR query that will result in a proof

[1] The client must ensure t is not located days in the future to avoid a server producing invalid proofs later.

of censorship if it does not receive the file requested. During a normal PIR request, a client encrypts a vector of 0 and 1 elements using using random coins for the encryption. In our scheme, these coins are the output of a pseudorandom generator with a **seed** (Q_{seed}) randomly selected by the client. This allows us to later reproduce the same query with only the seed and the index i. With this we create a compressed representation of the query which consists of the queried index (in the clear), and the seed.

The client creates the request Q and sends it to the server, along with the public key (as well as any cryptographic parameters needed to allow the server to perform homomorphic operations on the query[2]). The server then performs the query over its database using PIR, producing a short reply R that contains an encrypted representation of B_i. The server then hashes Q (including the public key) to obtain $H(Q)$, and produces a signature over a timestamp t, the query hash $H(Q)$, and the reply R. The server sends this signature, along with t and R to the client.

The client then extracts B_i from R using standard PIR extraction (i.e. decrypting R with its Q_{seed}-derived private key). The client then checks if this is the expected file by comparing it to the hash in the corresponding ticket for this file. If the client detects that the block has been censored, it now has a proof of censorship in the form of the server's response. The client publishes the original ticket, the server's signed PIR reply, and its Q_{seed} as a proof of censorship.

During this process, the server does not know what file is being requested. Therefore, to censor a file, the server must either not respond to any queries, or risk revealing that it has removed a file from its database.

Verifying Proofs of Censorship Given the server's public key, a signed ticket, a signed reply, and a query seed, a third party can verify whether the proof is valid evidence of censorship. To verify a proof of censorship, a verifier must perform several steps, detailed below.

(1) **Check Timestamps** The verifier checks that the ticket's timestamp (t_t) is before the reply's timestamp (t_r), ensuring that the query took place after the server committed to storing the file.

[2] E.g. in Paillier, this involves the public modulus generated by the client.

(2) **Check Ticket Signature** Given the ticket's timestamp t_t, the requested index i, and the hash of the file $H(B_i)$, the verifier validates the ticket's signature with the server's public key.

(3) **Regenerate PIR Query** Verifier uses Q_{seed} and the index i to deterministically derive the PIR query Q, and computes the hash over the query $H(Q)$.

(4) **Check Reply Signature** Given the reply's timestamp t_r, the computed hash $H(Q)$, and the server's reply, the verifier checks the reply's signature using the server's public key.

(5) **Extract Reply** Again using the key derived from Q_{seed}, the client extracts the PIR reply R to obtain B_i.

(6) **Check Data Hash** The verifier finally compares the hash of the extracted data B_i to the hash committed to in the ticket. If the hashes are equal, the server did **not** censor the file, and returned the expected result. However, if the hashes do not match, this is a valid proof of censorship.

2.4.2 Handling Multiple Blocks

So far, we have described how a client can efficiently receive a proof of censorship for a single uploaded block, but this does not address what happens if a file is larger than a single block. While a client could easily split up a file into multiple blocks, and receive a ticket for each block, storing each of these signed tickets may be expensive.

Instead, we can extend our design to allow for multiple blocks to receive a single ticket, while still allowing a proof to catch if an individual block is censored. To do this, our ticket will contain a Merkle tree root [117] in place of the single block's hash. The leaves of the Merkle tree will be the hashes of each block's data, and the ticket will consist of the list of hashes, and the final signature over the Merkle root, timestamp, and block index range.

To verify a proof, the verifier reconstructs the Merkle root from the suspected censored block and its Merkle tree-adjacent neighbors, and uses the root to verify the ticket. Thus, a verifier does

not have to know all of the hashes of the tree, but rather only those that allow it to reconstruct the Merkle root ($log(N)$ hashes for an N-block file). This allows proofs to stay relatively compact even as the file size grows. After validation of the ticket, the verifier can be assured that the hash of the suspected block ($H(B_i)$) they have been given is the correct hash for the given block B_i, and can perform the remaining checks described previously.

2.4.3 Security Argument

2.4.3.1 Non-repudiation

Assuming an honest client, that H is a collision resistant hash function, and the query privacy of the PIR scheme, we can show that the non-repudiation property holds (that a selectively censoring server will produce a proof of censorship).

The server can attempt to censor a file in two ways: by not responding to queries for the file, or by creating a malformed response. Malformed responses themselves can be made by either changing the data as it is stored/responded to, or by creating invalid signatures over the reply depending on the query.

A server that chooses to change its response based on the query (either not respond or produce an invalid signature) is prevented by the privacy of the PIR scheme. The server cannot know what file is being requested, and thus cannot selectively perform different behaviors based on the file requested. Honest clients will detect and reject malformed responses, effectively censoring the file they requested without the server knowing which file it is censoring.

The server can modify a file block directly, but because they have committed to the hash of the block's data in the ticket, and because of the collision resistance of the hash, the server is unable to change this data without producing a proof when that data is next requested.

2.4.3.2 Unforgeability

We show that the scheme is unforgeable assuming the servers signature scheme is unforgeable, H a random oracle, and the PIR scheme is the "vector-matrix type" [73] with an underling

encryption scheme that is "homomorphic quasi-committing."

Homomorphic Quasi-committing A homomorphic encryption scheme gen, enc, dec is homomorphic quasi-committing if an attacker cannot produce a ciphertext ct and values $m_1, m_2, c_1, c_2, pk, sk, r_{keygen}, r_1, r_2$, such that

$$pk, sk \leftarrow gen(\lambda; r_{keygen})$$

$$ct = c_1 \cdot enc(pk, m_1; r_1) + c_2 \cdot enc(pk, m_2; r_2)$$

where

$$dec(sk, ct) \neq c_1 \cdot m_1 + c_2 \cdot m_2$$

This property is closely related to correctness, which states that for all choices of random coins and query indexes, an incorrect result is returned with negligible probability. However, crucially, correctness is insufficient here. First, it does not ensure that a given query could be correct for two different choices of coins (and therefore private keys) and query indexes. I.e, it doesn't explicitly prevent a forgery where an attacker opens a legitimate query to a different result. Second, correctness effectively says there is a negligibly set of bad random coins which cause the scheme to fail. It provides no protections against an attacker choosing from that subset intentionally.

Homomorphic quasi-committing is a strong property to require of an encryption scheme. For the rest of the paper, we assume that the underlying encryption scheme used in the PIR is homomorphic quasi-committing in the random oracle model where r_{keygen}, r_1, r_2 are the output of a random oracle h evaluated on a nonce $nonce$ and the query index i.

Consider an attacker interacting with an honest content provider who produces a ticket and a proof of forgery for that ticket. Either the ticket or the PIR transcript must be forged. Forgeries in the ticket are prevented by the collision resistance of h and the security of the signature scheme.

Security Argument

We now consider the case of forgeries in the PIR transcript. By the collision resistance of the hash function and security of the signature scheme, a attacker cannot substitute either the query or response. As such the PIR transcript itself must represent the actual query to and response from

the provider. The only freedom an attacker has here is the choice of openings for the transcript (i.e. the random coins used for encryption and the public and private keys.) and she must reveal those as part of the proof. Thus the attacker reveals

$$pk, sk, r_{keygen}, r_1, \ldots, r_n, ct$$

for a query on block i such that

$$ct = B_i \cdot enc(pk, 1; r_i) + \sum_{j=0, j\neq i}^{N} B_j \cdot enc(pk, 0; r_j)$$

and

$$dec(sk, ct) \neq B_i$$

It follows that $0, 1, \sum B_j, B_i, pk, sk, r_{keygen}, \sum r_j, r_i$ violates the 'homomorphic quasi-committing" property of the encryption scheme for $B_* = \sum_{j=0, j\neq i}^{N} B_j \cdot enc(pk, 0; r_j)$ By assumption, this is not possible.

2.5 Implementation and Evaluation

We implemented our proof of censorship construction on top of XPIR, a fast PIR tool written in C++ [14]. Our implementation consisted of several hundred lines of modifications to XPIR, as well as an approximately 500-line application using the resulting libXPIR library. We used OpenSSL to perform the necessary signatures and hashes in our protocol.

The parameters we selected for testing were motivated by a simple text-only Twitter-like application, where messages are short (256 bytes, enough to include a short post and its metadata). We tested against a database of 1 million simulated messages in order to evaluate the time and size of different parts of the system. We used a quad-core Intel Core i5 CPU (3.30 GHz) with 32 GBytes of RAM to evaluate our prototype.

Table 2.1 shows the size and time to generate and validate our ticket, PIR query, and PIR reply (containing the proof). For our XPIR parameters, we choose to use XPIR's Learning with Errors implementation with 248 bits of security, using a polynomial of degree 2048 and a modulus of

Table 2.1: **Ticket and Proof Size**— We implemented a prototype of our proof-of-censorship system, simulating a database of 1 million 256-byte messages. We are able to keep a constant size of our proof at 2 MB. This allows clients that receive a proof of censorship to be able to easily distribute it.

	Size	Generation time (std-dev)	Validation time (std-dev)
Ticket	120 bytes	334 µs (0.53)	381 µs (4.61)
Query	3.8 Mbytes	28 ms (0.44)	n/a
Reply/Proof	2.0 Mbytes	2.8 s (0.19)	52 ms (0.54)

60 bits, with recursion level (d) 3, and aggregation (α) 16. All of the client operations are relatively fast: ticket validation (395 µs), query generation (27 ms), plaintext extraction (3.5 ms), and proof validation (45 ms) suggest that clients could be implemented in smartphones or even Javascript web pages without significant performance issues [10].

2.5.1 Scalability

We performed measurements of our proof of censorship and underlying PIR library to determine how the system scales. We measured the performance of downloading a single file with various database sizes, ranging from 5,000 to 1 million files of both small (256 byte) and large (1 MByte) size. For each database size, we used XPIR's built-in optimizer to determine the best parameters. XPIR chooses between two encryption options (Paillier, and the lattice-based learning with errors (LWE)), varying parameters of each to be optimal given a bandwidth estimate, file size, and number of files. We encouraged XPIR to optimize for small query and response sizes by providing a low bandwidth. Although Paillier produced the smallest query/responses, its server-side processing time was many orders of magnitude slower than LWE, effectively making it impractical. As a compromise between bandwidth and processing time, we selected LWE encryption optimized for small query and response sizes. For small files (256 bytes), the XPIR optimizer selected a polynomial of degree 2048 and 60-bit modulus with varying recursion and aggregation values depending on database size. When considering 1 MByte files the optimizer selected differing LWE flavors, all above the 80-bit security level, with polynomial degrees ranging from 2048 bits to 4096 bits and modulus bits ranging from 60 bits to 180 bits.

For each database size we measured the size of queries a client would have to generate and

Table 2.2: **Small File Scaling** — We measured query and response sizes generated for several different database sizes with 256 byte files to determine how the system scales for a Twitter-like content server. We find that even at millions of files, the system remains relatively practical: an 8 MByte query and 2 MByte response are needed to select a single file privately (with accompanying proof of censorship) from 10 million files.

Number of files	5k	10k	50k	100k	500k	1M	10M	100M
Recursion depth	1	1	2	2	2	3	3	3
Aggregation value	300	420	96	64	126	16	16	15
Query size (MBytes)	0.53	0.75	1.44	2.5	3.9	3.8	8.0	17.7
Reply size (MBytes)	0.56	0.78	1.44	1.0	2.0	2.0	2.0	2.4

the time to generate them. On the server side we measured the amount of replies it needed to send (based on file size and aggregation value) and the time it took to construct those queries. And finally, back on the client side, we measured the time it took to extract the actual response from the given replies. The results of our experiments are shown in Tables 2.2, 2.3 and Figure 2.4.

Table 2.3: **Large file scaling** — We also measured query and response sizes assuming 1 MByte files, to approximate our system being applied to an image or video-streaming content server. The XPIR optimizer chooses different security, polynomial and modulus parameters for each database size for 1 MByte files, due to it attempting to limit the amount of data the client has to send and receive over the network. While the parameters that are chosen are not always the most secure or privacy preserving, these parameters do result in the least amount of bandwidth necessary for clients to use. Additionally, while overheads are low, even with only 50 thousand files in a database, responses are 73 MBytes for a single megabyte file, likely making application of proof of censorship impractical to video streaming content providers.

Number of files	5k	10k	50k	100k	500k	1M	10M	100M
Recursion depth	2	2	2	2	2	2	3	3
Aggregation value	1	1	1	1	1	1	1	1
Query size (MBytes)	17.8	25.0	14.0	19.8	44.2	62.5	121.3	174.1
Reply size (MBytes)	32.8	32.8	73.1	73.1	83.7	83.7	138.4	199.4

For comparison, we allowed the optimizer to select Paillier 1024, which generates a 34.7 KB query and a 9.6 KB response when querying a 256 byte file from 1 million files. While this is several orders of magnitude smaller than LWE, server reply generation took nearly 2 hours for a single query. However, query generation and extraction times on the client were still fast (100ms and 77ms respectively). This suggests that with specialized modular arithmetic hardware on the centralized server, reply times could be considerably improved, potentially making this scheme practical and very attractive for its small query and response sizes.

We also note that a server need not contain all of its files in a single PIR database. Instead, one could partition the file space into many buckets, and requests could be made over a much

smaller number of files in each bucket, similar to an idea proposed in bbPIR [167]. This allows for a tradeoff between the granularity at which a server can censor without producing a proof, and the efficiency or scalability of the system as a whole. With more buckets, a server could choose to not respond to queries for an entire bucket without producing a proof of censorship. If each bucket only contained a few files, the collateral damage to censoring the entire bucket could be small. In addition, if multiple buckets exist, a server that wishes to censor could place new files in their own bucket, allowing for free censorship of the file in the future should it desire. To combat this, clients could choose which buckets they insert files into.

2.6 Discussion

In this section, we discuss possible attacks and defenses on our proof of censorship system, as well as describe potential incentives and future applications for this type of proof.

2.6.1 Attacks

Server Produces Invalid Tickets The server can choose to either not sign or not produce tickets during file upload, allowing it to delete those files later. However, the server does not know what file is being uploaded at the time of upload: the file could be encrypted, where the information to download the file (e.g. URL) contains the key used to decrypt it. Thus, the server must choose to produce tickets or not without knowing file contents, making it difficult to target specific content. In addition, clients that do not receive a valid ticket could reupload the file (perhaps through an anonymous proxy [55]) until they receive one.

Server Doesn't Sign Replies By using PIR, the server does not know what file is being downloaded. Therefore, it cannot know if a particular request is for the file it wishes to censor. It can choose to never sign replies (or sign them randomly), but it does so without knowledge of the file involved. In this case, we can require that honest clients refuse to extract downloaded files unless the PIR reply contains a valid signature, meaning that the server effectively would be censoring unknown files that were requested. This is effectively the same as a server shutting down

its service by not responding to any queries.

Incorrect Timestamps A server can advance timestamps in tickets (or delay them in replies), tricking verifying clients into thinking a proof of censorship is invalid because the reply appears to come after the ticket (a feature used to protect against client forgeability). To solve this, we require clients to check the timestamps of tickets and replies and ensure they are recent before considering the response valid. This may still leave room for an equivocating server to leave a small window to censor a file, but if uploaded files are encrypted, the server will have little information to determine if the file should be censored before the window has passed.

2.6.2 Incentives and Applications

How do we use a proof of censorship? The first answer is as a reputation sentinel: Proofs of censorship can be used to show others that a censoring content provider cannot be trusted. However, this only works if the proof is widely distributed (and not itself censored). As the proof is on the order of a few megabytes, it should be transferable to other verifiers via social media, email, or other content-sharing sites.

An intriguing possibility is to use proofs of censorship to impose a financial penalty. This could take the form of a smart contract in Ethereum [172] that controls a bond posted by the provider which is payable (in part) to whoever submits a valid proof of censorship.

Another option is to force the provider to reveal some sensitive secret if they ever censor a file. This can be accomplished via either multi party computation or trusted hardware. In either case, the content provider is forced to blindly interact with someone and evaluate a function that either returns nothing or, if fed a valid proof of work, returns the secret. For example, every request for a file could be accompanied by a query to the provider's SGX enclave [83] via an encrypted channel that is terminated within the enclave. The enclave could derive a private key from the server's secret, and use it to sign responses (requiring the enclave to have the secret loaded). If the enclave receives a valid proof of censorship, it returns the secret encrypted under the requester's key. Otherwise, it returns a dummy value. If the server chooses to provide no response at all, the

honest client aborts.[3] This forces a censoring provider to either take down their whole system service once they have censored a file, or risk leaking their secret.

Proofs of censorship may also be purposefully created by a provider to disclose the extent of what they have censored. In an effort toward transparency, many content providers voluntarily report takedown notices and removal requests to Lumen [104] (formerly Chilling Effects). However, there is no guarantee that a provider hasn't withheld reports from this database. To combat this, providers could submit their own proofs of censorship to the database, and any proofs of censorship not present in the database serve as evidence of the provider attempting to secretly censor without reporting.

2.7 Related work

Previous research has explored alternative solutions to the problem of content censorship.

One such approach is proof of retrievability, proposed by Juels et al. in 2007 [88]. In this model, servers provide cryptographic proof to users that they have access to a file. However, as previously mentioned, this does not mean that a server must provide such a proof for every file requested: if the server knows what portion of a file is being requested, they can censor specific parts or files by simply not responding.

Several works have provided monetary incentives for successful proofs of retrievability. Permacoin proposes a cryptocurrency with proof of retrievability of an agreed-upon file in place of the traditional proof of work [118]. This encourages miners to keep portions of the file around in order to qualify for mining rewards associated with the currency. Lavinia incentivizes publishers to store documents by having a third-party verifier check and provide payments to well-behaved publishers in a distributed system [24].

Numerous projects have detailed the idea of combining files to discourage their removal from servers. Tangler [164] stores files across multiple servers and "entangles" newly uploaded files with

[3] To prevent lazy but well meaning clients simply ignoring empty enclave responses, the enclave could instead return a decryption key needed for the particular file.

existing files using Shamir secret sharing [148]. This entanglement means that deleting an old file may inadvertently remove more recently uploaded files that depend on it, increasing the collateral damage in censoring a file. Dagster [155] `xor`s blocks of data together requiring users to obtain multiple blocks from the server in order to retrieve one block of desired data. This combining of blocks ties files together in such a way that if older blocks are removed, newer blocks are no longer accessible. However, newly uploaded files are not protected, and access patterns of files could be used to detect what file is being downloaded.

Others have leveraged distributed systems to make content harder to censor. Publius [165] allows clients to upload encrypted files to multiple servers along with parts of a split encrypted key in such a way that a threshold of servers behaving honestly will allow the file to be retrievable. Freenet [34] provides a volunteer-based peer-to-peer network for file retrieval with the aim of allowing both authors and readers to remain anonymous. Tor [55] supports hidden services, where a central provider could potentially obscure its network location while hosting objectionable content. However, all of these schemes lack a mechanism to discourage servers or participants from misbehaving, opting instead to either hide or share responsibility of hosted content. Moreover, they provide no guarantees on file lifetimes, which is determined by the resource constraints of the participating servers.

2.8 Future Work

Proof of censorship could be extended in several directions and applications. As mentioned previously, monetary incentives built on top of such proofs could encourage content providers to deploy such a system and keep them honest.

Beyond this, there are many open problems in how to apply proof of censorship to different applications in an efficient manner. For instance, applying this scheme to a video streaming service would be a difficult engineering task, as large file sizes, database volumes, and high bandwidth demands require low-overhead efficient solutions. To solve this problem, it may be possible to combine proof of censorship with other scalable private media retrieval systems such as Popcorn [72].

Proof of censorship could also augment existing key transparency applications, such as Certificate Transparency [99] or CONIKS [116]. Although these systems already detect server equivocation when a server modifies a particular object, they fail to provide any sort of guarantee on responses for every object in their certificate or key store. Using proof of censorship, these systems could provide such an assurance in addition to the protections provided.

2.9 Acknowledgements

We would like to thank the anonymous reviewers for their feedback on our work, as well as our shepherd Ryan Henry, who provided useful direction and thoughtful discussion on this paper.

2.10 Conclusion

Content censorship from providers remains a growing problem. As network effects push users and content toward more centralized provider platforms, legal and political pressures have followed suit. While centralized providers can claim they stand for free speech or open access, they have no mechanism to prove they do.

In this paper, we have presented a scheme whereby a content provider can stand behind such a claim cryptographically. By deploying this scheme, providers will create a cryptographically verifiable and transferable proof of censorship should they delete or remove access to a specific file. The threat of this proof provides a disincentive to content providers from even temporarily censoring a file, as their reputation with respect to Internet freedom is at stake.

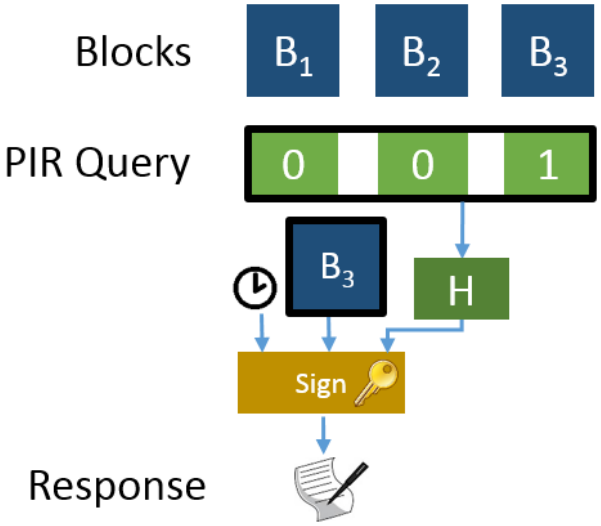

Figure 2.3: **Proof of Censorship** — To verify a file is still in place, a client constructs a PIR query for its block. The server, without knowing which block was requested, signs the encrypted PIR query and the response it produces. The client decrypts the response, and can verify the data is correct. If it is not, the client can combine the signed response, the parameters it used to generate the PIR query, and the original ticket for this file to produce a stand-alone proof-of-censorship that can be verified by a third party.

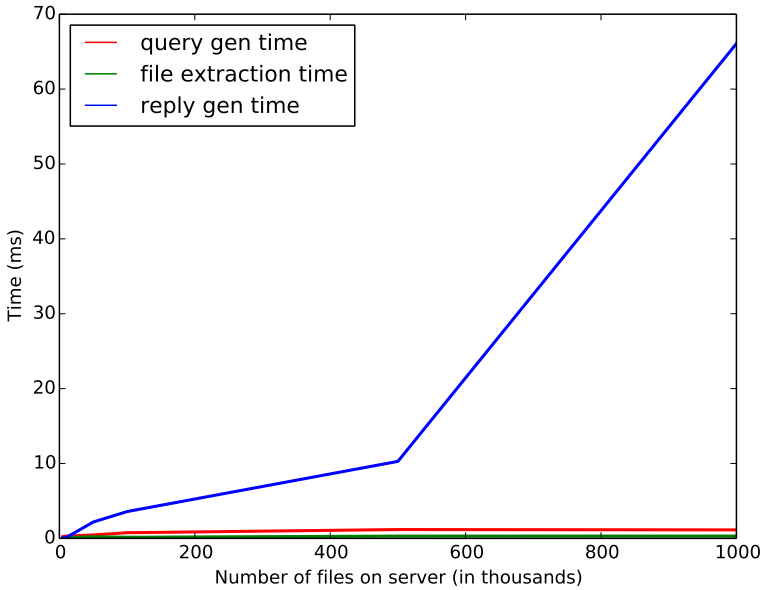

Figure 2.4: Amount of time taken for the actions necessary for our tool to work. The time taken for the client to generate queries, and to extract the response from the server replies is very minimal compared to the amount of time that is spent by the server generating the replies to send back to the server. This time however can be decreased by using servers with hardware specifically designed to do these computations.

Chapter 3

Improving Signal's Sealed Sender

3.1 Introduction

Secure end-to-end encrypted messaging applications, such as Signal, protect the content of messages between users from potential eavesdroppers using protocols like off-the-record (OTR) messaging [25, 53]. These protocols guarantee that even the service provider itself is unable to read communication between users. However, these protocols do not protect conversation **metadata**, including sender, recipient, and timing. For instance, if Alice sends a message to Bob, the server will learn that there is a relationship between those two users and when they communicated.

Protecting metadata. While leaking metadata may appear reasonable when compared to revealing the contents of the messages, observing metadata can have serious consequences. Consider that Alice may be a whistleblower communicating with a journalist [112] or a survivor of domestic abuse seeking confidential support [75]. In these cases, merely knowing **to whom** Alice is communicating combined with other contextual information is often enough to infer conversation content without reading the messages themselves. Former NSA and CIA director Michael Hayden succinctly illustrated this importance of metadata when he said the US government "kill[s] people based on metadata" [86].

Signal's recent **sealed sender** feature aims to conceal this metadata by hiding the message sender's identity. Instead of seeing a message from Alice to Bob, Signal instead observes a message to Bob from an anonymous sender. This message can only be decrypted by Bob, who then learns from the payload that the message originated with Alice. Ideally, using the sealed sender protocol

breaks the link between the sender and the receiver, preventing Signal from recording sender-recipient pairs, if ever compromised or compelled to do so.

While sealed sender is currently only deployed by Signal, Signal's design decisions are highly influential for other secure messaging platforms as it is a leader in deploying cutting-edge secure messaging features; the Signal protocol has been integrated into other services like WhatsApp. Understanding and uncovering flaws in sealed sender is therefore not only important to protecting the privacy of Signal's millions[1] of users [71], but also helps make sure sealed sender fully realizes its goal before it is integrated into other services with other sets of users.

A new SDA on message timings. We present a new statistical disclosure attack (SDA) applicable to messages in Signal's sealed sender, that would allow the Signal service—if compelled by a government or compromised—to correlate senders and receivers even when using the sealed sender feature. Previously, **statistical disclosure attacks** (SDAs) have been studied since the 2000s to link senders and recipients in anonymous mix networks [47, 111, 135, 49, 108]. These attacks work by correlating sender and receiver behavior across multiple rounds of the mix.

It is not immediately obvious how SDAs could be applied in the context of sealed sender messages, since there is no mix network and the identities of senders are (by design) never revealed. Thus, it is not clear how even the server could apply SDA attacks, since it only learns the destinations of messages, and never sources.

In this paper, we observe that, **by assuming that most messages receive a quick response**, we can overcome these seeming limitations of sealed-sender messaging and employ a SDA-style attack to de-anonymize sender-recipient pairs after passively observing enough messages.

Moreover, and crucially, **this quick-response assumption is guaranteed to be true in the presence of delivery receipts**, a feature of Signal's current implementation that cannot be disabled by the user. When Alice sends Bob a sealed sender message, Bob's device will automatically generate a delivery receipt that acknowledges Alice's message. Although this delivery receipt is also sent via sealed sender to Alice, the predictability of its timing makes our attack more effective.

[1] Signal does not publicly disclose its user count, but the app has been downloaded millions of times.

The differences between sealed sender messaging and a general mix network allow us to develop a simple, tailored SDA-style attack, using ideas similar to [111], which can be used to de-anonymize a conversation between two parties. Compared to prior work, our attack is more limited in scope, but is also more efficient: it runs in linear-time in the amount of traffic observed, and we prove that the probability our attack succeeds increases exponentially with the number of observations.

We validate the practicality of the timing attack in two ways. First, using a probabilistic model of communication, we prove a bound on the probability that Alice can be identified as communicating with Bob after a finite number of messages, independent of other users' activity. The probability also scales logarithmically with the number of active users.

Second, we run simulations to estimate the effectiveness of the attack in practice. In the most basic simulation, Alice can be uniquely identified as communicating with Bob after fewer than 10 messages. We also add complicating factors such as multiple simultaneous conversations with Alice and/or Bob and high-frequency users in the system, and show that these delay but do not prevent Alice from being de-anonymized.

Sealed sender conversations. To fix this problem, we provide a series of practical solutions that require only modest changes to Signal's existing protocol. We first define a simulation-based security model for sealed sender **conversations** (rather than just single messages) that allows the original recipient of the sealed sender message to be leaked but never the initiator of that message (sender) through the lifetime of the conversation. We then present three solutions that accomplish the goal of sealed sender conversations. Each is based on ephemeral identities, as opposed to communicating with long-term identifiers, such as the keys linked to your phone number in Signal. Each additional solution provides additional security protections.

Our first solution provably provides **one-way sealed-sender** conversations, a new security guarantee for which we provide a formal, simulation based definition. In this protocol, Alice initiates a sealed-sender conversation by generating a new ephemeral, public/secret key and anonymously

registers the ephemeral public key with an anonymous mailbox via the service provider. Alice then uses a normal sealed sender message to the receiver Bob to send the anonymous mailbox identifier for his replies. Alice can retrieve Bob's replies sent to that anonymous mailbox by authenticating with her ephemeral secret key, and the conversation continues using traditional sealed sender messages between Bob's long-term identity and the anonymous mailbox Alice opened.

We show that this solution can be further enhanced if both Alice and Bob use ephemeral identities, after the initial message is sent (using sealed sender) to Bob's long-term identity. This protocol provides both sender and receiver anonymity for the length of a conversation if the server is unable to correlate Bob's receipt of the initial message and his anonymous opening of a new mailbox, meaning the server has only one chance to deanonymize Bob. Importantly, even if the server is able to link these two events, this extension still (provably) provides **one-way sealed-sender**.

Neither of the above solutions offer authentication of anonymous mailboxes at the service provider, e.g., Signal. A malicious user could open large numbers of anonymous mailboxes and degrade the entire system. We offer an overlay solution of **blind-authenticated anonymous mailboxes** for either one-way or two-way sealed-sender conversations whereby each user is issued anonymous credentials regularly (e.g., daily) that can be "spent" (verified anonymously via a blind signatures) to open anonymous new mailboxes. To evaluate the practicality of using anonymous credentials in this way, we run a series of tests to compute the resource overhead required to run this overlay. We estimate that running such a scheme on AWS would cost Signal approximately $40 each month to support 10 million anonymous mailboxes per day.

Our contributions. In this paper, we will demonstrate

- A brief analysis of how the Signal protocol sends messages and notifications based on source code review and instrumentation (Section 3.2.2);
- The first attack on sealed sender to de-anonymize the initiator of a conversation in Signal (Section 3.3);
- Validation of the attack via theoretical bounds and simulation models (Section 3.4);

- A new security model that defines allowed leakage for sender-anonymous communication;
- A set of increasingly secure solutions, that are either one-way anonymous, two-way anonymous, and/or provide anonymous abuse protections. (Section 3.6);
- An evaluation of the resource overhead introduced by using blind signatures to prevent anonymous mailbox abuse, and estimates of its effective scalability to millions of users (Section 3.6.5); and
- Immediate stopgap strategies for Signal users to increase the difficulty of our attack (Section 3.7.1).

We include related work and the relevant citations in Section 3.8. We also want to be clear about the limitations of our work and its implications:

- We do **not** consider network metadata such as leakage due to IP addresses. See Section 3.2.3 and the large body of existing work on anonymizing proxies such as Tor.
- We do **not** consider messaging with more than two parties, i.e. group messaging. This is important future work; see the discussion in Section 3.7.3.
- Our attack does **not** suggest that Signal is less secure than alternatives, or recommend that users discontinue using it. Other messaging services do not even attempt to hide the identities of message senders.
- We do **not** believe or suggest that Signal or anyone else is using this attack currently.
- While we have implemented the core idea of our solution in order to estimate the cost of wider deployment, we have **not** undergone the serious engineering effort to carefully and correctly integrate this solution with the existing Signal protocol software in order to allow for practical, widespread deployment.

Responsible Disclosure. We have notified Signal of our attack and solutions prior to publication, and Signal has acknowledged our disclosure.

3.2 Background

We now give some background on the structure and types of messages in the Signal protocol [110], used in both the Signal and WhatsApp applications.

3.2.1 Sealed Sender Messages

Although secure end-to-end encrypted messaging applications like Signal protect the contents of messages, they reveal metadata about **which** users are communicating to each other. In an attempt to hide this metadata, Signal recently released a feature called sealed sender [106] that removes the sender from the metadata intermediaries can observe.

To send a sealed sender message to Bob, Alice connects to the Signal server and sends an encrypted message to Bob anonymously[2] . Within the payload of this encrypted message, Alice includes her own identity, which allows Bob to authenticate the message. Importantly, Signal still learns Bob's identity, which is needed in order to actually deliver it. The structure of sealed sender messages are illustrated in Figure 3.1.

Due to sender anonymity, Signal cannot directly rate-limit users to prevent spam or abuse. Instead, Signal derives a 96-bit **delivery token** from a user's profile key, and requires senders demonstrate knowledge of a recipients' delivery token to send them sealed sender messages. By only sharing this delivery token with his contacts, Bob limits the users who can send him sealed sender messages, thus reducing the risk of abuse[3] .

3.2.2 Types of Messages

We manually reviewed and instrumented the Signal messenger Android 4.49.13 source code [127] in order to understand the types of messages Signal sends. In addition to the messages that contain content to be delivered to the receiver, there are several event messages that can be sent automat-

[2] As we note in our threat model, we do not consider the information leakage from networking.

[3] There are a number of options available to Bob that can allow more fine-grained access control to his delivery token. Bob can opt to receive sealed sender messages from anyone even without knowledge of his delivery token, but this is disabled by default. Additionally, Bob can regenerate his delivery token and share it only with a subset of his contacts to block specific users.

ically, as discussed below. All of these messages are first padded to the next multiple of 160 bytes, then encrypted and sent using sealed sender (if enabled), making it difficult for the Signal service to distinguish events from normal messages based on their length.

Normal message. A normal text message or multimedia image sent from Alice to Bob is the typical message we consider. A short (text) message will be padded to 160 bytes, and longer messages padded to a multiple of 160 bytes, before encryption.

Delivery receipt. When Bob's device **receives** a normal message, his device will automatically send back a delivery receipt to the sender. When Alice receives the delivery receipt for her sent message, her device will display a second check mark on her sent message to indicate that Bob's device has received the message (see Figure 3.2). If Bob's device is online when Alice sends her message, the delivery receipt will be sent back immediately. We measured a median time of 1480 milliseconds between sending a message and receiving a delivery receipt from an online device. (See Figure 3.3 for CDF of times.) These receipts **cannot be disabled** in Signal.

Read receipt (optional). Bob's device will (optionally) send a read receipt to the sender when he has **viewed** a normal message, triggering a UI update on Alice's device (see Figure 3.2). Unlike delivery receipts, Bob can disable read receipts. However, Alice may still send read receipts for messages she receives from Bob. If Bob receives a read receipt but has the feature disabled, his user interface will not display the notification.

Typing notifications (optional). Alice's device will (optionally) send a **start typing** event when Alice is entering a message, which Bob's device will use to show that Alice is typing. If she does not edit the message for 3 seconds, a **stop typing** event will be sent. Each sent message is accompanied by a **stop typing** event to clear the receiver's typing notification. Like read receipts, typing notifications can be disabled such that the user will not send or display received notifications.

3.2.3 Threat Model

We assume that the service provider (e.g. Signal) passively monitors messages to determine which pairs of users are communicating. This models either an insider threat or a service provider compelled to perform surveillance in response to a government request. We assume Alice and Bob have already exchanged delivery tokens and they communicate using sealed sender. Once initiated, we assume that Alice and Bob will continue to communicate over time. Finally, we also assume that many other users will be communicating concurrently during Alice and Bob's conversation, potentially with Alice and/or Bob.

The service provider cannot view the contents of the encrypted sealed sender messages, but knows the destination user for these messages (e.g. someone sends a message to Bob). We assume that Alice and Bob have verified their respective keys out of band, and that the applications/devices they are using are secure. Although the service provider publishes the application, they typically distribute open-source code with deterministic builds, which we assume prevents targeting individual users.

We note that the service provider could infer a sender's identity from network metadata such as the IP address used to send a sealed sender message. However, this is a problem that could be solved by using a popular VPN or an anonymizing proxy such as Tor [141, 56]. For the purposes of this paper, we assume that users who wish to remain anonymous to Signal can use such proxies (e.g. Orbot [11]) when sending sealed sender messages (and, in our solution, when receiving messages to ephemeral mailboxes), and we do not use network metadata in our attack.

In terms of impact, we note that a recent study suggests as many as 15% of mobile users already use VPNs every day [82]; this prevalence is even higher in east Asia and, presumably, among vulnerable user populations.

3.3 Attack Description

We will present a kind of **statistical disclosure attack** (SDA) that can be used to de-anonymize a single user's contacts after a chain of back-and-forth messages, each of which is sent using sealed sender.

We first explain how, especially in the presence of delivery receipts, a sealed-sender messaging system can be viewed as a kind of mix network; this observation allows for the use of SDAs in our context and can be viewed as one of our contributions.

Next, we detail a simple attack for our specific use-case of sealed sender messaging, which can be viewed as a special case of an SDA attack proposed in [111].

3.3.1 From mixnets to sealed-sender

In anonymous networking, a simple **threshold mix** works as follows: When Alice wants to send a message to Bob, she instead encrypts it and sends it to a trusted party called the **mix**. Once the mix receives messages from a certain threshold τ number of other senders, the mix decrypts their destinations, shuffles them, and sends all messages out to their destinations at once. In this way, a network attacker can observe which users are sending messages and which are receiving message, but cannot easily infer which pairs of individuals are directly communicating.

The basis of SDAs, first proposed by [47], is that the messages sent through the mix over multiple rounds are not independent; a user such as Alice will normally send messages to the same associates (such as Bob) multiple times in different rounds. In the simplest case, if Alice sends messages only to Bob, and the other users in each round of mixing are random, then a simple **intersection attack** works by finding the unique common destination (Bob) out of all the mixes where Alice was one of the senders.

Over the last two decades, increasingly sophisticated variants of SDAs have been proposed to incorporate more complex mix networks [111], infer sender-receiver connections [108], adapt to the possibility of anonymous replies [49], and to use more powerful techniques to discover information

about the entire network topology [50, 135]. Fundamentally, these all follow a similar anonymous networking model, where an attacker observes messages into and out of a mix network, and tries to correlate senders and receivers after a large number of observations.

At first, it seems that the setting of sealed-sender messaging is quite different: the server (acting as the mix) does not apply any thresholds or delays in relaying messages, and the sender of each message is completely anonymous. Our key observation is that, **when many messages receive a quick reply**, as will be guaranteed in the presence of delivery receipts, a sealed-sender messaging system can be modeled as a kind of mix network:

- The **recipient** of a message, Bob, is more likely to send some reply in a short time window immediately after he receives a message: we call this time window an **epoch**.
- Bob's reply to Alice is "mixed" with an unknown, arbitrary number of other messages (which could be either normal messages or replies) during that epoch.
- The recipients of all messages during that epoch (following the message Bob received), can be considered as the message recipients out of the mix. Alice, who originally sent a message to Bob and is expected to receive a quick reply, will be among these recipients.

The task of our SDA, then, is to observe many such epochs following messages to a single target user, Bob, and attempt to discern the user Alice who is actually sending messages to Bob.

3.3.2 Attack Overview

Before proceeding to an overview of our attack, we first fix the terminology we will use:

Target/Bob The single user who is being monitored.

Associate/Alice Any user who sends some message(s) to the target Bob during the attack window

Non-associate/Charlie Any other user not sending messages to the target Bob.

Attack window The entire time frame under which the attack takes place, necessarily spanning multiple messages sent to the target Bob.

Target epoch A single epoch during the attack window immediately following a sealed sender message to the target. The epoch length is fixed depending on how long we should expect to see a response from the recipient.

Random epoch A single epoch during the attack window, of the same length as a Target epoch, but chosen uniformly at random over the attack window independently from Bob.

As discussed above, our attack setting is that a single user, Bob, is being targeted to discover an unknown associate Alice who is sending messages to Bob. Our SDA variant is successful when we can assume that Alice is more likely to appear as a message recipient in a **target epoch** immediately following a message received by Bob, than she is to appear in a **random epoch** when Bob did not receive a message.

Specifically, our attack is executed as follows:

(1) Create an empty table of counts; initially each user's count is zero.

(2) Sample a **target epoch**. For each user that received a message during the target epoch, increase their count in the table by 1.

(3) Sample a **random epoch**. For each user that received a message during the random epoch, decrease their count in the table by 1.

(4) Repeat steps 2 and 3 for n target and random epochs.

(5) The users in the table with the highest counts are most likely to be **associates** of the target.

Figure 3.4 gives a small example to illustrate this attack.

This is similar to the original SDA of [47], with a few of the improvements from [111] that allow for unknown recipient and background traffic distributions, more complex mixes such as pool mixes, and dummy traffic. In our setting, this means that we do not need to know **a priori** which users in the system, or which associates of the target user, are more or less likely to receive messages. We also do not need a guarantee that a reply is sent during **every** target epoch, or that the reply is always sent to the same associate Alice.

Essentially, our attack relies only on the assumptions that the distribution of background noise in each target/random epoch pair is the same, and that associates of the target are more likely to appear in target epochs than random epochs. Under only these assumptions, we can see that the expected count of any non-associate, over enough samples, is zero, while the expected count of any associate will increase linearly with the number of samples.

Compared to existing SDAs in the literature, our attack is more limited in scope: it does not attempt to model the complete distribution of all connections in the system, but merely to separate the associates from non-associates of a single target user. We also assume that the number of target and random epochs are the same (though this limitation would be easy to overcome). These limitations allow our attack to be very efficient for the attacker, who just needs to update a table for each user in each sample, and then find the largest values at the end to identify Bob's (likely) associates.

Clearly the question that remains is, how large must the number of samples n be in order for this attack to succeed? As we will see in the next section, the limited scope of our attack also makes it efficient in this sense: in reasonable settings, our attack requires only a handful of epochs to identify the target's associates with high probability.

3.4 Attack Evaluation

In this section, we evaluate our attacks from Section 3.3 first from a theoretical perspective, and second using a custom simulation.

While our attack is a variant of existing statistical disclosure attacks (SDAs) in the literature, the setting is slightly different, and our goals are more modest, seeking only to de-anonymize the contacts of a single target user.

3.4.1 Theoretical analysis of attack success

Here we provide statistical bounds to estimate the number of epochs needed for our attack to successfully de-anonymize one participant in a conversation. As before, say Bob is the target of

the attack, and we wish to find which other users are communicating with Bob.

Roughly speaking, we demonstrate that (1) all users in conversations with Bob can be identified provided he is not in too many other simultaneous conversations with other users, and (2) the number of epochs needed for this de-anonymization depends logarithmically on the total number of users. These results hold under some regularity assumptions on communication patterns, which are most sensible for short periods of back-and-forth messaging.

Statistical Model. Our statistical analysis relies on the following assumptions:

(1) The probability of receiving a message during any epoch is independent of receiving a message during any other epoch.

(2) Each user u (both associates and non-associates) has a fixed probability r_u of receiving a message during a random epoch.

(3) Any **associate** u has a fixed probability t_u of receiving a message during a target epoch, where $t_u > r_u$.

(4) Every **non-associate** u has the same probability of receiving a message during a target or random epoch, i.e., $t_u = r_u$.

The last assumption states that the communications of non-associates is not correlated with the communication patterns of Bob, which makes intuitive sense, as they are not involved in conversations with Bob. The regularity (that these probabilities are fixed and the events are independent) is most reasonable when considering short attack windows, during which any user's activity level will be relatively constant.

Theoretical attack success bound. In our attack, all users in the system are ranked according to their chances of being an associate of Bob after some number of target and random epochs. We now provide bounds on the number of epochs necessary to ensure that an arbitrary associate Alice is ranked higher than **all** non-associates.

Theorem 1. *Assume m total users in a messaging system. Let Alice be an associate of the target Bob with probabilities r_a, t_a of appearing in a random or target epoch respectively. Then, under the*

stated assumptions above, the probability that Alice is ranked higher than all non-associates after n random and target epochs is at least

$$1 - \frac{m}{{c_a}^n},$$

where $c_a = \exp((t_a - r_a)^2/4) > 1$ is a parameter that depends only on Alice's probabilities t_a and r_a.

The proof is a relatively standard analysis based on Hoeffding's inequality [80], and can be found in Appendix A.

We point out a few consequences of this theorem:

- The success of the attack depends only on the target user Bob and his sought-after associate Alice, not on the relative activity of any other users.
- The number of epochs needed to de-anonymize Alice with high probability scales **logarithmically** with the total number of users.
- The attack succeeds most quickly when Bob is in few other conversations (so t_a is large) and Alice is communicating mostly just with Bob (so r_a is small).

The following corollary, which results from solving the inequality of Theorem 1 and applying a straightforward union bound, gives an estimate on how many epochs are necessary to discover all of Bob's contacts with high probability.

Corollary 2. *Let $0 < p < 1$ be a desired probability bound, and assume m total users in a messaging system, of whom b are associates of a target user Bob, where the i'th associate has probabilities r_i, t_i of appearing in a random or target epoch respectively. Then, under the previous stated assumptions, with probability at least p, all b associates of Bob are correctly identified after observing*

$$\frac{4}{\min_i (t_i - r_i)^2} \left(\ln(m) + \ln(b) + \ln\left(\tfrac{1}{1-p}\right) \right)$$

target and random epochs.

Comparing to prior work, the closest SDA which has a similar theoretical bound is from

Danezis [47][4] . That work makes much stronger regularity assumptions than our model, assuming essentially that (1) all epochs contain the same number of messages (2) every target epoch contains exactly one reply from Bob, (3) Bob receives a message from each associate with uniform probability, and (4) all other users, and recipients, are selected uniformly at random from all m users. Later work also includes a theoretical bound [135], but their model is much more general than ours, where they seek to reveal the entire network rather than a single target user.

3.4.2 Attack simulation

We cannot directly validate the effectivenesses of our attacks in practice, as we do not have access to Signal's servers and there is no public sample dataset of Signal sealed sender messages. Instead, we perform **simulations** based on generalized but realistic assumptions on message patterns. We do not claim our simulations will reveal the exact number of messages needed to deanonymize a particular user, as that would depend on exact messaging patterns. Rather, our simulations give a sense of the order of magnitude of messages needed to deanonymize a user.

We simulated sequences of target and random epochs (e.g. epochs where Bob does or does not receive a message) and ranked users by their score. Recall that a user's score increases if they appear in a target epoch. We simulated 1 million active users, with 800 messages per epoch. This corresponds to users sending on average about 70 messages per day, with 1 second epochs[5] .

Within each epoch, we select a random set of 800 message destinations. In a target epoch, Alice (the associate) is sent a message to represent Bob's delivery receipt that would be sent to her automatically. The remaining messages are chosen randomly: 25% of messages are selected as "repeat" messages (same sender and receiver) from prior epochs (representing one side of a prior conversation), and another 25% are selected as "responses" to messages in prior epochs (representing a conversation's response). The remaining 50% of messages are messages from and to a random pairing of users from the set of 1 million active users. We find that the percent of repeats/replies

[4] Unfortunately, there appear to be at least three slightly different versions of this bound in the published literature ([47, equation (6)]; [48, equation (9.8)]; [111, page 5]), making it difficult to compare bounds.

[5] Based off our observation of round-trip delivery receipt times

has limited impact on the number of epochs to identify an associate until over nearly all messages are repeats (i.e. each epoch is essentially the same small set of senders/receivers). We choose half of the epochs to be target epochs (where Alice messages Bob) and half as random (where Alice does not message Bob).

Social graph significance. We note our experiment does not rely on a particular social graph (or rather, assumes a fully connected one), as any user can message any other. In preliminary experiments, we examined the impact of several different graph generators that are designed to simulate social networks, but found no noticable change in our results. Specifically, we used the Erdös-Rényi [60] model, Barabási-Albert [15] model, Watts-Strogatz [169] model, and a fully connected graph, but found they all resulted in a similar number of epochs needed to deanonymize the associate (Alice). Given this result, we opted to use the fully connected graph model for simplicity.

Figure 3.5 shows the result of several attack simulations. We ran each attack simulation for 100 runs, and at each epoch, report the average rank of Alice's score based on our attack. First, in the "Alice Only" variant, only Alice messages Bob (and no one else). Even though there are thousands of other users messaging randomly, Alice's score quickly becomes the top ranked user: within 5 messages, she is uniquely identified as messaging Bob.

If multiple users are also messaging Bob while Alice does, it takes more total epochs to identify Alice (and her co-associates messaging Bob). In this scenario, each target epoch is selected to be either Alice or one of 5 co-associates that messages Bob (6 total conversations with Bob).

If there are popular users present (e.g. users that receive messages in a large fraction of all epochs), then it may be more difficult to identify Alice without accounting for them. However, since we remove users that also appear in a large fraction of random epochs, Alice is still eventually ranked uniquely as messaging Bob.

Finally, we combine our popular users and multiple messagers into a single simulation, which is dominated by the multiple messagers effects.

Summary. In the worst case, it takes on the order of 60 epochs to identify the users messaging

Bob. Note that only half of these are messages to Bob, and the other half are random epochs. If only one person is messaging Bob, the number of messages needed is under 5 to identify Alice as the associate of Bob.

3.5 Formalizing Sealed Sender Conversations

Sealed sender messages were initially introduced in Signal to obscure the communication graph. As we have just shown, the current instantiation fails to accomplish this goal. Before we present our solutions to this problem, we briefly discuss formalizations for the properties that a perfect implementation should accomplish. We call such a system a **sealed sender conversation**, because unlike sealed sender messages, the anonymity properties must be maintained throughout the lifetime of the conversation.

Our goal in introducing this formalization is to specify **exactly** how much information a service provider can learn when it runs a sealed sender conversation protocol. In a sealed sender conversation between two users, the mediating service provider should learn only the identity of the **receiver** of the first message, no matter the messaging pattern of the users. Unlike sealed sender messages, the anonymity of the sender must be maintained across the full conversation, not just individual messages. As such, we require a definition that argues about the privacy of the users at the **conversation** level, rather than at the message level, as in sealed sender messaging. We formalize sealed sender conversations by giving an **ideal functionality**, presented in Figure 3.6. We note that this definition fundamentally reasons over conversations, even if it does this in a message-by-message way by using an internal conversation table. Our ideal functionality captures our desired properties by specifying the maximum permissible information leakage for each message, depending on which member of the conversation sent the message.

Our ideal functionality models a sealed sender conversation and explicitly leaks certain information to the service provider. Users are able to do three things: (1) start a conversation, (2) send messages in an existing conversation, and (3) receive messages. When a user starts a new conversation, the initial receiver's identity is leaked to the service provider, along with a unique

conversation identifier `cid`. All subsequent messages sent in this conversation are linked with the identifier `cid`. If they are being sent to the initial receiver, their destination is leaked. Otherwise, the service provider learns that the message is part of some known `cid`, but never learns the identity of that end of the conversation. While we do not explicitly include timestamps in our modeling, timestamps are implicitly captured by our model because the service provider is notified immediately whenever the ideal functionality receives a message. This is equivalent because the absolute time at which a message is sent is not important in our context, just the relative time between messages.

Users receive messages via pull notifications. These pull notifications leak no more information than the message itself does; if the receiver is anonymous, then the pull notification process leaks no information about the receiver. While we formalize this notion using pull notifications, this is compabile with Signal-style push notifications, where the receiver and the server maintain long-lived TLS connections. These communication channels are equivalent to a continuous pull notification, and thus a simulator can easily translate between the two communication paradigms. Finally, because the service provider may arbitrarily drop messages, we give the service provider the power to approve or deny any pull notification request.

While leaking the conversation identifier might seem like a relaxation of sealed sender messages, we note that our timing attack succeeds by guessing with high likelihood the sender of a message. As such, Signal's sealed sender does not meet this ideal functionality, as our timing correlation attack in Section 3.3 shows. This is because the `cid` of a message, although not explicitly sent with the ciphertext, can be inferred with high probability by its timing. One final note is our definition does not prevent a service provider from using auxiliary information about a conversation (**e.g.** time zone information) to reidentify the initiator of the conversation. Such attacks are incredibly difficult to formalize and are beyond the scope of our work. Rather, we only require that the **protocol** itself cannot be used to reidentify the participants.

3.5.1 Security Definition for One-Way Sealed Sender Conversations

We now give a formal definition for one-way sealed sender conversations using a simulation based security definition. We present the ideal functionality for one-way sealed sender conversations in Figure 3.6. Importantly, this definition does not rule out learning information about the sender based on timing of sending messages, **e.g.** the sender's time zone. We model the service provider as a party P_{service} that can control delivery of messages and delivery receipts. Note that the ideal functionality leaks the contents of the message m to the service provider only if the receiver of that message is corrupted. This models that if the service provider can decrypt the messages it is relaying, it may make delivery decisions based on knowledge of the plaintext.

We say that a protocol securely realizes this ideal functionality (in the stand alone model) if a corrupted service provider and an arbitrary number of corrupted users cannot determine if they are interacting in the **real experiment** or with the **ideal experiment** with non-negligible probability in the security parameter λ. In the real experiment, the adversary starts by statically corrupting the service provider and any number of users. Then, each honest user follows it own arbitrary strategy, interacting with the service provider using the protocol. The corrupt parties can follow an adversarially chosen strategy. In the ideal experiment, the adversary again begins by statically corrupting the service provider and any number of users. Then, the honest players follow an arbitrary strategy but interact directly with the ideal functionality. The service provider and corrupted users interact with a simulator Sim, which mediates interaction between the adversary and the ideal functionality. At the end of each experiment, a distinguisher algorithm takes in the views of the service provider and the corrupted parties and attempts to determine if the interaction was in the real experiment or the ideal experiment. Note that because the simulator may not know which parties are interacting, it cannot leak this information to the adversary.

We denote the output of the ideal world experiment for any ideal world adversary Sim and honest players with arbitrary strategies P_H on inputs x as $\mathsf{Ideal}_{P_H,\mathsf{Sim}}(1^\lambda, x)$. We denote the output of the real experiment with adversary $\mathcal{A}$ running protocol Π on input x as $\mathsf{Real}_{P_H,\mathcal{A},\Pi}(1^\lambda, x)$. We

say that a protocol Π securely realizes the ideal functionality described in Figure 3.6 if there exists a simulator Sim such that

$$\left|\mathsf{Ideal}_{P_H,\mathsf{Sim}}(1^\lambda, x) - \mathsf{Real}_{P_H,\mathcal{A},\Pi}(1^\lambda, x)\right| < \mathbf{negl}(\lambda)$$

3.6 Solutions

We now present three protocols that follow the security definition from Section 3.5 and, in particular, prevent the attacks presented in Section 3.3. We first outline a **one-way** sealed sender conversation in Section 3.6.2, in which the initiator of the conversation remains anonymous. We prove that our construction meets the definition presented in Section 3.5.1. In Section 3.6.3, we extend this protocol to give better privacy to the receiver using a **two-way** sealed sender conversation. Finally, in Section 3.6.4, we address denial of service attacks that malicious users could launch against the server.

Overview of Solutions. Our key observation is that the attack described in Section 3.3 is only possible because both users in a conversation are sending messages to the other's long-term identity. Over time, these messages can be correlated, revealing the identities of the users. On the other hand, if **anonymous** and **ephemeral** identities are used instead, then user's true identities can remain hidden. However, anonymous identities lead to a bootstrapping problem: **how do users initiate and authenticate a conversation if they are using fresh, pseudonyms?**

In a **one-way sealed sender conversations**, the identity of one side of the conversation is leaked, namely the initial message receiver, in order to solve this bootstrapping problem. This closely models the situation of a whistle-blower, where the informant wishes to stay anonymous, but the reporter receiving the information can be public. At a high level, the initiator of the conversation begins by creating a fresh, anonymous identity and then sends this identity to a receiver via a normal sealed sender message (thus solving the bootstrapping problem). The conversation proceeds with the initiator of the conversation sending messages to the receiver using sealed sender (one way), and

the conversation receiver sending replies to the initiator's anonymous identity. Importantly, the identity of the initiator is never leaked, as no messages exchanged in the conversation contain that person's long-term identity. We prove that out protocol securely realizes the definition of sealed sender conversations presented in Section 3.5.1.

A straightforward extension is to move towards **two-way sealed sender conversations** where both parties use anonymous identities. This solution is described in Section 3.6.3. When an initiator starts a conversation as described above, the receiver also creates a new anonymous identity and sends it via sealed sender back to the conversation initiator. This protocol offers a single opportunity to link the receiver to their new, anonymous identity (by correlating the timing of the received message and the registering of a new public key), but, as we have shown, network noise makes it difficult to re-identify users with only a single event. Even in the unlikely case that the conversation receiver is linked to their long-term identity, we show that the conversation initiator remains anonymous.

Both protocols place the service provider at risk of denial of service attacks, and so in Section 3.6.4, we aim to limit the power of users to arbitrarily register anonymous identities. Allowing users to create unlimited anonymous identities would lead to strain on the service provider if there is no way to differentiate between legitimate anonymous identities and malicious ones. To prevent these attacks, users are each given a limited number of anonymous credentials that they can "spend" to register anonymous keys, reminiscent of the earliest e-cash systems [32]. These credentials can be validated by the service provider to ensure that a legitimate user is requesting an anonymous identity without revealing that user's identity. We use blind signatures to implement our anonymous credentials. We evaluate the practicality of this approach in Section 3.6.5 and show that it could be deployed cheaply for either one-way or two-way sealed sender conversations.

For simplicity, we assume that communicating users have already exchanged delivery tokens. Any protections derived from these delivery tokens can be added to the following protocols in a straightforward manner. Additionally, we assume users connect to the service provider via an anonymous channel, e.g., Tor or Orbot.

3.6.1 Preliminaries

Sealed Sender We assume that the service provider implements the sealed sender mechanism described in Section 3.2.1. Specifically, we assume that a client can generate a public/private key pair and publish their public key as an address registered with the service. If the server permits it through some verification process, the server will allow messages to be sent to that public key without a sender.

More formally, we assume that the system has a sealed sender encryption scheme Π_{ssenc}. While Signal does not give a proof of security for the scheme it uses, for our constructions we will assume that Π_{ssenc} is a signcryption scheme that satisfies ciphertext anonymity [102] and adopt the notation presented in [168] for its algorithms[6] . We say a sealed sender encryption scheme Π_{ssenc} is a set of three algorithms:

- $\mathsf{SSKeyGen}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$ generates a public/private key pair.
- $\mathsf{SSEnc}(m, \mathsf{sk}_s, \mathsf{pk}_r) \to c$ takes in a message m, the sender's secret key sk_s and the receiver's public key pk_r, and outputs a ciphertext c
- $\mathsf{SSDecVer}(\mathsf{sk}_r, c) \to \{(m, \mathsf{pk}_s), \perp\}$ takes in the receiver's private key sk_r and a ciphertext c and either outputs a message m,and the public key of the sender pk_s, or returns the error symbol $\perp$. (Note that this actually constitutes decryption followed by verification in the notation of [168], returning $\perp$ when either step fails.)

Formal security definitions are given in [168]. In short, the scheme satisfies (1) message indistinguishability, (2) unforgeability, and (3) ciphertext anonymity, meaning the ciphertext reveals nothing about the sender or receiver.

Blind Signatures The mechanism to prevent abuse for the creation of anonymous accounts relies on the cryptographic primitive of **blind signatures**, as first proposed by [32]. Blind signature schemes have 5 algorithms: BSKeyGen, BSBlind, BSSign, BSExtract and BSVerify. BSBlind takes

[6] We note that ciphertext anonymity is actually a stronger primitive than required, as there is no need for receiver anonymity.

in the public key of the signer, a message, and some randomness and outputs a blinded message. BSSign takes in the signer's private key and a blinded message and outputs a blinded signature. BSExtract takes in a blinded signature and the randomness used in blinding and outputs a normal signature. Finally, BSVerify takes in a message and the signer's public key and decides if the signature is valid.

The interaction between a server with the signing keypair sk, pk and a client is as follows:

(1) Client generates the blinded message

$b \leftarrow \mathsf{BSBlind}(m, pk; r)$ for $r \leftarrow_{\$} \{0,1\}^{\lambda}$

(2) Client sends b to the server for signing.

(3) Server computes the blinded signature

$s_{blind} \leftarrow \mathsf{BSSign}(b, sk)$ and returns it to the client.

(4) Client extracts the real signature

$s \leftarrow \mathsf{BSExtract}(s_{blind}, pk; r)$

(5) Client, in a different network connection, sends the initial message m and the real signature s to the server, who runs $\mathsf{BSVerify}(pk, m, s)$

The blind signature scheme should have the usual signature unforgeability property. Additionally, it should be impossible for a server to link the blinded message and blinded signature to the real message and real signature. We use the RSA-based construction of blind signatures from [32].

3.6.2 One-way Sealed Sender Conversations

First, we provide the construction of sealed sender conversations which we build on in this solution and those that follow. Recall that a sealed sender conversation reveals the flow of the conversation (including message timing, etc.) and the identity of the initial receiver, but at no point can the service provider identify the initial sender.

The intuition behind our solution is straightforward: when initiating a new conversation, a sender generates an ephemeral, per-conversation key pair. This key pair is registered with the

service provider anonymously, but otherwise is treated as a normal identity in the system. Throughout the lifetime of the conversation, this identity key is used instead of the long-term identity of conversation initiator. As long as the ephemeral public key is never associated with the long-term identity, and never used in any other conversations, the service provider cannot learn anything about the true identity of the user that generated that ephemeral identity.

Generally, the flow of a sealed sender conversation is as follows. During the setup, each sender P_s with long-term keys $(\mathsf{pk}_s, \mathsf{sk}_s)$ creates entries $(P_r, \mathsf{pk}_r, \mathsf{pk}_s)$ for each receiver P_r with public key pk_r. Some user, who we call the initiator, starts the conversation by running the **Initiate Conversation** protocol below where P_s generates and registers an ephemeral identity for a receiver P_r. Whenever the receiver comes online (or possibly immediately by receiving a push notification) and receives the appropriate information, they will locally associate the ephemeral key with the initiator for the duration of the conversation. From this point, both users may send messages using the **Send Message** protocol and receive those messages from the service provider via **Push Message**, over an open, long-term connection. The protocol **Open Receiver Connection** is used to establish a channel for such push notifications, either for a user's long-term mailbox, or for an ephemeral mailbox created for a single conversation.

Every user must maintain a **conversation table**, to remember where messages should be sent in all ongoing conversations. Each table entry stored by a user P_s is a tuple $(P_r, \mathsf{pk}_\beta, \mathsf{pk}_\alpha, \mathsf{sk}_\alpha)$, where P_r is the actual message recipient, pk_β is the recipient's mailbox (public key) to which the message is addressed, and $(\mathsf{pk}_\alpha, \mathsf{sk}_\alpha)$ is the key pair used to sign and encrypt the message. Depending on who initiated the conversation, one of pk_β or pk_α will correspond to an ephemeral identity pk_e, and the other will correspond to one of the long-term identities pk_r or pk_s.

Initiate One-Way Sealed Conversation to P_r:

(1) Initiator P_s does the following:

(a) looks up P_r's long-term public key pk_r

(b) generates fresh ephemeral keys $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSKeyGen}(1^\lambda)$

(c) encrypts $c \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSEnc}(\texttt{"init"} \| \mathsf{pk}_e, \mathsf{sk}_s, \mathsf{pk}_r)$

(d) connects to the service provider anonymously and sends $c \| \mathsf{pk}_e$ for pk_r

(e) appends $(P_r, \mathsf{pk}_r, \mathsf{pk}_e, \mathsf{sk}_e)$ to the conversation table

(f) Registers a new mailbox for the public key pk_e and uses **Open Receiver Connection** with keypair public key $\mathsf{pk}_e, \mathsf{sk}_e$ to establish a connection for push notifications.

(2) The service provider delivers c (sealed sender) to P_r based on pk_r, either immediately pushing the message or waiting for the receiver to come online.

(3) When the receiver P_r receives the message to its long-term mailbox pk_r, it:

(a) decrypts and verifies

$(\texttt{"init"} \| \mathsf{pk}_e, x, \mathsf{pk}_s) \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSDecVer}(\mathsf{sk}_r, c)$

(b) appends $(P_s, \mathsf{pk}_e, \mathsf{pk}_r, \mathsf{sk}_r)$ to the conversation table

(c) uses **Send Message** to send a delivery receipt to P_s (which now goes to pk_e from the conv. table)

Send Message to P_*

(1) Sender looks up freshest entry $(P_*, \mathsf{pk}_\beta, \mathsf{pk}_\alpha, \mathsf{sk}_\alpha)$ in the conversation table.

(2) Sender encrypts $c \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSEnc}(m, \mathsf{sk}_\alpha, \mathsf{pk}_\beta)$

(3) Sender sends c for pk_β to the service provider, anonymously if necessary.

(4) If there is an open connection associated with pk_β, the service provider uses **Push Message** for c over that connection. Otherwise, the service provider sets the message as pending in the mailbox associated with pk_β

Open Receiver Connection for $(\mathsf{pk}_\beta, \mathsf{sk}_\beta)$

(1) Receiver connects to the service provider and demonstrates knowledge of key pair $(\mathsf{pk}_\beta, \mathsf{sk}_\beta)$ such that there is a registered mailbox for public key pk_β

(2) The receiver and the server build a long-term connection for message delivery, indexed by pk_β

(3) If there are any pending messages in the mailbox associated with pk_β, use **Push Message** for those messages.

Push Message c **to** pk_β

(1) Service provider looks up an open connection indexed by pk_β. If such a connection exists, the service provider sends c over it

(2) Receiver decrypts c as $(m, \mathsf{pk}_\alpha) \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSDecVer}(\mathsf{sk}_\beta, c)$ and verifies an entry $(P_*, \mathsf{pk}_\alpha, \mathsf{pk}_\beta, \mathsf{sk}_\beta)$ exists in the conversations table, dropping it otherwise.

We prove that this construction securely realizes the definition Figure 3.6 in the standalone model in Appendix B. The proof is straightforward: we construct a simulator and show that an adversary corrupting the service provider and any number of clients cannot distinguish between the real protocol and interacting with the ideal functionality.

3.6.3 Two-way Sealed Sender Conversations

While the construction above successfully realizes sealed sender conversations, the identity of the receiver is still leaked to the service provider. Ideally, we would like for both users in a conversation to communicate using only ephemeral identities, so that the service provider sees only the flow of messages in a conversation but does not learn either party's long-term identity. However, this again leads to a bootstrapping problem: if both users use fresh, anonymous identities, **how do they exchange this ephemeral contact information while remaining anonymous?**

While heavyweight cryptography (such as PIR or ORAMs) may provide a more robust solution, in this work we focus on scalable solutions that might plausibly be adopted by secure messaging platforms. As such, we present a natural extension of our one-way sealed sender conversation protocol.

After an initiator creates an ephemeral key pair, opens a new mailbox, and sends this to the receiver, the receiver responds by doing the same thing: creating a second ephemeral key pair, opening a second mailbox, and sending this back to the initiator as part of the initial delivery

receipt. After this, **both** the conversation initiator and receiver will have conversation table entries of the form $P_*, \mathsf{pk}_{e1}, \mathsf{pk}_{e2}, \mathsf{sk}_{e2}$, with two different ephemeral keys for sending and receiving messages in the conversation.

This requires minimal changes to the previous protocol. Essentially, the **Initiate** protocol gains another section for the recipient to create their own ephemeral identity, but the **Send**, **Open Connection**, and **Push Message** protocols are identical. In Appendix C we provide the full details of these updated protocols, along with an additional protocol **Change Mailbox** which is used to update an ephemeral key pair for one side of an existing conversation.

Security. We have two security goals for this protocol. First, we require that this protocol is a secure instantiation of a one-way sealed sender conversation, just like the protocol above. This is clear, as the only party whose behavior changes from the protocols in Section 3.6.2 is the initial receiver. Simulating their behavior is easy because that user's identity is already leaked by the ideal functionality. As such, the proof remains nearly identical to that in Appendix B.

Second, we require that the service provider has only one chance to identify the initial receiver. Note that besides the initial messages, all sent messages are only linked to the anonymous identities. Thus, no information about the users' true identities are leaked by these messages. This only source of information about these identities comes from the timing of the mailbox's initial opening, so this is the only chance to identify the initial receiver. As described in our simulations, in a reasonably busy network it is difficult to link two events perfectly. Instead, it requires many epochs of repeated behavior to extract a link. Therefore, giving the service provider only a single chance to de-anonymize the receiver will most likely (**though not provably**) provide two-sided anonymity. To further decrease the chance of a successful attack, the initial receiver can introduce some initial random delay in opening and using a new mailbox.

Obscuring the Conversation Flow. A natural generalization of this approach is to switch mailboxes often throughout a conversation, possibly with **each message**. This may provide further obfuscation, as each mailbox is only used once. While analyzing **how well** this approach would

obscure the conversation flow is difficult, as linking multiple messages together requires the service provider to find a timing correlation between the various mailboxes' activities, it is clear it provides no worse anonymity than the above construction.

3.6.4 Protecting against Denial of Service

Both constructions presented above require users to anonymously register public keys with the service provider. This provides an easy way for attackers to launch a denial of service attack: simply anonymously register massive numbers of public keys. As such, we now turn our attention to bounding the number of ephemeral identities a user can have open, without compromising the required privacy properties.

We build on **anonymous credential** systems, such as [32]. Intuitively, we want each user in the system to be issued a fixed number of anonymous credentials, each of which can be exchanged for the ability to register a new public key. To implement this system, we add two additional protocols to those presented above: **Get signed mailbox key** and **Open a mailbox**.

In **Get signed mailbox key**, a user P_s authenticates to the service provider with their long-term identity pk_s and uses a blind signature scheme to obliviously get a signature σ_{es} over fresh public key pk_{es}. We denote the service provider's keypair $(pk_{\text{sign}}, sk_{\text{sign}})$. In **Open a mailbox**, a user P_s anonymously connects to the service provider and presents $(\mathsf{pk}_{es}, \sigma_{es})$. If σ_{es} is valid and the service provider has never seen the public key pk_{es} before, the service provider opens a mailbox for the public key pk_{es}. These protocols are described below:

Get signed mailbox key

(1) User authenticates using their longterm public key. Server checks that the client has not exceeded their quota of generated ephemeral identities.

(2) Client generates $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSKeyGen}(1^\lambda)$

(3) Client blinds the ephemeral public key $b \leftarrow \Pi_{\text{bs}}.\mathsf{BSBlind}(\mathsf{pk}_e, pk_{\text{sign}}; r)$ with $r \leftarrow \{0,1\}^\lambda$.

(4) Server signs the client's blinded public key with $s_{blind} \leftarrow \Pi_{\text{bs}}.\mathsf{BSSign}(b, sk_{\text{sign}})$ and returns

the blinded signature to the client.

(5) Client extracts the real signature locally with $\sigma_e \leftarrow \Pi_{\text{bs}}.\mathsf{BSExtract}(s_{blind}, pk_{\text{sign}}; r)$

Open a mailbox

(1) Client connects anonymously to the server and sends pk_e, σ_e

(2) Server verifies $\Pi_{\text{bs}}.\mathsf{BSVerify}(pk_{\text{sign}}, \sigma_e) = 1$ and checks pk_e has not been used yet.

(3) Server registers an anonymous mailbox with key pk_e with an expiration date.

Integrating these protocols into one-way sealed sender conversations and two-way sealed sender conversations is straightforward. At the beginning of each time period (**e.g.** a day), users run **Get signed mailbox key** up to k times, where k is an arbitrary constant fixed by the system. Then, whenever a user needs to open a mailbox, they run the **Open a mailbox** protocol. Sending and receiving messages proceeds as before.

It is important that (1) the signing key for the blind signature scheme public key pk_{sign} be updated regularly, and (2) anonymous mailboxes will eventually expire. Without these protections, malicious users **eventually** accumulate enough anonymous credentials or open mailboxes that they can effectively launch the denial of service attack described above. Additionally, each time period's pk_{sign} must be known to all users; otherwise the server could use a unique key to sign each user's credentials, re-identifying the users.

3.6.5 Blind Signature Performance

To test the feasibility of using blind signatures, we implemented the protocols in Section 3.6.4 for a single client and server. This represents the **cryptographic overhead** of applying our solution, as the remainder (sending and receiving messages, registering keys) are services already provided by Signal.

The networking for both the client and server are written in Python, with the Django web framework [2] on the server. Starting with the code provided in [17], we implement an RSA-2048

Table 3.1: Timing results (in seconds) for protocols of Section 3.6.4, using RSA-2048 ciphertexts and ECDSA.

Network Conditions	ECDSA KeyGen	Get Signed Mailbox Key	Open a Mailbox
End-to-End	0.049	0.061	0.039
User Local	0.049	0.032	0.024
Server Local	N/A	0.013	0.001

blind signature [32] library in Java that can be called via RPC. Although RSA ciphertexts are large, they are very fast to compute on modern hardware.

We evaluated our implementation by running the server on an AWS instance with 2 Intel Xeon processors and 4 GB of RAM. The client was running on a consumer-grade laptop, with a 2.5 GHz Intel i7 with 16 GB of RAM, located in the same region as the AWS server. We report the timing results in Table 3.1 for each protocol. To better isolate the overhead from network delay, we also report the execution time when server and client are running locally on the same machine.

Importantly, ECDSA KeyGen can be run in the background of the client, long before the interactive phase of the protocol starts. For maximum security, a user may close an old mailbox and get a new signed key (with the same anonymous connection), and then open a new mailbox with each message that they send. This incurs an overhead of less that 100ms, even including network delay. The communication overhead of running this full protocol is less that 1KB, constituting 3 RSA-2048 ciphertexts and 1 ECDSA public key.

3.6.6 Deployment Considerations

Key Rolling. It is critical that the server maintain a database of ephemeral identities previously registered on the system in order to check for re-use of old ephemeral identities. Note that to prevent reuse, this database must be maintained for as long as the identities are valid and grows with the number of mailboxes, not the number of users.

We suggest that Signal update their mailbox signing key at regular intervals, perhaps each day, and leave two or three keys valid for overlapping periods of time to avoid interruptions in service. Because the validity of a signed mailbox key is tied to the signing key, each update allows

the server to "forget" all the keys that it saw under the old signing keys as they cannot be reused.

Mailbox Opening. It is important that users perform **Get signed mailbox key** (where Signal learns a user's identify) and **Open a mailbox** in an uncorrelated way. Otherwise, Signal could link the two and identify the anonymous mailbox. We recommend performing **Get signed mailbox key** at regular intervals (e.g. the same time each day), but careful consideration must be taken for users that are offline during their usual time. Users should not come online and perform both operations immediately if sending to a new conversation. To avoid this, clients should maintain a small batch of extra signed mailbox keys for new conversations.

Cost Overhead. We analyze the worst case cost of scaling our protocol. We generously assume that 10 million anonymous mailboxes will be opened every day. The server's part of opening these mailboxes constitutes calls to BSVerify and BSSign and a database query (to check for repeated identities). In our experiments, the two blind signature operations, including the Django networking interface, took a cumulative .014 seconds. Using AWS Lambda, supporting 10 million messages each day would cost approximately $10 per month. We estimate that doing 10 million reads and writes a day to a DynamoDB database would cost approximately $20 per month, using AWS's reserved capacity pricing.

Using the key rolling scheme described above, the database contains at most the number of messages delivered in a day times the number of simultaneously valid keys. At 10 million messages each time with a two overlapping valid keys, this means the database would contain at most 20 million ephemeral identities. Assuming 256-bit identity values, the entire database would never exceed a few GB of data. Therefore, we conservatively estimate that the marginal cost of supporting our protocol for 10 million ephemeral identities per day would be under $40 per month. We note our analysis does not consider the personnel cost associated with developing or maintaining this infrastructure. Ideally, this would be amortized along with Signal's existing reliability and support infrastructure.

3.7 Discussion

3.7.1 Other solutions

In this section, we consider alternative, **minor changes** to the existing sealed sender protocol and evaluate their effectiveness.

Random delays. Users could send delivery or read receipts after a random delay, making it harder for attackers to correlate messages. This forces an attacker to increase the **epoch duration** to perform the same attack. We analyze the effect of varying epoch duration in Figure 3.5, and find that even with hour-long epochs—likely rendering delivery receipts useless—users could still be identified within 60 messages. We conclude that injecting random delays is an ineffective way to achieve anonymity.

Cover traffic. Users could send random sealed-sender messages that are transparently ignored by the recipient in order to cover for the true pattern of ongoing conversations. Based on our experiments, we again see that cover traffic slows down our attack, but at a linear rate with the amount of extra traffic: even with 10x extra messages, the anonymity set of potential senders to Bob after 100 messages is under 1000 users. This mitigation strategy has obvious costs for the service provider, without significant benefit to user anonymity.

Disable automatic receipts. While Signal users can disable read receipts and typing notifications, they currently cannot turn off delivery receipts. Adding an option for this would give users the choice to greatly mitigate this attack. We note disabling would have to be mutual: Alice turning off delivery receipts should also prevent Bob from sending them, different from how Signal currently disables read receipts. We also note users could potentially still be linked purely by their messages eventually, making this only a partial mitigation.

3.7.2 Drawbacks and Likelihood of Adoption

We believe that the solution we have proposed in Section 3.6 is both practical and cost-effective. However, there are a few drawbacks. Most importantly, it adds complexity to the system, and complexity always increases the likelihood of error and vulnerability. In particular, the key rolling scheme we suggest in Section 3.6.6 requires increased complexity in the back-end key management system. While the compromise of these keys would not leak message content, it could allow for a cheap resource denial attack on Signal.

A second important drawback of our solution is the assumption that a malicious service provider cannot use network information to identify users. As mentioned, using Tor [56, 11] would address this, but only if enough users did so to increase the anonymity set.

Finally, our ephemeral identities may increase complexity for users that use Signal on multiple devices. Signal would need to securely share or deterministically generate these keys with other devices in a privacy-preserving way.

Given the limited scope and impact of these drawbacks, we believe that is reasonable to believe that Signal or other secure messengers could potentially adopt our solution.

3.7.3 Group messaging

The OTR and Signal protocols were first designed for pairwise communication, and we have focused on such conversations in this work. However, group messaging is an important use case for private messaging services, and has recently shown to be vulnerable to different kinds of attacks [143, 38, 145].

An interesting direction for future work would be to extend our attacks to this setting. It is clear that received receipts and read receipts do not work the same way in groups as they do for two-way conversations. On the other hand, group messages have additional **group management messages** which are automatically triggered, for example, when a new member attempts to join the group. It would be interesting to understand if, for example, our attack could exploit these

message to de-anonymize **all** members of a given group chat.

Fortunately, it does seem that our main solution proposed in Section 3.6 would be applicable to the group chat setting: all members of the group chat would create new, anonymous mailboxes used only for that particular group. However, this would still leave the difficulty of the initial configuration and key management, which would be more complicated than that two-party setting. We consider this to be important and useful potential future work.

3.8 Related Work

Attacks on mobile messaging. Mobile messaging services have been hugely popular for decades, but the SMS protocol was designed primarily for efficiency and not with privacy in mind [78]. Usability studies have shown that many users want or even assume that their text messages are private [74], which has made SMS a "Goldmine to exploit" for state surveillance [19, 63]. Even encrypted alternatives to SMS are still targeted by hackers and state-level surveillance tools, as seen for example by the NSO group's Pegasus spyware, which was used to target the text messages of journalists and politicians in multiple countries [109].

Statistical disclosure attacks. SDAs were first proposed as an attack on mix networks by [47], and later strengthened to cover more realistic scenarios with fewer or different assumptions [111, 108, 49]. More recent variants consider the entire network, and attempt to learn as much as possible about all sender-receiver correlations over a large number of observations [50, 135, 89]. See [129] for a nice overview and comparison of many existing results.

Private messaging. Perhaps in response to these highly-publicized attacks, third-party applications which provide end-to-end encrypted messaging, such as WhatsApp (since 2016), Telegram, and Signal, are rapidly gaining in popularity [97]. A good overview for the interested reader would be the SoK paper of Unger et. al. from 2015 [161].

The first cryptographically sound, scalable system for end-to-end encrypted messaging is the OTR protocol from 2004 [25], which had significant influence on the popular systems used

today [110, 67, 37].

Since OTR, significant research has investigated how to remove or hide metadata to provide anonymous chat applications. Indeed, similar problems have been noted in mix-nets [101]. Many such as Ricochet [27] rely on Tor [56]. Other techniques for obscuring metadata are injecting noise, like Pond [98] and Stadium [160], or decentralization [93]. Many of these solutions require sharing cryptographic identities out-of-band, rather than build off human-friendly or already known identities.

DC-net based messengers like Dissent [42] or Verdict [43] have also been proposed, but suffer problems in scaling to the number of users seen on popular messaging applications [161, 162]. Others such as Riposte [41] have made use of private information retrieval to achieve anonymity, but this is also expensive in practice. We focus on sealed sender in this paper, as it is the most widely-deployed in practice attempt to provide sender anonymity in secure messaging.

3.9 Conclusion

In this work we analyze and improve upon Signal's sealed sender messaging protocol. We first identify a type of statistical disclosure attack (SDA) that would allow Signal to identify who is messaging a user despite sealed sender hiding message sources. We perform a theoretical and simulation-based analysis on this attack, and find that it can work after only a handful of messages have been sent to a user. Our attack is possible because of two features of the sealed sender protocol: (1) metadata (specifically, recipient and timing) is still revealed, and (2) Signal sends automatic delivery receipts back to the sender immediately after a message is received.

We suggest a protection against this attack, in which users anonymously register ephemeral mailbox identities with Signal, and use those to communicate rather than long-term identities such as phone numbers. To prevent abuse, we suggest Signal use anonymous credentials, implemented with blind signatures, and implement a prototype that demonstrates our solution is performant and cost-effective to deploy.

Signal has taken a first step into providing anonymous communication to millions of users

with the sealed sender feature. Signal's design puts practicality first, and as a result, does not provide strong protection against even known disclosure attacks. Nonetheless, we believe this effort can be improved upon without sacrificing practicality, and we hope that our work provides a clear path toward this end.

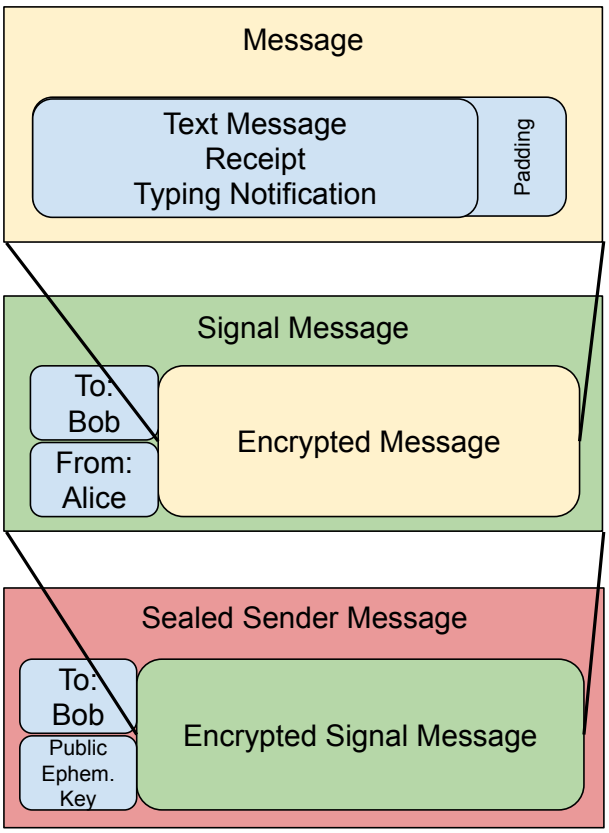

Figure 3.1: **Structure of Signal Messages** — All messages Alice sends to Bob through Signal (receipts, text messages, or events) are first padded to the next multiple of 160 bytes. The padded message is then encrypted under the shared key between Alice and Bob and then combined with 'To: Bob' and 'From: Alice' metadata to form a Signal Message. If both Alice and Bob have sealed sender enabled then Alice will then generate an ECDHE key pair and derive a new shared secret with Bob's public key to encrypt the Signal Message and combine with 'To: Bob' and the public ephemeral key to form a sealed sender message that will be sent to Bob.

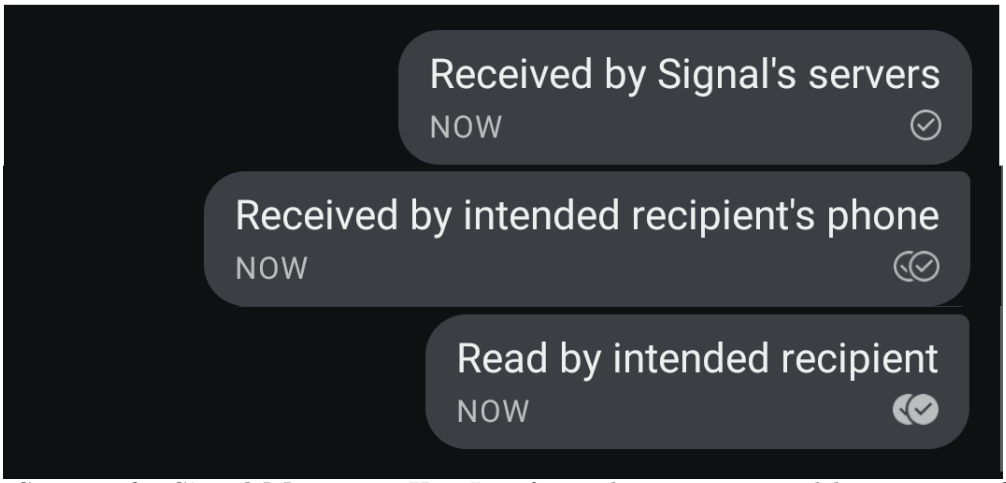

Figure 3.2: **Stages of a Signal Message** — User Interface indicating message delivery status. One hollow check mark signifies that the message is en route. Two hollow check marks signifies the receipt of a delivery receipt for the message. Finally, two filled check mark signifies the receipt of a read receipt for the message.

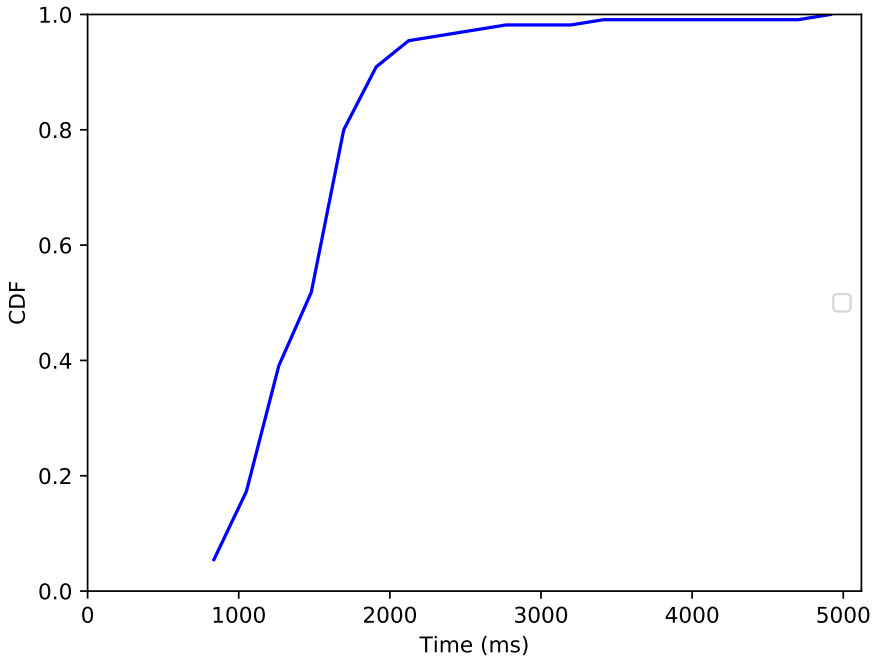

Figure 3.3: **CDF of Delivery Receipt timing** — CDF of time between a device sending a message (to another online device) and receiving a Delivery Receipt. The median time is 1480ms and 90% of Delivery Receipts were received within 1909ms.

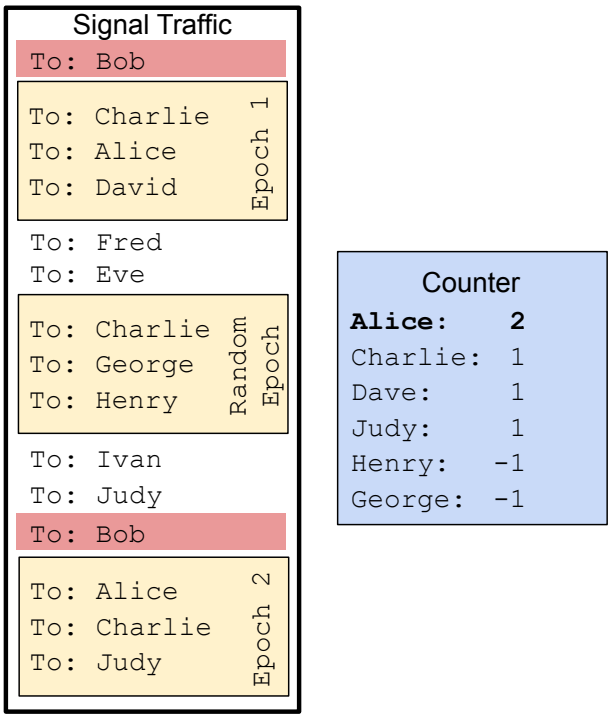

Figure 3.4: **Attack Overview** — Our SDA variant has the service provider (Signal) keep count of all users who receive messages in the *epoch* after Bob receives a message to determine who is consistently messaging at the same time as Bob is receiving a message. Additionally, the service provider will begin an epoch at a random time to keep track of users which are messaging independent of the associates of Bob, and those users will be deducted from the counter. As such, "popular" users such as *Charlie* will not mask Alice's behavior.

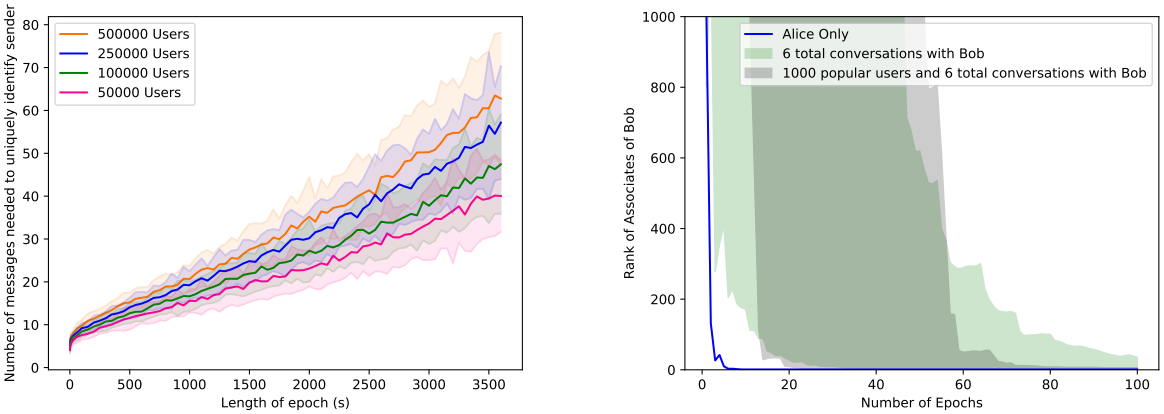

Figure 3.5: **Left: Effect of delayed Read Receipts** — The attack assumes that each epoch lasts one second, and thus the log collects all delivery receipts that are sent within 1 second of Bob receiving a sealed sender message. A possible simple solution to this attack is to delay delivery receipts. We tested the effectiveness of the attack with variably sized epochs and determined that if delivery receipts were delayed a full hour (making them effectively worthless for their purpose) that with a user base of 500,000 users (each sending 50 messages a day) Bob would need to receive 60 messages from the victim user to identify Alice as the sender.
Right: Effect of popular users in our SDA — We examined the effectiveness of our SDA variant by examining the cases where only Alice is messaging Bob and where Bob is being messaged by Alice and 5 other users. The graph shows the rank of those messaging Bob, how many users have received more messages than those messaging Bob. When only Alice is messaging Bob each of the attack epochs are started by her, meaning her rank will very quickly drop. When multiple users are messaging Bob there is a range of ranks, represented by the green band which bounds the lowest ranked user messaging Bob (on the bottom) and the highest ranked individual messaging Bob (on the top). When epochs are begun by multiple users, an individual's rank takes a while to drop. The graph shows that for over 45 epochs one of the users messaging Bob has a rank of over 1000, while another user messaging Bob has dropped to a rank of 0 (meaning they have received a message after Bob received a message the most of any user in the system). The black band considers the same situation, but with 1000 popular users in the system which our variant accounts for.

Ideal Functionality For Sealed Sender Conversation System

- $P_1, \ldots, P_n$: A set of n (possibly corrupt) users of the system
- P_{service}: A single corrupt service provider that is in charge of relaying messages between users
- Active Conversation Table C_{active} with entries of the form $(\texttt{convo-id}, \texttt{initiator}, \texttt{receiver})$, Delivery Pending Message Table M_{pending} with entries of the form $(\texttt{convo-id}, \texttt{sender}, \texttt{receiver}, \texttt{plaintext})$

Start Conversation: Upon receiving a message $(\mathsf{StartConvo}, P_j)$ from a user P_i, the ideal functionality generates a unique identifier `cid`, and performs the following:

- If P_i or P_j is corrupt, send $(\mathsf{ApproveNewConvoCorrupt}, P_i, P_j, \texttt{cid})$ to P_{service}
- If both P_i and P_j are honest, $(\mathsf{ApproveNewConvo}, P_j, \texttt{cid})$ to P_{service}

P_{service} responds to either message with $(\mathsf{Approve})$ or $(\mathsf{Disapprove})$

- If P_{service} responds with $(\mathsf{Disapprove})$, the ideal functionality halts
- If P_{service} responds with $(\mathsf{Approve})$, the ideal functionality sends $(\mathsf{NewConvo}, P_i, P_j, \texttt{cid})$ to both P_i and P_j and adds $(\texttt{cid}, P_i, P_j)$ to C_{active}.

Send Message: Upon receiving a message $(\mathsf{SendMessage}, \texttt{cid}, m)$ from party P_i, the ideal functionality checks the active conversations table C_{active} for an entry $(\texttt{cid}, P_j, P_i)$ or $(\texttt{cid}, P_i, P_j)$. If no such entry exists, the ideal functionality drops the message. The ideal functionality generates a unique identifier `mid` and performs the following:

- If there is an entry and P_j is corrupted, the ideal functionality sends $(\mathsf{NotifySendMessageCorrupt}, \texttt{cid}, \texttt{mid}, m, P_i, P_j)$ to P_{service}, and add $(P_i, P_j, \texttt{cid}, \texttt{mid}, m)$ to M_{pending}.
- If an entry $(\texttt{cid}, P_i, P_j)$ exists, send $(\mathsf{NotifySendMessage}, \texttt{cid}, \texttt{mid}, P_j, |m|)$ to P_{service}, and add $(P_i, P_j, \texttt{cid}, \texttt{mid}, m)$ to M_{pending}.
- If an entry $(\texttt{cid}, P_j, P_i)$ exists, send $(\mathsf{NotifyAnonymousSendMessage}, \texttt{cid}, \texttt{mid}, |m|)$ to P_{service}, and add $(P_i, P_j, \texttt{cid}, \texttt{mid}, m)$ to M_{pending}.

Receive Message: Upon receiving a message $(\mathsf{ReceiveMessage}, \texttt{cid})$ from party P_j, the ideal functionality checks C_{active} for an entry $(\texttt{cid}, P_j, P_i)$ or $(\texttt{cid}, P_i, P_j)$. If such an entry exist, it performs one of the following:

- If P_i is corrupt, the ideal functionality then sends $(\mathsf{ApproveReceiveMessageCorrupt}, \texttt{cid}, P_i, P_j)$ to P_{service}, which responds with tuples of the form $(\texttt{cid}, P_i, P_j, m)$. The ideal functionality then sends $(\mathsf{Sent}, P_i, P_j, \texttt{cid}, m)$ to P_j for each such tuple.
- If there is an entry $(\texttt{cid}, P_j, P_i)$ in C_{active} and entries $(P_i, P_j, \texttt{cid}, \texttt{mid}, m)$ in M_{pending}, the ideal functionality sends $(\mathsf{ApproveAnonymousReceiveMessage}, \texttt{cid}, \texttt{mid}, |m|)$ to P_{service} for each such entry. P_{service} responds to each message with either $(\mathsf{Approve}, \texttt{mid})$ or $(\mathsf{Disapprove}, \texttt{mid})$. If P_{service} responds with $(\mathsf{Approve}, \texttt{mid})$, the ideal functionality sends $(\mathsf{Sent}, P_i, P_j, \texttt{cid}, m)$ to P_j.
- If there is an entry $(\texttt{cid}, P_i, P_j)$ in C_{active} and entries $(P_i, P_j, \texttt{cid}, \texttt{mid}, m)$ in M_{pending}, the ideal functionality sends $(\mathsf{ApproveReceiveMessage}, \texttt{cid}, \texttt{mid}, |m|, P_j)$ to P_{service} for each such entry. P_{service} responds to each message with either $(\mathsf{Approve}, \texttt{mid})$ or $(\mathsf{Disapprove}, \texttt{mid})$. If P_{service} responds with $(\mathsf{Approve}, \texttt{mid})$, the ideal functionality sends $(\mathsf{Sent}, P_i, P_j, \texttt{cid}, m)$ to P_j.

Figure 3.6: Ideal functionality formalizing the leakage to the service provider for a one-way sealed sender conversation.

Chapter 4

Mind the IP Gap: Measuring the impact of IPv6 on DNS censorship

4.1 Introduction

Internet censorship is a global problem that affects over half the world's population. Censors rely on sophisticated network middleboxes to inspect and block traffic. A core component of Internet censorship is DNS blocking, and prior work has extensively studied how DNS censorship occurs, both for specific countries [126, 79] and globally [90, 46, 134, 147]. These studies generally perform active measurements of DNS resolvers for large sets of domains, and identify forged censorship responses from legitimate ones. Unfortunately, this prior work has focused exclusively on the IPv4 Internet, in part because scanning the IPv6 Internet for open resolvers is difficult [121], owing to its impossible-to-enumerate 128-bit address space. In this paper, we perform the first comprehensive global measurement of DNS censorship on the IPv6 Internet. We leverage a recent network measurement technique that can discover dual-stack IPv6 open resolvers from their IPv4 counterpart [77], and use these IPv4-IPv6 resolver pairs to study DNS censorship globally. We then use this data to measure the difference in censorship on IPv4 and IPv6.

IPv6 is becoming more widely deployed, with nearly 35% of current Internet traffic being served over native IPv6 connections [12]. However, IPv6 has fundamentally different performance characteristics [52], network security policies [45], and network topologies [44] compared to the traditional IPv4 Internet. While it may seem that censors either do or don't support detecting and censoring IPv6 DNS in an all-or-nothing fashion, we find that there is a tremendous range of how well a censor blocks in IPv6 compared to IPv4. In particular, although nearly all of the countries

we study have some support for IPv6 censorship, we find that most block less effectively in IPv6 compared to IPv4. For instance, we observe Thailand censors on average 80% fewer IPv6 DNS resources compared to IPv4 ones, despite a robust nation-wide censorship system [68].

Studying censorship in IPv6 can provide opportunities for circumvention tools. By identifying ways that censors miss or incorrectly implement blocking, we can offer these as techniques that tools can exploit. Moreover, because of the complex and heterogeneous censorship systems censors operate, many of these techniques would be costly for censors to prevent, requiring investing significant resources to close the IPv4/IPv6 gap in their networks. For this reason, we believe IPv6 can provide unique techniques for circumvention researchers and tool developers alike, that will be beneficial in the short term and potentially robust in the longer term.

We find a significant global presence of IPv6 DNS censorship — comparable, but not identical to well documented IPv4 censorship efforts. Censors demonstrate a clear bias towards IPv4, censoring `A` queries in IPv4 at the highest rates, and a propensity for censoring native record types (`A` in IPv4, `AAAA` in IPv6). At the country level we break down differences by resolver and domain across resource record and interface type. We find that multiple countries — Thailand, Myanmar, Bangladesh, Pakistan, and Iran — present consistent discrepancies across all resolvers or domains indicating centrally coordinated censorship, where the policies that govern IPv4 and IPv6 censorship are managed centrally. Other countries show more varied discrepancies in the ways that resolvers censor IPv4 and IPv6, due to decentralized models of censorship, such as that in Russia [139], or due to independent and varied corporate network firewalls. We also identify behavior indicative of censorship oversight that can be advantageous to censorship circumvention. For example Brazil and Thailand censor IPv6 queries that rely on 6to4 bridges at lower rates, presumably due to the encapsulation of an IPv6 DNS request in an IPv4 packet, instead of appearing as UDP.

Taken all together, this study provides a first look at IPv6 DNS censorship and the policy gaps that arise from the IPv6 transition. We provide the following contributions:

- We conduct the first large-scale measurement of IPv6 DNS censorship in over 100 IPv6-

connected countries. We find that while most censors support IPv6 in some capacity, there are significant gaps in how well they censor IPv6.

- We provide methodological improvements on measuring DNS censorship that avoids relying on cumbersome IP comparisons (that are not robust to region-specific DNS nameservers). Our methods are easily reproducible, and can be used in future measurement studies.
- We characterize the difference in censorship of both network type (IPv4 and IPv6), and resource type (`A` and `AAAA` record), and identify trends in several countries.
- Using our findings, we suggest several new avenues of future exploration for censorship circumvention researchers, and censorship measurements.

The remainder of this paper is organized as follows. Section 4.2 provides background information in DNS censorship, and the relation of IPv6 to relevant DNS infrastructure. We outline our compiled methodology and ethical design considerations in Section 4.3 before presenting our findings on the global prevalence of IPv6 censorship in Section 4.4. We then dig into per country analysis based on Resource Record types in Section 4.5 and IP protocol version in Section 4.6. We select several case studies to highlight in Section 4.7 before covering related work in Section 4.8. Finally Section 4.9 provides discussion and contextualization of this work before concluding.

4.2 Background

Our work is focused on uncovering differences in **Internet censorship** that occur from **DNS censorship** mechanisms' failure to effectively adapt to the Internet's **adoption of the IPv6 protocol**. In this section, we provide an overview of Internet censorship approaches (Section 4.2.1), DNS censorship mechanisms and infrastructure (Section 4.2.2), and the impact of IPv6 on DNS censorship (Section 4.2.3).

4.2.1 Internet censorship

What constitutes Internet censorship? Internet censorship can be broadly defined as the act of filtering or blocking access to Internet content [6]. Like previous work, we also use this definition when measuring censorship.

Corporate and national censorship. Our working definition of censorship applies to any Internet traffic filtering or blocking actions, regardless of the entities that perform them or their purpose for doing so. With a few exceptions, much of prior work has largely focused on measuring censorship with a presumed attribution of measured events to national content access policies. However, it is important to note that corporate censorship, where businesses deploy information controls systems to prevent access to specific types of content, is also globally prevalent. Throughout our work, we take care to identify when it is likely that our measurements are reporting instances of corporate censorship rather than national censorship. This classification is done using Maxmind's Connection Type database [4] which characterizes IP end-points as 'Corporate', 'Cable/DSL' (associated typically with residential networks), or 'Cellular'. This distinction is important in order to provide context for some of our results. For example, from our study we find that 0.9% of our tests in the United States were censored. Without the additional context that: (1) more that 50% of all our measurement vantage points in the US were placed in corporate networks and (2) these end-points were responsible for the identified censorship events, this finding would be confusing, assumed incorrect, or incorrectly attributed to a national censorship policy.

Internet censorship techniques. Blocking access to Internet content can occur in a variety of ways and at different layers of the networking stack, all of which require the ability to monitor network traffic passing through the censor's borders. The simplest and most common approaches are: (1) IP-based blocking in which a censor maintains blocklists of IP addresses and prevent connections to these IP addresses; (2) DNS manipulation where censors inject (when the censor is a man-in-the-middle) or return (when the censor is the resolver) incorrect responses to domains that are to be censored; and (3) HTTP proxying in which a censor acts as a proxy to clients

within its border with the intention of blocking access to intercepted 'unsuitable' content. While these censorship mechanisms are passive and possible to evade, it is known that countries such as China have deployed comprehensive censorship infrastructure that is capable of active probing, protocol inspection, and incorporates multiple approaches for censorship. In fact, there has been a large body of work to identify censorship techniques globally [134, 125, 147, 156, 65, 133, 140] and specific to individual countries [79, 16, 139, 175, 68, 122]. Unfortunately, the impact of the growing deployment of IPv6 networks has not been previously studied — leaving gaps in our knowledge of how censors are handling the network transition and what opportunities exist for developers of circumvention tools. Our work fills this gap by studying how the transition to IPv6 impacts DNS censorship mechanisms.

4.2.2 DNS censorship

The Domain Name System (DNS) underpins the global internet by providing a mapping from human readable hostnames to routable IP addresses making domain name resolution the first step in almost all connection establishment flows. However, the widely deployed DNS system is implemented as a plaintext protocol allowing on-path eavesdroppers to inspect the hostnames as clients attempt to establish connections and in some cases inject falsified responses to interfere.

Censors have long been known to use DNS injection to block requests, observing DNS requests and injecting false responses for requests to censored domains. The Chinese traffic inspection system, called the Great Firewall (GFW), is documented injecting falsified DNS responses as early as 2002 [40]. This censorship has been shown to be a packet injection from an on-path adversary monitoring for hostnames in DNS queries that match regular expressions [79].

Distributed and centralized DNS censorship. In a centralized DNS censorship mechanism, a single entity manages the 'blocklists' or filter-rules associated with DNS censorship decisions. This is the case in countries such as China where traffic inspection devices are housed at or near border gateways [174]. In contrast, countries such as India and Russia are known to delegate censorship orders to regional ISPs [70] who may choose to implement them either via DNS (typically by recon-

figuring their own resolvers) or other censorship approaches [139, 176, 150]. The latter approach results in different implementations of DNS censorship and the possibility for inconsistencies across different ISPs.

4.2.3 IPv6

IPv6 6to4 bridges. As more IPv6-only network connected devices emerge, there is an increasing need for technologies that facilitate communication betweeb IPv4 and IPv6 end-points. One such technology is 6to4 bridging. The 6to4 protocol was originally designed as a mechanism to bridge the IPv6 deployment gap as public 6to4 bridges would provide interoperability to legacy systems such that IPv4 services could communicate with IPv6 networks and vice-versa. The protocol works by assigning a `/48` subnet with the encoded local address to an interface then allowing that interface to receive traffic encapsulated in an IPv4 header at the local address. In this way clients without outgoing connection to an IPv6 capable network can send IPv6 packets over to an IPv6 capable host who forwards them onward. In our study, we come across many vantage points which are IPv6 connected via a 6to4 bridge. These are easily identifiable since their IPv6 addresses are drawn from the `2002::/16` address block [30]. In the context of DNS, sending a DNS request over IPv6 to `2002:0102:0304::` will send an `[IPv6|UDP|DNS]` packet over IPv6 to the 6to4 gateway at that address which will then encapsulate the packet in IPv4 to 1.2.3.4 (whose 32-bit address is encoded in the IPv6 address `0102:0304`) in order to transit a portion of the network with no connectivity. For this portion of the journey the packet is `[IPv4|IPv6|UDP|DNS]` until it reaches the destination or another network capable of routing IPv6 packets.

The impact of IPv6 on DNS censorship. As with other critical protocols, DNS has also adapted to the IPv6 protocol. The `AAAA` resource record type was introduced to aid in the resolution of domains to their IPv6 addresses. Further, the existing DNS protocol is IP-independent and can therefore be deployed on IPv4 and IPv6 networks. Therefore, it is now possible and common for IPv4 and IPv6-hosted DNS servers to receive both `A` and `AAAA` queries. This is in contrast to a IPv4-

dominant Internet where `A` queries to IPv4 resolvers were the norm. The changes outlined above also influence the mechanics and success of DNS censorship operations. A theoretically comprehensive DNS censorship strategy using response injection requires traffic monitoring infrastructure to: (1) analyze both IPv4 and IPv6 traffic and (2) parse both `A` and `AAAA` queries accounting for hostnames that may not implement resource records of one type or the other. Our work provides a snapshot documentation of contemporary censorship strategies through the IPv6 transition.

4.3 Dataset and Methodology

The analysis presented in the remainder of this paper is based on the results of 22.4M DNS `A` and `AAAA` resolution requests for 714 domains sent to 7,843 IPv4- and IPv6-capable resolvers located in 128 countries. In this section, we explain our process for identifying resolver targets for our queries (Section 4.3.1), domains that are the subject of our queries (Section 4.3.2), and our process for identifying the occurrence of a censorship event from the results of each query (Section 4.3.3).

4.3.1 Selecting resolvers

Our work is aimed at characterizing the inconsistencies that exist in the handling of IPv4- and IPv6-related DNS queries — i.e., differences in the handling of `A` and `AAAA` query types over IPv4 and IPv6 connections. Therefore, we required that each resolver used for our measurements was IPv4- and IPv6-capable.

Identifying resolvers with IPv4- and IPv6-capabilities. Our approach, builds on the work of Hendricks et al. [77] who identified IPv6 open resolvers to measure the potential for IPv6-based DDoS attacks.

IPv6-only domain. We begin by creating a new domain, owned and controlled by us, with the name server set to an IPv6-only record. In other words, to query this domain (which we call the IPv6-only NS domain), one must contact the IPv6-only name server.

Identifying IPv4-capable resolvers. We use `zmap` [59] to scan the entire IPv4 address space and issue a DNS `A` query on port 53 for a separate IPv4 control domain we also control. This yields an initial list of 7.2M IPv4 DNS resolvers.

Verifying IPv6 capabilities of IPv4-capable resolvers. Next, we issue a DNS `A` query for a resolver-specific subdomain of our IPv6-only NS domain to each of these IPv4 DNS resolvers. The subdomain encodes the IPv4 address of the resolver being targeted. Therefore, if our domain was `v6onlyNS.io` and our resolver target was `1.1.1.1`, our DNS query requested the `A` record for `1-1-1-1.v6onlyNS.io`. Since IPv4-only resolvers will not be able to communicate with our IPv6-only Name Server, we expect this resolution to fail for IPv4-only resolvers. On the other hand, resolvers with any form of IPv6 connectivity will be able to connect to our IPv6-only Name Server. Thus, by examining the logs of our IPv6-only name server, we are able to identify the set of resolvers that successfully reached our server and their corresponding IPv4 addresses. The associated IPv6 address for each successful query is extracted from packet captures of the IPv6-only name server giving us 33.7K (IPv4, IPv6) address matchings.

Filtering and geolocating resolvers. The approach detailed above yields matchings that suggest the presence of 'infrastructure' resolvers — e.g., multiple IPv4 resolvers have the same IPv6 address associated with them. These are cases where the IPv4 resolver simply forwards requests to a dedicated multi-machine DNS infrastructure rather than performing the resolution by itself. Although this does not change the validity of our results regarding the IPv6-related inconsistencies of resolvers, we still remove these cases in order to minimize the influence of such infrastructure resolvers. This yields 14.9K resolver pairs. Finally, to confirm the correctness of our list of IPv4/IPv6 resolver pairs, we: (1) use the Maxmind GeoIP dataset [4] to geolocate the IPv4 and IPv6 addresses of a resolver pair and keep those which belong to the same region which reduces us to 14.2K resolver pairs, (2) perform a follow up scan by issuing `A` and `AAAA` requests for both our control domains using `zdns` [59] and filter out those pairs where an incorrect response was received.

In total, we obtained 7,843 resolver pairs across 106 different countries. Table D.1 illustrates the geographic distribution of the resolvers.

4.3.2 Selecting target domains

In order to identify inconsistencies in IPv4/IPv6-related DNS censorship we require that the domains we use are: (1) sensitive and likely to be censored in a large number of countries and (2) have valid `A` and `AAAA` records associated with them.

Identifying sensitive domains. Censored Planet's 'Satellite' project, which performs global longitudinal measurements of DNS censorship, [156] maintains a list of domains that combines sensitive domains in each country with popular domains randomly chosen from the Alexa Top-10K. The sensitive domains in this list were gathered by the Citizen Lab using regional experts to curate lists for each country [96]. Given the input from experts and the availability of comparable validation data from Censored Planet, we utilize this list as the starting point for our study as well. We start with 2506 sensitive domains.

Identifying usable domains. Not all the domains on the Satellite list are usable in our study since they do not have IPv6 connectivity or `AAAA` records. We filter out unusable domains by using `zdns` to perform `A` and `AAAA` resource record requests from Google and Cloudflare's four public DNS resolvers (`8.8.8.8`, `8.8.4.4`, `1.1.1.1`, and `1.0.0.1`) and removing those with invalid `A` or `AAAA` records. A record is invalid if it does not contain valid resource appropriate IPv4 or IPv6 addresses. This gives us with 775 sensitive and infrastructure appropriate domains. Finally, we follow-up by making TLS connections (using `zgrab2`), from our uncensored vantage point, to each of the IP addresses contained in the DNS responses. We locally verify the obtained TLS certificates and exclude all domains whose verification fails. This final list contains 714 sensitive domains whose TLS certificates and IPv4 and IPv6 addresses are valid. We use this list for all the measurements reported in this paper.

4.3.3 Identifying DNS censorship events

In general, our goal is to err on the side of caution and avoid false-positives in our censorship determination. We achieve this by accounting for unreliability of resolvers and domains in our lists.

Removing unstable resolvers. For each of the 7,843 (IPv4, IPv6) resolver pairs and 714 sensitive domains, we send a DNS `A` and `AAAA` resource record request for a domain to the IPv4 and IPv6 addresses associated with the resolver. We follow this up with a `A` and `AAAA` resource record request for a set of 3 control domains (owned and operated by us). We discard data from pairs which failed to resolve any one of our control domains correctly since this is a sign of resolver instability. This left us with 7,441 stable resolver pairs.

Distinguishing censorship from domain instability. For the remaining resolvers, we extract the IPv4 and IPv6 resource records returned for each domain — even those that arrive from multiple responses to a single query (a sign of an on-path censor). Next, we use `zgrab2` to establish TLS connections to the IP addresses contained in the resource records. In each of these connections, we set the TLS SNI to be the domain whose records were requested. We then locally verify the validity of the retrieved TLS certificates. Since the domains themselves might be unreliable, we repeat this verification procedure three times. Only if this step fails all three times do we conclude that the IP we obtained from that resolver for that domain was censored.

4.3.4 Ethics

Our experimental design has incorporated ethical considerations into the decision-making process at multiple stages. Censorship measurement has inherent risks and trade-offs: better understanding of censorship can help support and inform users, but specific measurements may carry risk to participants or network users. We rely on The Menlo Report [57], its companion guide [18], and the censorship specific ethical measurement guidelines discussed by Jones et al. [87] to carefully weigh these trade-offs in our experimental design.

Consent. To align with the guiding principle of *respect for persons* we structure the data collection to implicate as few individuals as possible. Specifically we rely on open resolvers which typically have little or no direct association with individuals in lieu of measurement from client based software. While we cannot acquire direct or proxy consent from the operators of the open resolvers we consider the trade-off between the implied consent standard and the value in the measurements we make. We note that the goal of our ethical analysis is not to eliminate risk, but to minimize it wherever possible. As noted by Jones et al. in some cases acquiring consent from operators may not only be impossible, but could increase the risk to operators as it introduces their acknowledgement of, or active participation in, the measurement at hand [87]. This analysis aligns with previous work relying on open resolvers to collect impactful results while minimizing risk on individuals [134, 147, 156].

Privacy. Our study collects no personal data about any end users or open resolver operators. The analysis completed herein uses resolver addresses, public Anonymous System (AS) identifiers, and country codes. All measurements are initiated from within the United States. Beyond this, measurement domains are not drawn from any human browsing patterns or history as the suspected censored domains are a subset of the Satellite measurement results (**cf.,** Section 4.3.2).

Resource usage. The vantage that was used for data collection is connected to the internet with a 1 Gbps interface that scanned using the default rates for `zmap` and `zgrab2` tools (line rate). However, the structure of the scan was established such that individual resolvers and domains for the DNS probe and TLS certificate validation respectively would be accessed in round robin order — i.e., when probing the open resolvers every target would receive a first request before any target would receive the subsequent request. Equivalently for validating TLS certificates, each of the 714 target domains would receive a first attempted handshake before any target would receive a subsequent handshake, limiting the bandwidth any one host will receive.

Table 4.1: Top-25 countries with over 25 resolver pairs and the highest average rates of DNS censorship across both query types and network interfaces. Rates are expressed as the percentage of censored DNS queries. Darker shaded cells indicate a higher rate of DNS censorship (compared to the country's average). The average rate of censorship for a country is computed across all four IP/query combinations. The global row contains the mean of each column and includes data from the countries with less than 25 resolvers. These means weigh the contribution of each country equally, rather than weighted by the number of resolvers used in tests.

Country	Resolver pairs	IPv4 A	IPv4 AAAA	IPv6 A	IPv6 AAAA	Avg.
China (CN)	194	29.27	32.32	28.41	32.08	30.52
Iran (IR)	277	25.12	24.49	21.95	21.45	23.25
Hong Kong (HK)	67	7.00	5.57	4.91	5.24	5.68
Russia (RU)	312	5.51	4.78	4.49	4.50	4.82
Ukraine (UA)	35	6.49	3.05	2.64	2.87	3.76
Indonesia (ID)	56	6.46	2.99	2.41	2.26	3.53
Argentina (AR)	47	5.20	3.51	2.22	1.28	3.05
Thailand (TH)	186	8.25	1.18	1.13	0.93	2.87
Malaysia (MY)	50	4.92	2.03	1.27	1.35	2.39
Mexico (MX)	150	3.78	2.05	1.75	1.94	2.38
Bangladesh (BD)	29	6.42	1.28	0.89	0.81	2.35
Colombia (CO)	27	3.39	3.28	1.45	1.14	2.31
Italy (IT)	38	2.36	2.10	2.25	2.52	2.31
Brazil (BR)	160	2.67	1.67	1.99	2.08	2.10
Bulgaria (BG)	30	3.24	2.91	1.15	1.06	2.09
Poland (PL)	48	2.00	1.13	2.90	0.96	1.75
South Africa (ZA)	93	2.41	1.60	1.18	1.33	1.63
Korea (KR)	632	2.33	1.06	1.24	1.22	1.46
Chile (CL)	65	2.67	0.70	1.10	0.79	1.32
Romania (RO)	44	2.30	1.03	0.83	0.93	1.27
Spain (ES)	49	0.91	0.75	1.40	1.94	1.25
India (IN)	226	1.27	1.52	1.04	1.16	1.25
Belgium (BE)	31	1.56	1.26	0.95	0.95	1.18
Turkey (TR)	114	1.29	0.96	1.27	1.02	1.14
Viet Nam (VN)	252	1.53	0.89	1.42	0.67	1.13
Global	7,428	3.10	1.83	1.77	1.60	2.07

4.4 Prevalence of DNS Censorship

Overview. In this section, we focus on **providing a high-level understanding on the prevalence of DNS censorship on IPv4 and IPv6 networks**. Specifically, we measure the global prevalence of DNS censorship that occur in the following four cases: (1) a DNS `A` query is sent over IPv4, (2) a DNS `AAAA` query is sent over IPv4, (3) a DNS `A` query is sent over IPv6, and (4) a DNS `AAAA` query is sent over IPv6. While much of prior work has focused on case (1), the increased adoption of IPv6 necessitates the analysis of cases (2-4) which are provided in our work. In each case, we use our collected dataset (**cf.,** Section 4.3) to summarize the base rate of censorship.

How common is DNS censorship? Our data shows the mean base rate of DNS censorship among the 106 countries included in our study, across all query and network types, for our list of domains is 2.1%. A snippet of the base rates observed in each of our four `A/AAAA`-IPv4/IPv6 combinations for the 25 countries which have at least 25 pairs of resolvers that were tested and perform the most censorship is illustrated in Table 4.1. A full description along with breakdowns by

network connection types is provided in the Appendix (Table D.1 and Table E.1). In general, our results concur with prior work which has also found high levels of DNS censorship in China, Iran, Russia, and Hong Kong. As one might expect, when we break down our results by censorship in corporate and non-corporate networks (Table E.1), we see that the presence of countries including Spain, Australia, Korea, and Japan are primarily due to corporate censorship.

Trends in censorship of IPv6-related queries in heavily censoring countries. By measuring IPv6-related behaviors of censorship mechanisms, we uncover a large number of DNS censorship inconsistencies in heavily censoring countries. Because our dataset is balanced (i.e., all the domains have both `A` and `AAAA` records and all the tested resolvers have an IPv4 and IPv6 interface), if censorship is independent of the query type and interface, we expect to see a uniform rate of censorship across all query types and interface combinations in Table 4.1. However, we find that this is not true. We observe two trends common to many of the countries performing the most censorship. First, we see that, in comparison to any other query-interface combination, **`A` queries sent over IPv4 are the most heavily censored**. Second, we see that **`AAAA` queries are censored less than `A` queries, regardless of whether they are sent over IPv4 and IPv6 networks**. This immediately suggests that censorship apparatus in a large number of censoring regions are not fully IPv6-capable. Besides the possibility of misconfiguration of the DNS censorship mechanisms, this may also be because censors in these regions are not yet widely deployed on IPv6 networks in their country. The only exception to both these generalizations is China — the most censoring country in our data. China shows an unusual **preference towards blocking `AAAA` records regardless of whether they are sent over IPv4 or IPv6**. We explore China and several other countries with interesting patterns as specific case studies in Section 4.7.

4.5 Censorship of Resource Records

Overview. In this section, we focus on *identifying and characterizing differences in the handling of IPv4 and IPv6 resource records* in DNS censorship deployments. Specifically, we seek to answer the

following questions: (Section 4.5.1) In which countries is the censorship of IPv4 **resource records** (DNS `A` queries) significantly different than the censorship of IPv6 resource records (DNS `AAAA` queries)?, (Section 4.5.2) what are the characteristics of the **resolvers** which exhibit differences in the handling of `A` and `AAAA` queries?, and (Section 4.5.3) what are the characteristics of **domains** in which these differences are frequently observed?

4.5.1 `A` vs. `AAAA` resource censorship

We use the responses received from our `A` and `AAAA` queries sent to the same set of resolvers and for the same set of domains (**cf.,** Section 4.3 for data collection methodology). We then apply the censorship determination methods described in Section 4.3.3 to measure the prevalence of censorship on our `A` and `AAAA` DNS queries. Finally, we perform statistical tests to identify significant differences in the prevalence of censorship of `A` and `AAAA` queries within each country.

Identifying differences within a country. To measure differences in DNS query handling within a specific country, we compare the prevalence of censorship on `A` and `AAAA` queries by aggregating responses across each resolver within the country. This presents us with two distributions (one each for the group of `A` and `AAAA` queries) of the fraction of censored domains observed at each resolver in the country. We use a two-sample t-test to verify statistical significance of any observed differences between the two groups for each country. In our statistical analysis, we aim to achieve a significance level of 5% ($p \leq .05$) **over all our findings**. Therefore, we apply a Šidák correction [13] to control for Type I (false-positive) errors from multiple hypothesis testing. This requires $p \leq 1 - .05^{1/n_c}$ for classifying a difference as significant, where n_c is the total number of countries in our dataset (106). This approach reduces the likelihood of false-positive reports of within-country differences. The presence of a statistically significant difference for a specific country would imply that `A` and `AAAA` resource types appear to undergo different censorship mechanisms within that country (if a centralized mechanism for censorship exists) or that a significant number of resolvers within that country have inconsistencies in their censoring of each query type. A summary of our results are presented in Table 4.2.

Table 4.2: Differences in blocking rates of `A` and `AAAA` queries observed over IPv4, IPv6, and all resolvers in a country. 'pp' denotes the change in terms of percentage points (computed as `AAAA` blocking rate - `A` blocking rate) and the %age value denotes the percentage change in blocking rate (computed as $100 \times \frac{\texttt{AAAA}\text{ blocking rate} - \texttt{A}\text{ blocking rate}}{\texttt{A}\text{ blocking rate}}$). Only countries having a statistically significant difference are reported. A negative value indicates that `A` queries observed higher blocking rates than `AAAA` queries in a given country. *ns* indicates the difference was not statistically significant and thus omitted. The United States and Myanmar are included separately as they show significance between IPv4 and IPv6 censorship (Section 4.6).

Country	IPv4 resolvers	IPv6 resolvers	All resolvers
Thailand (TH)	-7.1 pp (-85.7%)	*ns*	-3.7 pp (-77.5%)
Bangladesh (BD)	-5.1 pp (-80.0%)	*ns*	-2.6 pp (-71.3%)
Pakistan (PK)	-2.1 pp (-73.6%)	-2.8 pp (-59.8%)	-2.5 pp (-60.2%)
Chile (CL)	-2.0 pp (-57.6%)	*ns*	-1.1 pp (-58.9%)
Vietnam (VN)	*ns*	-0.7 pp (-52.5%)	-0.7 pp (-47.1%)
Korea (KR)	-1.3 pp (-54.6%)	*ns*	-0.6 pp (-36.2%)
China (CN)	3.1 pp (+10.4%)	3.7 pp (+12.9%)	3.4 pp (+11.7%)
United States (US)	-0.5 pp (-33.2%)	0.4 pp (+72.3%)	*ns*
Myanmar (MY)	-2.9 pp (-58.9%)	*ns*	*ns*

How many countries demonstrate large-scale inconsistencies in their handling of `A` and `AAAA` queries? In total, only seven countries showed a statistically significant difference in the rate at which `A` and `AAAA` DNS requests were blocked. We note that this is a conservative lower-bound due to the statistical test used, which minimizes false positive errors at the expense of false negatives. This finding suggests the presence of independent censorship mechanisms for handling each query type in the seven countries (Thailand, Bangladesh, Pakistan, Chile, Vietnam, Korea, and China). Of these, six (China being the only exception) were found to have lower blocking rates for `AAAA` queries than `A` queries. In fact, the `AAAA` censorship rates were between 36-78% lower than the `A` censorship rate suggesting that their censorship mechanisms for `AAAA` queries that are associated with IPv6 connectivity are still lagging. Further analysis shows that the differences are mostly found on the IPv4 interfaces of our resolvers (**cf.,** *IPv4 resolvers* column in Table 4.2) where the `AAAA` censorship rates were up to 86% lower than the `A` censorship rates. This finding is indicative of a tendency for network operators to have focused efforts on maintaining infrastructure for censoring `A` queries sent to IPv4 resolvers, while paying less attention to the handling of `AAAA` queries and their IPv6 interfaces. It also presents an opportunity for circumvention tool developers to exploit. China presents the only exception with a preference for blocking `AAAA` queries on both IPv4 and IPv6 interfaces of resolvers with a 10% and 13% higher `AAAA` censorship rate, respectively. We investigate this anomaly in Section 4.7.

4.5.2 A/AAAA-inconsistent resolvers

Given our above results which suggest that there are a number of countries in which A and AAAA queries are censored differently, we now seek to understand the characteristics of the resolvers that cause these differences. We first focus on identifying the individual resolvers in each country that have statistically different behaviors for A and AAAA queries. Then, we compare the AS distributions of these resolvers with the set of all resolvers in a country. This comparison tells us if the A/AAAA inconsistencies are specific to a subset of ISPs, or if the inconsistencies exist across the whole country (suggesting centralized censorship). Finally, we identify the types of networks hosting inconsistent resolvers to get a measure of whether users in residential networks may exploit these DNS inconsistencies for circumvention.

Identifying differences in individual resolvers. We begin our analysis by identifying the individual resolver pairs (i.e., we consider the IPv4- and IPv6-interfaces of a resolver as a unit), within each of the seven countries listed above that have a statistically significant difference in their censorship of A and AAAA queries. To measure differences in DNS query handling of individual resolvers, we compare the ratio of censored responses each resolver observes for A and AAAA queries.

We use a two-proportion z-test to verify the statistical significance of any observed difference in the ratios between the two groups for each resolver. Similar to our within-country analysis, we apply a Šidák correction to account for our testing of multiple hypotheses and use $p \leq 1 - .05^{1/n_{r_c}}$ to classify a difference as significant, where n_{r_c} is the total number of resolvers in our dataset belonging to country c. A summary of our results is provided in Table 4.3.

Which countries have the largest fractions of resolvers exhibiting A and AAAA resolution inconsistencies? Immediately standing out from the other countries are Thailand, Bangladesh, and Pakistan. These countries have A and AAAA inconsistencies in 62-82% of their resolvers. In comparison, other countries with statistically significant differences have inconsistencies arising from anywhere between 3-30% of their resolvers.

How spread out are the A/AAAA-inconsistent resolvers? We calculate the entropy of the

distribution of censorship by record query type of all resolvers in the country ($S^{\text{all}}_{\text{query}}$) and compare it with the entropy of the distribution of censorship by record query type of the inconsistent resolvers in the country ($S^{\text{inconsistent}}_{\text{query}}$). This serves as a measure of the diversity of ASes observed in both cases. In order to compare the two measures, we use the Kullback-Leibler divergence (∇_{query}) distance [91]. In simple terms, the KL-divergence between two distributions (X, Y) measures the number of additional bits required to encode X given the optimal encoding for Y. In other words, it is the relative entropy of one distribution given another. This computation is helpful for hypothesizing the censorship infrastructure that causes the inconsistencies. Finding a small ∇_{query} value in a country signifies that the inconsistent resolvers had a similar distribution to all the resolvers in that country. This would suggest the presence of a centralized mechanism that (roughly) equally impacts all ASes in the country that is responsible for the inconsistencies. Conversely, a higher ∇_{query} value indicates that there is a strong change in the distribution of resolvers – i.e., a disproportionate number of inconsistencies arise from a smaller set of ASes. This would be indicative of local configuration inconsistency (at the network or resolver level), rather than a centralized configuration inconsistency. We note that this does not provide a measure of how centralized censorship is within a country overall, but rather, it describes how uniform the `A` vs. `AAAA` inconsistencies are.

Based on this analysis, we once again see that Thailand, Bangladesh, and Pakistan stand out with small ∇_{query} values (0.14 - 0.48). This suggests a country-wide censorship mechanism is responsible for the inconsistent censorship of `A` and `AAAA` that impacts all ASes nearly equally. Korea, China, and the United States on the other hand demonstrate high ∇_{query} scores suggesting the presence of network- or resolver-level misconfigurations are responsible. This is confirmed by inspecting the ASes hosting the resolvers with inconsistencies. For example, in the United States, resolvers in just 5 ASes (of 249 ASes with resolvers) account for 56% of all `A` and `AAAA` inconsistencies.

What types of networks exhibit the most `A` and `AAAA` inconsistencies? We use the

Table 4.3: Characteristics of the resolvers which demonstrated a statistically significant difference in their handling of `A` and `AAAA` queries in each country. 'AS diversity' denotes the entropies of (all) resolver distribution ($S^{\text{all}}_{\text{query}}$) and `A/AAAA`-inconsistent resolver distribution ($S^{\text{inconsistent}}_{\text{query}}$) across a country's ASes, and '∇_{query}' represents the Kullback-Leibler divergence of the distribution of inconsistent resolvers from the distribution of all resolvers in the country's ASes (**cf.,** Section 4.5.2). 'Most inconsistent type' denotes the connection type with the most number of `A/AAAA`-inconsistent resolvers.The United States and Myanmar are included separately as they show significance between IPv4 and IPv6 censorship (Section 4.6).

Country	**Total pairs**	**Inconsistent pairs** (% of total pairs)	**Most inconsistent AS** (# inconsistent pairs)	**AS diversity** $S^{\text{all}}_{\text{query}}$	$S^{\text{inconsistent}}_{\text{query}}$	∇_{query}	**Most inconsistent type** (# inconsistent pairs)
Thailand (TH)	186	152 (81.7%)	AS9835 Government IT Services (40)	4.50	4.06	0.14	Cable/DSL (110)
Bangladesh (BD)	29	18 (62.1%)	AS 9230 Bangladesh Online (4)	4.10	3.61	0.48	Cable/DSL (18)
Pakistan (PK)	23	15 (65.2%)	AS 17911 Brain Telecom (3)	3.43	3.06	0.25	Cable/DSL (12)
Chile (CL)	65	20 (30.1%)	AS 27651 Entel Chile (13)	3.08	1.14	1.18	Corporate (13)
Vietnam (VN)	252	64 (25.4%)	AS 131353 NhanHoa Software (37)	3.89	2.22	0.71	Cable/DSL (59)
Korea (KR)	632	80 (12.7%)	AS 9848 Sejong Telecom (13)	3.12	4.17	1.30	Cable/DSL (58)
China (CN)	194	6 (3.1%)	AS 4538 China Education and Research Network (2)	3.89	2.25	2.56	Corporate (4)
United States (US)	1,228	175 (14.3%)	AS 30475 WEHOSTWEBSITES (35)	6.28	4.69	1.31	Corporate (129)
Myanmar (MY)	50	30 (60%)	AS 136170 Exabytes Network (10)	3.31	2.42	0.48	Corporate (26)

Maxmind GeoIP2 connection type database (retrieved in 01/2022 [4]) to identify the connection type of the resolvers responsible for `A` and `AAAA` inconsistencies. We find that, in most countries, Cable/DSL network connections (typically associated with residential networks) were most likely to host a resolver exhibiting an inconsistency. Of the seven countries with statistically significant overall differences, only Thailand and China were found to have a high ratio of `A/AAAA` -inconsistent resolvers in corporate networks. Combined with our previous results which suggest the presence of an inconsistency in a centralized mechanism in Thailand, Bangladesh, and Pakistan, these results show that these inconsistencies are likely extending to residential networks — a promising sign for the citizen users of circumvention tools which exploit the `A/AAAA` gap.

4.5.3 Characteristics of anomalous domains

We now identify the category of domains that appear to exist in the gap of `A` and `AAAA` blocking. In addition to providing a general categorization of these domains, we also analyze whether their category distributions vary significantly from the category distribution of websites that received any blocking. We do this in order to identify the specific policies or mechanisms that differ between the censorship mechanisms for `A` and `AAAA` queries.

Identifying differences in domain behaviors within a country. We continue using our statistical approach for identifying differences within a country. We measure the ratio of blocking that occurs for a domain's `A` and `AAAA` records within the country and then compare these using a

two-proportion z-test with a Šidák corrected p-value of $1 - .05^{dom}$ where dom is the total number of tested domains (714). We only label the behavior of a censor with regards to a domain as different over `A` and `AAAA` queries if the z-test finds the difference to be statistically significant. Once again, we do this to err on the side of caution in order to minimize over-reporting and false-positives of censorship and, in this case, its corresponding policy differences over `A` and `AAAA`.

Do these `A/AAAA`-inconsistent domains hint at policy gaps? For each country, we begin our analysis by deriving domain categories, using the McAfee domain categorization service [8], for domains in the following two lists: (1) $\mathrm{D}_{\mathrm{any}}$ which contains the domains which experienced any blocking events inside a country and (2) $\mathrm{D}_{\mathrm{inconsistent}}$ which contains the `A/AAAA`-inconsistent domains identified by our z-test. Next, we compute the KL-divergence between the category distributions of the two lists (i.e., $\nabla^{\mathrm{domains}}_{\mathrm{query}} = \mathrm{KLDivergence}(\mathrm{D}_{\mathrm{any}}, \mathrm{D}_{\mathrm{inconsistent}})$). A small $\nabla^{\mathrm{domains}}_{\mathrm{query}}$ would signify that the domains that experience inconsistent treatment are not from a largely different category distribution than the set of all domains that experience any type of blocking. This would suggest that the inconsistencies do not arise from a content-specific policy gap that exists in the censorship mechanism implemented over `A` and `AAAA` queries. On the other hand, a large difference would signify that the category distributions are very different and that domains with specific types of content appear to have more inconsistencies — suggesting a content-based policy gap.

Based on this analysis, we find that the United States and Thailand have the smallest $\nabla^{\mathrm{domains}}_{\mathrm{query}}$ scores (0.4-1.2). Conversely, Pakistan and Myanmar have high $\nabla^{\mathrm{domains}}_{\mathrm{query}}$ scores (>2). After manual inspection of these results, we attribute the low divergence in the United States to the fact that censorship observed in the US arise largely from a range of corporate networks (Section 4.4). These resolvers are fairly consistent in their blocking of domains belonging to McAfee's 'P2P/File Sharing' ,'Malicious Sites' and 'Technical/Business Forums' categories. Of the most blocked individual domains, we see domains related to cryptocurrency (which were categorized under 'Technical/Business Forums'), 'netflix.com' and 'utorrent.com'. On the other hand, Thailand shows low divergence despite the large number of residential networks. This suggests that there is no significant policy

gap that causes the `A/AAAA`-inconsistencies. Rather, it suggests an incomplete implementation of an existing mechanisms for `A` query censorship. On the other hand, in Pakistan and Myanmar where the divergence scores were high, we found that the biggest contribution to the high divergence scores arose from the 'Pornography' and 'Government/Military' domain categories, respectively. This suggests the presence of a content-policy gap in the implementation of `A/AAAA` DNS censorship implementations that disproportionately allows sites in these categories to evade censorship.

4.6 Censorship of IPv4 and IPv6 Resolvers

Overview. In this section, we focus on the difference in censorship for DNS queries sent to the **IPv4 and IPv6 interfaes of resolvers**. Specifically, we answer the following questions: (Section 4.6.1) In which countries are the DNS censorship mechanisms for IPv4 and IPv6 traffic significantly different?, (Section 4.6.2) what are the characteristics of resolvers that exhibit differences in the censorship of IPv4 and IPv6 traffic?, and (Section 4.6.3) what are the characteristics of the domains in which such differences are frequently exhibited?

4.6.1 IPv4 vs IPv6 infrastructure censorship

To measure differences in censorship of DNS queries sent over IPv4 and IPv6 we examine distributions similar to Section 4.5.1 but this time investigate two distributions (corresponding to the IPv4 and IPv6 interfaces of resolvers) of the fraction of censored domains from resolvers within the corresponding country. We again use a two-sample t-test to with Šidák correction ($p \leq 1 - .05^{1/n_c}$, where n_c is the total number of countries in our dataset (106)) to verify statistical significance of any observed differences. The presence of a statistically significant difference for a specific country would imply that the country appears to have different censorship mechanisms for IPv4 and IPv6 DNS traffic (if a centralized mechanism for censorship exists) or that a significant number of resolvers within that country are not consistent in their censorship of IPv6 and IPv4 traffic. A summary of our results are presented in Table 4.4.

Table 4.4: Differences in blocking rates of DNS queries sent to IPv4 and IPv6 interfaces of each resolver in a country. 'pp' denotes the change in terms of percentage points (computed as blocking rate of IPv6 - blocking rate of IPv4) and the %age value denotes the percentage change in blocking rate (computed as $100 \times \frac{\text{IPv6 blocking rate} - \text{IPv4 blocking rate}}{\text{IPv4 blocking rate}}$). Only countries having a statistically significant difference are reported. A negative value indicates that queries sent over IPv4 observed higher blocking rates than those sent over IPv6 in a given country. *ns* indicates the difference was not statistically significant and thus omitted. Korea and Chile are included separately as they demonstrated significant difference in `A`/`AAAA` censorship (Section 4.5.1).

Country	`A` queries	`AAAA` queries	All queries
Thailand (TH)	-7.1 pp (-86.3%)	*ns*	-3.7 pp (-78.1%)
Iran (IR)	-3.2 pp (-12.6%)	-3.0 pp (-12.5%)	-3.1 pp (-12.5%)
Bangladesh (BD)	-5.5 pp (-86.1%)	*ns*	-3.0 pp (-77.9%)
Myanmar (MY)	-3.6 pp (-74.2%)	*ns*	-2.1 pp (-62.3%)
United States (US)	-0.9 pp (-64.8%)	*ns*	-0.5 pp (-42.6%)
Korea (KR)	-1.1pp (-46.5%)	*ns*	*ns*
Chile (CL)	-1.6pp (-58.7%)	*ns*	*ns*

Which countries demonstrate large-scale inconsistencies in their handling of IPv4 and IPv6 DNS traffic? In total, we find only five countries (Thailand, Iran, Bangladesh, Myanmar, and the United States) with statistically significant differences in their handling of DNS queries over IPv4 and IPv6 traffic — suggesting the use of independent censorship mechanisms for IPv4 and IPv6. Interestingly, all these countries appear to have gaps in their IPv6 censorship apparatus — i.e., IPv4 rates of blocking are higher than IPv6 rates in all countries with significant differences. These differences result in IPv6 queries experiencing between 12% and 78% less censorship than their IPv4 counterparts. Once again, this suggests a tendency for network operators to more effectively maintain IPv4 DNS censorship infrastructure than IPv6 infrastructure. These gaps present opportunities for the success of circumvention tools with IPv6 capabilities. Further analysis shows that these differences primarily apply to `A` queries, suggesting that these countries fail to censor most `A` queries when they are sent over IPv6, but that they are able to censor `AAAA` records sent over IPv6 effectively. These findings are particularly noteworthy for circumvention efforts in Thailand, Myanmar, and Iran where IPv6 adoption rates are high (between 15% and 45%) and dual-stack tools may be used for circumvention of DNS censorship.

4.6.2 IPv4/IPv6-inconsistent resolvers

Next, we look at the characteristics of resolver pairs that see inconsistent censorship on their IPv4 and IPv6 interfaces.

Identifying IPv4/IPv6-inconsistent resolvers. Our approach is similar to the methods used to identify `A/AAAA`-inconsistent resolvers (**cf.,** Section 4.5.2). We compare the ratios of censored responses received from a single resolver pair's IPv4 and IPv6 interfaces. We test whether these ratios are statistically different using a z-test with a Šidák corrected $p \leq 1 - .05^{1/n_{rc}}$ being required for a statistically significant difference. A summary of the characteristics of the inconsistent resolvers identified in each country is illustrated in Table 4.5.

Which countries have the largest fraction of IPv4/IPv6-inconsistent resolvers? Two countries from our previous analysis on `A/AAAA`-inconsistencies once again appear with a large fraction of IPv4/IPv6-inconsistent resolvers — Thailand (81%) and Bangladesh (65%). Myanmar presents a new addition with 60% of its resolvers demonstrating IPv4/IPv6-inconsistencies. Other countries were found to have smaller fractions ranging from 12-26%.

How spread out are the IPv4/IPv6-inconsistent resolvers? In order to characterize the spread of IPv4/IPv6-inconsistent resolvers within a country, we compute the entropy of the AS distribution of all resolvers and IPv4/IPv6-inconsistent resolvers within a country ($S_{\text{net}}^{\text{all}}$ and $S_{\text{net}}^{\text{inconsistent}}$) and then compute the KL-divergence of the distribution of inconsistent resolvers from the distribution of all resolvers in that country (∇_{net}). Similar to before, a large change in ∇_{net} means that the IPv4/IPv6-inconsistencies arise from a small fraction of ASes and would suggest that the gaps exist due to local network/resolver configurations — as would be the case if regional operators implement their own DNS censorship mechanisms. Conversely, a small change means that the gaps that exist roughly equally impact all the ASes having resolvers and would suggest that the gaps exist due to misconfigurations in a centralized DNS censorship mechanism. Our results again suggest the presence of a centralized blocking mechanism in Thailand, Bangladesh, and Myanmar ($\nabla_{net} \in [0.13, 0.48]$) which causes the IPv4/IPv6-inconsistencies. The United States has the highest ∇_{net} observed which indicates that regional policies are responsible for the IPv4/IPv6-inconsistencies.

Table 4.5: Characteristics of the resolvers which demonstrated a statistically significant difference in their handling of DNS queries over IPv4 and IPv6 each country. 'AS diversity' denotes the entropies of (all) resolver distribution ($S_{\text{net}}^{\text{all}}$) and IPv4/IPv6-inconsistent resolver distribution ($S_{\text{net}}^{\text{inconsistent}}$) across a country's ASes, and '∇_{net}' represents the Kullback-Leibler divergence of the distribution of inconsistent resolvers from the distribution of all resolvers in the country's ASes (**cf.,** Section 4.6.2). 'Most inconsistent type' denotes the connection type with the most number of IPv4/IPv6-inconsistent resolvers.Korea and Chile are included separately as they demonstrated significant difference in `A/AAAA` censorship (Section 4.5.1).

Country	**Total pairs**	**Inconsistent pairs** (% of total pairs)	**Most inconsistent AS** (# inconsistent pairs)	**AS diversity** $S_{\text{net}}^{\text{all}}$	$S_{\text{net}}^{\text{inconsistent}}$	∇_{net}	**Most inconsistent type** (# inconsistent pairs)
Thailand (TH)	186	151 (81.2%)	AS 9835 Government IT Services (39)	4.50	4.10	0.13	Cable/DSL (108)
Iran (IR)	277	74 (26.7%)	AS 208161 PARSVDS (11)	5.03	3.81	0.87	Cable/DSL (57)
Bangladesh (BD)	29	19 (65.2%)	AS 9230 Bangladesh Online (4)	4.10	3.72	0.39	Cable/DSL (19)
Myanmar (MY)	50	30 (60.0%)	AS 136170 Exabytes Network (10)	3.31	2.42	0.48	Corporate (26)
United States (US)	1,228	151 (12.3%)	AS 30457 WEHOSTWEBSITES (36)	6.28	5.22	1.44	Corporate (102)
Korea (KR)	632	101 (16.0%)	AS 9848 Sejong Telecom (13)	3.12	4.14	0.95	Cable/DSL (73)
Chile (CL)	65	16 (24.6%)	AS 27651 Entel Chile (12)	3.08	1.19	1.04	Corporate (11)

What types of networks exhibit the most IPv4 and IPv6 inconsistencies? An overwhelming majority of the inconsistent resolvers in Thailand, Iran, and Bangladesh (77%-100%) are found to be present in networks with (Maxmind categorized) Cable/DSL connection-types that are typically associated with residential networks. Put in the context of our previous result which suggests the presence of a centralized DNS censorship mechanism in Thailand and Bangladesh, this suggests that the IPv4/IPv6 gaps that exist in this mechanism also extend to residential networks in the country. Myanmar and the United States experience such inconsistencies primarily due to their corporate networks which contain between 67-87% of their inconsistent resolvers.

4.6.3 Characterization of anomalous domains

We now seek to understand the category of domains that get through the infrastructural gap between DNS queries over IPv4 and IPv6. In other words, does the category of domain influence whether or not there is a difference in censorship between IPv4 and IPv6 interfaces?

Identifying differences in domain behaviors within a country We use an approach similar to the one defined in Section 4.5.3 for identifying domains that get through the gap in between IPv4 and IPv6. For each domain, we measure the ratio of blocking that occurs over IPv4 and IPv6. We then apply the z-test with a corrected p-value (as described in Section 4.5.3) to these ratios. This gives us all domains which had significant differences in censorship over IPv4 and IPv6.

Do these IPv4/IPv6-inconsistent domains hint at policy gaps? For each country, we derive domain categories for the following two lists; (1) D_{any} which contains the domains which experienced any blocking events inside a country and (2) $D_{inconsistent}$ which contains the IPv4- and IPv6-inconsistent domains identified by our z-test. We again used KL-divergence between to compare these two category distributions. A small $\nabla_{net}^{domains}$ would signify that the two category distributions are not largely different suggesting the policy gap over IPv4/IPv6 is not content specific; A large $\nabla_{net}^{domains}$ would signify that the category distributions are indeed very different suggesting a content-based policy gap.

Based on this analysis, we see that Bangladesh, US and Iran have the smallest $\nabla_{net}^{domains}$ scores (0-1.3) suggesting that there was little to no difference in the distribution of the two categories. This suggests that for these countries, there is no content-based policy gaps. On the other hand, Pakistan, China and Myanmar had the highest $\nabla_{net}^{domains}$ scores (2.7-4.3). These scores suggest that some categories of domains get through the IPv4/IPv6 censorship gap much more than others. "Gambling" is the category most likely to get through this gap for China and Pakistan and "Government/Military" is the most likely to get through this gap in Iran.

4.7 DNS Censorship Case studies

In this section, we provide interesting findings about the censorship infrastructure in three heavily censored regions which have high IPv6 adoption and have not been discussed in detail in prior sections: Iran, Thailand, and China.

Iran. While Iran supports both IPv4 and IPv6, it is more effective at blocking queries from IPv4 resolvers (in both `A` and `AAAA` records). Notably, 272 of the 277 IPv6 resolvers in Iran rely on 6to4 bridges, all of which block queries over IPv6 at statistically significantly lower rates (-3.2%) than queries over IPv4. However, we find that queries over IPv6 are censored by resolvers using 6to4 bridges at nearly the same rate as resolvers that use native IPv6 connections (note that our sample size of 5 native IPv6 resolvers in Iran is too small for statistical validation). This is a

rather surprising result since one expects that the censorship rate of IPv4 should be independent of whether the IPv6 is native or 6to4 bridged. One potential explanation is that public 6to4 appliances in Iran may be infrastructural, and hosted by the state-run ISPs. See Table E.1 (Appendix) for a breakdown of censorship rates by countries with 6to4 bridges. We also perform an inspection of the rates of censorship in corporate and non-corporate networks in Iran. We find that there are no significant differences between them (23%). This points towards a centralized censorship apparatus within the country.

Thailand. In Thailand, we see significant differences in the handling of `A` queries sent over IPv4 and all other query-network combinations — 8.2% of `A` queries sent to IPv4 resolvers were censored, compared to $\approx$ 1% of all other query and network combinations. For instance, examining the IPv4 interface of one of the resolvers with the largest blocked domains list: it blocked 240 `A` records, but only 15 `AAAA` records from our domain list. Meanwhile, its IPv6 counterpart did not block any domains (either `A` or `AAAA` records). These findings suggest an incomplete DNS censorship infrastructure that is yet to effectively handle IPv6 traffic. Furthermore, the number of requests censored broken down by connection type closely matches the proportion of the resolvers of that connection type. For example, 74% of all resolvers in Thailand are of type Cable/DSL and they are responsible for 72% of all blocked queries. This, coupled with findings in Section 4.6.2 (Thailand having a low ∇_{net}) present a strong case for a centralized mechanism for censorship. Despite the already low rate of native IPv6 censorship (1.3%), we find that the 112 6to4-bridged resolvers had an even lower rate of censorship (0.6%) which suggests, once again, an inadequately equipped IPv6 censorship mechanism.

China. China is unique in that it shows an unusual preference **toward** blocking `AAAA` records over `A` records. While it is known that the Great Firewall can block `AAAA` records and injects IPv6 traffic, it is not clear why it would block those records more than `A` records. Manual investigation reveals 21 domains that almost exclusively have their `AAAA` record blocked, but not their `A` record. For example, `gmail.com`'s `AAAA` record is blocked by over 95% of resolvers in China, but the

corresponding `A` record for `gmail.com` is only blocked by 1% of resolvers. The other 20 domains have similar patterns, leading to China's slight preference in blocking `AAAA` records over `A` records. We do not find any instances of domains in China that are similarly exclusively blocked by `A` record but not `AAAA`. As discussed in Section 4.6.3, China has a high ∇_{net} which suggests that there are individual resolver-level inconsistencies. 90% of all resolvers in China fall within one standard deviation of the mean number of censored queries by all resolvers in China. Those that exceed these bounds are overwhelmingly classified as 'Cellular' by Maxmind's connection type database. Although they make up just 17% of all our resolvers in China, they present 50% of the resolvers with censorship rates higher than mean+std. China has a much larger percentage of native IPv6 resolvers compared to our other case studies and it is still the most consistent in its censorship rate for `AAAA` queries over both 6to4 and native IPv6 resolvers (30%) when compared with their IPv4 counterparts. However, the censorship rate for 6to4 when compared with native IPv6 resolvers was still slightly lower (28% vs 32%). Taken all together, China appears to be the best equipped IPv6 censor with gaps only arising in the IPv4 censorship apparatus.

4.8 Related Work

There is a significant body of previous work investigating DNS based censorship strategies which can be generally broken down along several dimensions: by their duration, scope, and vantage points.

Longitudinal or snapshot measurements of DNS censorship. Many initial investigations into censorship techniques and proposals for measurement methodology provide an evaluation of DNS censorship during a single snapshot in time [126, 40, 163, 147, 134]. Similarly, a snapshot can capture DNS censorship centering around a specific event like an election or social uprising [16]. Established methodology can then be used to gain perspectives on the way that censorship evolves over time as part of a longitudinal measurement [79, 65, 156, 125, 140]. Our work provides a snapshot view of DNS censorship during the period of transition from IPv4 to IPv6.

Regional or global measurements of DNS censorship. Censorship strategies are not universal and each censor is unique to some degree. Targeted measurement studies contribute to a better understanding of block-list infrastructure [139, 79] and explain blocking phenomena [40, 126]. Global studies provide high level view on the use of DNS censorship internationally providing context and and understanding of prevalence to specific censorship techniques [163, 147, 134, 156, 125]. Our work performs a global measurement of DNS censorship, however, through the lens of the differences in IPv4 and IPv6 censorship deployment, we are able to make data-driven hypotheses about the censorship infrastructure deployed in several countries.

Using open resolvers as measurement vantage points. Open DNS resolvers provide a unique vantage point from which to study the Internet and a significant number of previous efforts have relied on them to gain insights into the censorship mechanisms in different regions.

In 2007, Lowe et al. queried open resolvers hosted within China eliciting injected responses. This was the first use of open resolvers to characterize Chinese censorship infrastructure and strategy [103]. In 2008 In 2012, anonymous authors focused on the Chinese Great Firewall's DNS injection and the collateral poisoning effect that it had on open resolvers around the world [124]. In 2014, Wander et al. used open resolvers to look more broadly for global poisoning of DNS resolution by any censoring country finding that spoofed addresses were leaking primarily from China and Iran [166]. Similar to Dagon et al., in 2015 Kührer et al. performed a global study of the reliability of open DNS resolution finding evidence of censorship, injected advertisements, and other suspicious or malicious behavior by returned addresses [90].

More recently, Satellite (2016) outlined a methodology for regularly discovering the set of available open resolvers and querying hosts in order to detect paths that return incorrect or inconsistent resource records [147]. Iris (2017) relied on a similar scanning methodology but developed a set of metrics using follow-up scans and requests that allow the differentiation between inconsistency, misconfiguration, and manipulation [134]. These metrics are based on several factors including: address consistency, TLS certificate validation, HTTP content hash, geolocation, DNS

PTR lookup, and AS information. These supplemental elements allow Iris to handle cases like CDNs, virtual hosting, distributed / forwarded resolution requests and more. The 2020 Censored Planet project incorporated and extended the methods of the Satellite and Iris as part of a comprehensive and longitudinal global censorship measurement study [156]. Although several of these measurement platforms perform filtering to identify reliable resolvers, they are all limited to using open resolvers in the IPv4 space. Consequently, they are unable to provide insights into the state of IPv6 censorship.

IPv6 censorship measurement. Previous censorship studies primarily focus on measurements in the context of IPv4. However, there have been several efforts to explicitly incorporate IPv6. In March 2020 Hoang et al. began collection of DNS records injected by the Great Firewall in order to classify the addresses provided, block-pages injected, and the set of hostnames that receive injections [79]. Their analysis investigates the commonality of addresses injected by the GFW, finding that all injected `AAAA` responses are drawn from the reserved teredo subnet `2001::/32`. However, because this study does not directly compare the injection rates of A vs AAAA or differences in injection to DNS queries sent over IPv4 versus IPv6, our efforts complement their findings and provide a more detailed understanding of IPv6 censorship in China. Although not focused on DNS censorship, a 2021 investigation of HTTP keyword block-lists associated with the Great Firewall found that results are largely the same between IPv4 and IPv6 [171]. However, the authors note that over IPv6 connections, the the Firewall failed to apply it's signature temporary 90 second "penalty box" blocking subsequent connections between the two hosts described by numerous previous studies [174, 35]. This supports our finding that for now the GFW infrastructure supporting IPv4 and IPv6 are implemented and/or deployed independently.

4.9 Discussion and Conclusions

In this section, we detail limitations (Section 4.9.1), directions for future research (Section 4.9.2), and the takeaways of our work. (Section 4.9.3).

4.9.1 Limitations

Measuring censorship in order to gain an understanding of the underlying infrastructure and identify weaknesses for circumvention is a challenging task due to the absence of ground truth for validation and the often probabilistic nature of censorship and networking failures which are easily confused. Although we take care to always err on the side of caution and consider many confounding factors including end-point type and AS diversity, our work is fundamentally a best-effort attempt at trying to identify the gaps that have emerged in DNS censorship deployments because of the increased adoption of IPv6. The limitations of our study arise from three sources.

External sources of data. Our study relies on multiple data sources including Satellite and the Citizen Lab for our domain lists, McAfee's domain labeling services for categorizing our data, and Maxmind's datasets for geolocating and classifying connection types of our resolvers. Although each of these datasets has been validated in the past and are commonly used in research, our results and their corresponding analyses are limited by their reliability.

Statistical limitations. Throughout our study, we aimed to err on the side of caution in order to avoid presenting false-positives in our results. Therefore, we relied on rigorous statistical approaches in order to identify differences in censorship between `A/AAAA` query and IPv4/IPv6 network types. This included grouping related hypotheses together and performing Sidak corrections to ensure that the confidence level achieved across the entire group (rather than for each individual hypothesis) was 95%. Although this provides our results with credibility, our strict methods almost certainly assure false-negatives. In fact, this becomes apparent in our identification of `A/AAAA`-resolvers in China. Our statistical methods identified only six resolvers with significant differences (Section 4.5), however, the manual analysis performed in the case study (Section 4.7) shows that a much larger number of resolvers performed censorship over a small set of just 21 domains. This is a fundamental limitation of any test of two proportions such as the z-test used in this work.

Resolver selection. Our work required us to identify resolvers that were IPv4 and IPv6 capable. For this, we only partially followed the steps outlined by Hendricks et al. (**cf.,** Section 4.3.1). We

decided not to take explicit steps to guarantee that each resolver pair was in fact a single dual stack resolver. Doing so would have resulted in the complete removal of 6to4 bridges and infrastructure resolvers. Our reasoning was that such resolvers do play an important role in the censorship apparatus since they are the ones most commonly used by citizens in a censored region. Therefore, we decided that their removal would have harmed (1) the representativeness of our measurements and (2) the effectiveness of any circumvention approaches that might seek to exploit the censorship gaps identified in this work. On the other hand, this choice means that it is possible that our dataset may have multiple pairs of resolvers in a country that belong to the same resolving infrastructure.

The presence of many 6to4 bridges. One specific category of open resolver that we find to be relatively common in our collected data are those that rely on a 6to4 Bridge. We find many countries where IPv6 resolver availability is supplemented with 6to4 and note that the filtering stage of our resolver methodology ensures that resolver IP pairs geolocate within the same country. This means that any IPv6 queries sent to 6to4 resolvers do traverse national networks as UDP over IPv6 for at least some potion of their path to the final resolver. Further, if our queries do not get served by the resolver's cache, it is expected that the recursive queries will still pass through the censor. Therefore, still providing standard censorship infrastructure some opportunities to censor. Two specific examples of a substantial 6to4 supplement are Brazil in which 94 of the 160 resolvers identified used 6to4 and Thailand with 112 out of 186 resolvers. This understanding of the protocols that are traversing the censored links provide a limitation on our measurement in the sense that we do not get a perfect comparison of IPv4 with IPv6 that, however this trades-off with our ability to gain an interesting interesting perspective on the ability of censors to handle protocols specifically deployed to assist in bridging the IPv4 - IPv6 gap. For example in both Brazil and Thailand the resolvers relying on 6to4 censor requests at lower rates of injection for requests sent over IPv6 on 6to4 as compared with Native IPv6. This suggests an opportunity of circumvention tool developers targeting citizens in these regions.

4.9.2 Circumvention Opportunities

Our work identifies many IPv6-related DNS censorship inconsistencies around the world. Each of these provides interesting directions for future work and opportunities for circumvention tool developers. One particularly interesting direction concerns the discrepancy in the effectiveness of native IPv6 and 6to4-bridged IPv6 censorship. We find that there are several countries where 6to4 bridges experience significantly lower amounts of censorship — including China and India. This presents the possibility of simply using 6to4 tunnels to circumvent DNS censorship. Further, 6to4 is not the only protocol that has seen deployment as part of an effort to bridge the gap between IPv4 and IPv6 deployment. The `6rd` protocol extends the subnet range of 6to4. The Teredo protocol uses UDP to encapsulate IP traffic. Finally, the ISATAP protocol provides an extended 6to4 adding configuration using multicast and DNS. This strategy is not only applicable to DNS censorship circumvention as tunneling protocols can be used to encapsulate any next layer. Our work also highlights the need for integration of IPv6-related measurements by longitudinal censorship measurement platforms such as Satellite, OONI, and ICLab.

4.9.3 Conclusions

The current state of global IPv6 DNS censorship does not divide into a binary bifurcation: block or allow, the landscape is nuanced and developing. However, we identify clear trends in IPv6 censorship that can help motivate future research and circumvention tool development. In this work we present the first large scale measurement of DNS injection based censorship in IPv6 and provide a reproducible methodology for incorporating this measurement into longitudinal measurements going forward. We find significant evidence of DNS injection in IPv6 in many countries around the world. Despite this, we find that censoring countries still favor native network infrastructure, censoring `A` queries over IPv4 at the highest rates. At the country level we observe discrepencies that inform if censorship is centrally coordinated, or if there is looser coordination between entities enacting the DNS censorship regimes. The IP layer and the gap that exists between full IPv6

deployment and the current state of the Internet provide a telling lens for evaluating censorship and developing censorship circumvention technologies. IPv6 plays a growing role in global connectivity and, wherever possible, censorship measurements should endeavor to include IPv6 analysis to provide the most actionable understanding of global censorship infrastructure.

Bibliography

[1] Content ids. https://support.google.com/youtube/answer/2797370?hl=en. Accessed: 2017-02-28.

[2] Django. https://www.djangoproject.com/. Accessed: 2019-11-21.

[3] Facebook privacy policy. https://www.facebook.com/policy.php. Accessed: 2022-03-16.

[4] GeoIP2 Connection Type Database | MaxMind. https://www.maxmind.com/en/geoip2-connection-type-database.

[5] igraph. https://igraph.org/c/. Accessed: 2019-06-18.

[6] Internet censorship (part 2): The technology of information control | townsend center for the humanities. https://townsendcenter.berkeley.edu/blog/internet-censorship-part-2-technology-information-control.

[7] jquery. https://jquery.com. Accessed: 2017-02-28.

[8] Mcafee customer url ticketing system. https://www.trustedsource.org/?p=mcafee. Accessed: 2022-02-01.

[9] Twitter privacy policy. https://developer.twitter.com/en/docs/twitter-for-websites/privacy. Accessed: 2022-03-16.

[10] Web cryptography API. https://www.w3.org/TR/2016/PR-WebCryptoAPI-20161215/.

[11] Orbot: Tor for Android. https://guardianproject.info/apps/orbot/, 2020.

[12] IPv6 Google Statistics | Google. https://www.google.com/intl/en/ipv6/statistics.html, 2022. Accessed: 2022-01-25.

[13] Hervé Abdi et al. Bonferroni and šidák corrections for multiple comparisons. Encyclopedia of measurement and statistics, 3:103–107, 2007.

[14] Carlos Aguilar-Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. XPIR: Private information retrieval for everyone. Proceedings on Privacy Enhancing Technologies, 2:155–174, 2016.

[15] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. Reviews of modern physics, 74(1):47, 2002.

[16] Simurgh Aryan, Homa Aryan, and J Alex Halderman. Internet censorship in iran: A first look. In 3rd USENIX Workshop on Free and Open Communications on the Internet (FOCI 13), 2013.

[17] Aris Athanasiou. Blind-RSA. https://github.com/arisath/Blind-RSA. Accessed: 2019-11-21.

[18] M. Bailey, D. Dittrich, and E. Kenneally. Applying ethical principles to information and communication technology research. Technical report, U.S. Department of Homeland Security, 2013-10.

[19] James Ball. NSA collects millions of text messages daily in 'untargeted' global sweep. The Guardian, Jan 2014.

[20] Michael Bayer. Sqlalchemy 0.3 documentation, 2007.

[21] Daniel J Bernstein. Curve25519: new diffie-hellman speed records. In International Workshop on Public Key Cryptography, pages 207–228. Springer, 2006.

[22] Daniel J Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: Elliptic-curve points indistinguishable from uniform random strings. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, pages 967–980. ACM, 2013.

[23] Jan Beznazwy and Amir Houmansadr. How china detects and blocks shadowsocks. In Proceedings of the ACM Internet Measurement Conference, pages 111–124, 2020.

[24] Cecylia Bocovich, John A Doucette, and Ian Goldberg. Lavinia: An auditpayment protocol for censorship-resistant storage. Financial Cryptography and Data Security, 2017.

[25] Nikita Borisov, Ian Goldberg, and Eric Brewer. Off-the-record communication, or, why not to use PGP. In Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society, WPES '04, pages 77–84, New York, NY, USA, 2004. ACM.

[26] Julien Bringer, Hervé Chabanne, David Pointcheval, and Qiang Tang. Extended private information retrieval and its application in biometrics authentications. In CANS, volume 7, pages 175–193. Springer, 2007.

[27] J Brooks et al. Ricochet: Anonymous instant messaging for real privacy, 2016.

[28] Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. pages 268–289, 2003.

[29] Jan Camenisch, Jean-Marc Piveteau, and Markus Stadler. Blind signatures based on the discrete logarithm problem (rump session). pages 428–432, 1995.

[30] B. Carpenter and K. Moore. Connection of IPv6 Domains via IPv4 Clouds . Internet Engineering Task Force, 2003. Accessed: 2022-02-01.

[31] Melissa Chase, Apoorvaa Deshpande, Esha Ghosh, and Harjasleen Malvai. SEEMless: Secure End-to-End Encrypted Messaging with Less Trust. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS '19, pages 1639–1656. ACM, 2019.

[32] David Chaum. Blind signatures for untraceable payments. pages 199–203, 1982.

[33] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on, pages 41–50. IEEE, 1995.

[34] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W Hong. Freenet: A distributed anonymous information storage and retrieval system. In Designing Privacy Enhancing Technologies, pages 46–66. Springer, 2001.

[35] Richard Clayton, Steven J Murdoch, and Robert NM Watson. Ignoring the great firewall of china. In International Workshop on Privacy Enhancing Technologies, pages 20–35. Springer, 2006.

[36] Bram Cohen. Bittorrent protocol, 2006.

[37] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila. A formal security analysis of the Signal messaging protocol. In 2017 IEEE European Symposium on Security and Privacy (EuroS&P), pages 451–466, April 2017.

[38] Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican, and Kevin Milner. On Ends-to-Ends Encryption: Asynchronous Group Messaging with Strong Security Guarantees. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS '18, pages 1802–1819, New York, NY, USA, 2018. ACM.

[39] US Congress. Digital Millennium Copyright Act. Public Law, 105(304):112, 1998.

[40] Global Internet Freedom Consortium et al. The great firewall revealed. http://www.internetfreedom.org/files/WhitePaper/ChinaGreatFirewallRevealed.pdf, 2002.

[41] H. Corrigan-Gibbs, D. Boneh, and D. Mazières. Riposte: An Anonymous Messaging System Handling Millions of Users. In 2015 IEEE Symposium on Security and Privacy, pages 321–338, May 2015.

[42] Henry Corrigan-Gibbs and Bryan Ford. Dissent: Accountable Anonymous Group Messaging. In Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS '10, pages 340–350. ACM, 2010.

[43] Henry Corrigan-Gibbs, David Isaac Wolinsky, and Bryan Ford. Proactively accountable anonymous messaging in verdict. In Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13), pages 147–162, 2013.

[44] Jakub Czyz, Mark Allman, Jing Zhang, Scott Iekel-Johnson, Eric Osterweil, and Michael Bailey. Measuring ipv6 adoption. In Proceedings of the 2014 ACM Conference on SIGCOMM, SIGCOMM '14, pages 87–98, New York, NY, USA, 2014. ACM.

[45] Jakub Czyz, Matthew Luckie, Mark Allman, Michael Bailey, et al. Don't forget to lock the back door! a characterization of ipv6 network security policy. In Network and Distributed Systems Security (NDSS), 2016.

[46] David Dagon, Chris Lee, Wenke Lee, and Niels Provos. Corrupted dns resolution paths: The rise of a malicious resolution authority. 2008.

[47] George Danezis. Statistical Disclosure Attacks. In Security and Privacy in the Age of Uncertainty, pages 421–426. Springer US, 2003.

[48] George Danezis. Better Anonymous Communications. PhD thesis, University of Cambridge, 2004.

[49] George Danezis, Claudia Diaz, and Carmela Troncoso. Two-sided statistical disclosure attack. In Privacy Enhancing Technologies, pages 30–44. Springer Berlin Heidelberg, 2007.

[50] George Danezis and Carmela Troncoso. Vida: How to use bayesian inference to de-anonymize persistent communications. In Privacy Enhancing Technologies, pages 56–72. Springer Berlin Heidelberg, 2009.

[51] Casey Devet, Ian Goldberg, and Nadia Heninger. Optimally robust private information retrieval. In USENIX Security Symposium, pages 269–283, 2012.

[52] Amogh Dhamdhere, Matthew Luckie, Bradley Huffaker, kc claffy, Ahmed Elmokashfi, and Emile Aben. Measuring the Deployment of IPv6: Topology, Routing and Performance. In Proceedings of the 2012 Internet Measurement Conference, IMC '12, pages 537–550, New York, NY, USA, 2012. ACM.

[53] Mario Di Raimondo, Rosario Gennaro, and Hugo Krawczyk. Secure off-the-record messaging. In Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society, WPES '05, pages 81–89, New York, NY, USA, 2005. ACM.

[54] Roger Dingledine. Obfsproxy: the next step in the censorship arms race. Tor Project official blog, 2012.

[55] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004.

[56] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13, SSYM'04, pages 21–21, Berkeley, CA, USA, 2004. USENIX Association.

[57] D. Dittrich and E. Kenneally. The menlo report: Ethical principles guiding information and communication technology research. Technical report, U.S. Department of Homeland Security, 2012-08.

[58] Suelette Dreyfus. The idiot savants' guide to rubberhose. Official documentation for the Rubberhose file system, 2001.

[59] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. Zmap: Fast internet-wide scanning and its security applications. In In Proceedings of the 22nd USENIX Security Symposium, 2013.

[60] Paul Erdös and Alfréd Rényi. On random graphs, I. Publicationes Mathematicae (Debrecen), 6:290–297, 1959.

[61] Facebook. Government requests report. https://govtrequests.facebook.com/, 2017.

[62] Roy Thomas Fielding. Architectural styles and the design of network-based software architectures. PhD thesis, University of California, Irvine, 2000.

[63] Anna Fifield. Chinese app on Xi's ideology allows data access to users' phones, report says. The Washington Post, October 2019.

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

[65] Arturo Filasto and Jacob Appelbaum. Ooni: Open observatory of network interference. In FOCI, 2012.

[66] Electronic Frontier Foundation. A guide to youtube removals. https://www.eff.org/issues/intellectual-property/guide-to-youtube-removals.

[67] T. Frosch, C. Mainka, C. Bader, F. Bergsma, J. Schwenk, and T. Holz. How secure is TextSecure? In 2016 IEEE European Symposium on Security and Privacy (EuroS&P), pages 457–472, March 2016.

[68] Genevieve Gebhart and Tadayoshi Kohno. Internet censorship in thailand: User practices and potential threats. In 2017 IEEE European symposium on security and privacy (EuroS&P), pages 417–432. IEEE, 2017.

[69] Google. Transparency report. https://transparencyreport.google.com/, 2017.

[70] Devashish Gosain, Anshika Agarwal, Sahil Shekhawat, H. B. Acharya, and Sambuddho Chakravarty. Mending wall: On the implementation of censorship in India. In SecureComm. Springer, 2017.

[71] Andy Greenberg. Signal is finally bringing its secure messaging to the masses. Feb 2020.

[72] Trinabh Gupta, Natacha Crooks, Whitney Mulhern, Srinath TV Setty, Lorenzo Alvisi, and Michael Walfish. Scalable and private media consumption with popcorn. In NSDI, pages 91–107, 2016.

[73] Syed Mahbub Hafiz and Ryan Henry. Querying for queries: Indexes of queries for efficient and expressive IT-PIR. Cryptology ePrint Archive, Report 2017/825, 2017. https://eprint.iacr.org/2017/825.

[74] Jonna Häkkilä and Craig Chatfield. 'It's like if you opened someone else's letter': User perceived privacy and social practices with SMS communication. In Proceedings of the 7th International Conference on Human Computer Interaction with Mobile Devices & Services, MobileHCI '05, pages 219–222. ACM, 2005.

[75] Sam Havron, Diana Freed, Rahul Chatterjee, Damon McCoy, Nicola Dell, and Thomas Ristenpart. Clinical computer security for victims of intimate partner violence. In 28th USENIX Security Symposium (USENIX Security 19), pages 105–122, Santa Clara, CA, August 2019. USENIX Association.

[76] Kieran Healy. Using Metadata to find Paul Revere, Jun 2013.

[77] Luuk Hendriks, Ricardo de Oliveira Schmidt, Roland van Rijswijk-Deij, and Aiko Pras. On the potential of ipv6 open resolvers for ddos attacks. In International Conference on Passive and Active Network Measurement, pages 17–29. Springer, 2017.

[78] Friedhelm Hillebrand, Finn Trosby, Kevin Holley, and Ian Harris. Short Message Service (SMS): The Creation of Personal Global Text Messaging. Wiley, 2010.

[79] NP. Hoang, AA. Niaki, J. Dalek, J. Knockel, P. Lin, B. Marczak, M. Crete-Nishihata, P. Gill, and M. Polychronakis. How Great is the Great Firewall? Measuring China's DNS Censorship. In 30th USENIX Security Symposium, pages 3381–3398. USENIX Association, 2021.

[80] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. J. Amer. Statist. Assoc., 58:13–30, 1963.

[81] Geoff Huston. IPv6 in 2020. https://blog.apnic.net/2021/02/08/ipv6-in-2020/. Accessed: 2022-01-25.

[82] Global Web Index. VPN users around the world. 2018.

[83] Intel. Intel Software Guard Extensions. https://software.intel.com/en-us/sgx.

[84] Bernstein D. J., van Gastel B., Janssen W., Lange T., Schwabe P., and Smetsers S. Tweetnacl: A crypto library in 100 tweets. In: Aranha D., Menezes A. (eds) Progress in Cryptology - LATINCRYPT, 2014., 2015.

[85] J Jia and P Smith. Psiphon: Analysis and estimation, 2004.

[86] Johns Hopkins Foreign Affairs Symposium. The price of privacy: Re-evaluating the NSA. https://youtu.be/kV2HDM86XgI?t=1079, April 2014.

[87] Ben Jones, Roya Ensafi, Nick Feamster, Vern Paxson, and Nick Weaver. Ethical concerns for censorship measurement. In Proceedings of the 2015 ACM SIGCOMM Workshop on Ethics in Networked Systems Research, NS Ethics '15, page 17–19, New York, NY, USA, 2015. Association for Computing Machinery.

[88] Ari Juels and Burton S Kaliski Jr. Pors: Proofs of retrievability for large files. In Proceedings of the 14th ACM conference on Computer and communications security, pages 584–597. Acm, 2007.

[89] D. Kesdogan, D. Agrawal, Vinh Pham, and D. Rautenbach. Fundamental limits on the anonymity provided by the MIX technique. In 2006 IEEE Symposium on Security and Privacy (S&P'06), pages 14 pp.–99, 2006.

[90] Marc Kührer, Thomas Hupperich, Jonas Bushart, Christian Rossow, and Thorsten Holz. Going wild: Large-scale classification of open dns resolvers. In Proceedings of the 2015 Internet Measurement Conference, pages 355–368, 2015.

[91] Solomon Kullback and Richard A Leibler. On information and sufficiency. The annals of mathematical statistics, 22(1):79–86, 1951.

[92] Eyal Kushilevitz and Rafail Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on, pages 364–373. IEEE, 1997.

[93] Albert Kwon, Henry Corrigan-Gibbs, Srinivas Devadas, and Bryan Ford. Atom: Horizontally scaling strong anonymity. In Proceedings of the 26th Symposium on Operating Systems Principles, pages 406–422. ACM, 2017.

[94] Albert Kwon, David Lazar, Srinivas Devadas, and Bryan Ford. Riffle. Proceedings on Privacy Enhancing Technologies, 2016(2):115–134, 2016.

[95] Albert Kwon, David Lazar, Srinivas Devadas, and Bryan Ford. Riffle. Proceedings on Privacy Enhancing Technologies, 2016(2):115–134, 2016.

[96] Citizen Lab and Others. Url testing lists intended for discovering website censorship, 2014. https://github.com/citizenlab/test-lists.

[97] Brandom LaBelle. Secure messaging apps are growing faster in corrupt countries. Apptopia blog, 2018.

[98] Adam Langley. Pond, 2015.

[99] Ben Laurie, Adam Langley, and Emilia Kasper. Certificate transparency. Technical report, 2013.

[100] Stevens Le Blond, David Choffnes, Wenxuan Zhou, Peter Druschel, Hitesh Ballani, and Paul Francis. Towards efficient traffic-analysis resistant anonymity networks. SIGCOMM Comput. Commun. Rev., 43(4):303–314, August 2013.

[101] Hemi Leibowitz, Ania M Piotrowska, George Danezis, and Amir Herzberg. No right to remain silent: isolating malicious mixes. In 28th USENIX Security Symposium (USENIX Security 19), pages 1841–1858, 2019.

[102] Benoît Libert and Jean-Jacques Quisquater. Efficient signcryption with key privacy from gap Diffie-Hellman groups. pages 187–200, 2004.

[103] Graham Lowe, Patrick Winters, and Michael L Marcus. The great dns wall of china. MS, New York University, 21:1, 2007.

[104] Lumen. Lumen database. https://lumendatabase.org/.

[105] Lumen. Brazil - court order to twitter. https://www.lumendatabase.org/notices/12866354, 2016.

[106] Joshua Lund. Technology preview: Sealed sender for signal, Oct 2018.

[107] Eric Mahe and Jean-Marie Chauvet. Fast gpgpu-based elliptic curve scalar multiplication. IACR Cryptology ePrint Archive, 2014:198, 2014.

[108] Nayantara Mallesh and Matthew Wright. The reverse statistical disclosure attack. In Information Hiding, pages 221–234. Springer Berlin Heidelberg, 2010.

[109] Bill Marczak, John Scott-Railton, Sarah McKune, Bahr Abdul Razzak, and Ron Deibert. Hide and Seek: Tracking NSO Group's Pegasus Spyware to Operations in 45 Countries. Technical Report 113, University of Toronto, September 2018.

[110] Moxie Marlinspike. Advanced cryptographic ratcheting. Signal Blog, November 2013.

[111] Nick Mathewson and Roger Dingledine. Practical traffic analysis: Extending and resisting statistical disclosure. In Privacy Enhancing Technologies, pages 17–34. Springer Berlin Heidelberg, 2005.

[112] Susan E. McGregor, Polina Charters, Tobin Holliday, and Franziska Roesner. Investigating the computer security practices and needs of journalists. In 24th USENIX Security Symposium (USENIX Security 15), pages 399–414, Washington, D.C., August 2015. USENIX Association.

[113] David McGrew and John Viega. The galois/counter mode of operation (gcm). Submission to NIST Modes of Operation Process, 20, 2004.

[114] Jon McLachlan, Andrew Tran, Nicholas Hopper, and Yongdae Kim. Scalable onion routing with torsk. In Proceedings of the 16th ACM conference on Computer and communications security, pages 590–599. ACM, 2009.

[115] Mega. Mega, the privacy company. https://mega.nz/.

[116] Marcela S Melara, Aaron Blankstein, Joseph Bonneau, Edward W Felten, and Michael J Freedman. Coniks: Bringing key transparency to end users. In Usenix Security, pages 383–398, 2015.

[117] Ralph C Merkle. Method of providing digital signatures, January 5 1982. US Patent 4,309,569.

[118] Andrew Miller, Ari Juels, Elaine Shi, Bryan Parno, and Jonathan Katz. Permacoin: Repurposing Bitcoin work for data preservation. In Security and Privacy (SP), 2014 IEEE Symposium on, pages 475–490. IEEE, 2014.

[119] Prateek Mittal, Femi G Olumofin, Carmela Troncoso, Nikita Borisov, and Ian Goldberg. PIR-Tor: Scalable anonymous communication using private information retrieval. In USENIX Security Symposium, pages 31–31, 2011.

[120] P. Mockapetris. DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION . Internet Engineering Task Force, 1987. Accessed: 2022-01-27.

[121] Austin Murdock, Frank Li, Paul Bramsen, Zakir Durumeric, and Vern Paxson. Target generation for internet-wide ipv6 scanning. In Proceedings of the 2017 Internet Measurement Conference, pages 242–253, 2017.

[122] Zubair Nabi. The anatomy of web censorship in pakistan. In 3rd USENIX Workshop on Free and Open Communications on the Internet (FOCI 13), 2013.

[123] Ellen Nakashima. Apple vows to resist FBI demand to crack iPhone linked to San Bernardino attacks. https://www.washingtonpost.com/world/national-security/us-wants-apple-to-help-unlock-iphone-used-by-san-bernardino-shooter/2016/02/16/69b903ee-d4d9-11e5-9823-02b905009f99_story.html, February 2016.

[124] Hovership Nebuchadnezzar. The collateral damage of internet censorship by dns injection. ACM SIGCOMM CCR, 42(3):10–1145, 2012.

[125] Arian Akhavan Niaki, Shinyoung Cho, Zachary Weinberg, Nguyen Phong Hoang, Abbas Razaghpanah, Nicolas Christin, and Phillipa Gill. Iclab: A global, longitudinal internet censorship measurement platform. In 2020 IEEE Symposium on Security and Privacy (SP), pages 135–151. IEEE, 2020.

[126] Arian Akhavan Niaki, Nguyen Phong Hoang, Phillipa Gill, Amir Houmansadr, et al. Triplet censors: Demystifying great Firewall's DNS censorship behavior. In 10th USENIX Workshop on Free and Open Communications on the Internet (FOCI 20), 2020.

[127] Open Whisper Systems. Signal source code. https://github.com/signalapp, 2013.

[128] Rafail Ostrovsky and William E Skeith III. A survey of single-database private information retrieval: Techniques and applications. In International Workshop on Public Key Cryptography, pages 393–411. Springer, 2007.

[129] Simon Oya, Carmela Troncoso, and Fernando Pérez-González. Meet the family of statistical disclosure attacks. In IEEE Global Conference on Signal and Information Processing, GlobalSIP, pages 233–236. IEEE, 2013.

[130] Ramakrishna Padmanabhan, Arturo Filastò, Maria Xynou, Ram Sundara Raman, Kennedy Middleton, Mingwei Zhang, Doug Madory, Molly Roberts, and Alberto Dainotti. A multi-perspective view of internet censorship in myanmar. In Proceedings of the ACM SIGCOMM 2021 Workshop on Free and Open Communications on the Internet, pages 27–36, 2021.

[131] David A Patterson, Garth Gibson, and Randy H Katz. A case for redundant arrays of inexpensive disks (RAID), volume 17. ACM, 1988.

[132] Kari Paul. Facebook under fire as human rights groups claim 'censorship' of pro-Palestine posts. https://www.theguardian.com/media/2021/may/26/pro-palestine-censorship-facebook-instagram, May 2021.

[133] Paul Pearce, Roya Ensafi, Frank Li, Nick Feamster, and Vern Paxson. Augur: Internet-wide detection of connectivity disruptions. In 2017 IEEE Symposium on Security and Privacy (SP), pages 427–443. IEEE, 2017.

[134] Paul Pearce, Ben Jones, Frank Li, Roya Ensafi, Nick Feamster, Nick Weaver, and Vern Paxson. Global measurement of DNS manipulation. In 26th USENIX Security Symposium (USENIX Security 17), pages 307–323, 2017.

[135] Fernando Pérez-González and Carmela Troncoso. Understanding statistical disclosure: A least squares approach. In Privacy Enhancing Technologies, pages 38–57. Springer Berlin Heidelberg, 2012.

[136] David Pointcheval and Olivier Sanders. Short randomizable signatures. pages 111–126, 2016.

[137] David Pointcheval and Olivier Sanders. Reassessing security of randomizable signatures. pages 319–338, 2018.

[138] Matthew Prince. Happy IPv6 Day: Usage On the Rise, Attacks Too. https://blog.cloudflare.com/ipv6-day-usage-attacks-rise/. Accessed: 2022-01-25.

[139] Reethika Ramesh, Ram Sundara Raman, Matthew Bernhard, Victor Ongkowijaya, Leonid Evdokimov, Anne Edmundson, Steven Sprecher, Muhammad Ikram, and Roya Ensafi. Decentralized control: A case study of russia. In Network and Distributed Systems Security (NDSS) Symposium 2020, 2020.

[140] Abbas Razaghpanah, Anke Li, Arturo Filasto, Rishab Nithyanand, Vasilis Ververis, Will Scott, and Phillipa Gill. Exploring the design space of longitudinal censorship measurement platforms. arXiv preprint arXiv:1606.01979, 2016.

[141] M. G. Reed, P. F. Syverson, and D. M. Goldschlag. Anonymous connections and onion routing. IEEE Journal on Selected Areas in Communications, 16(4):482–494, May 1998.

[142] Armin Ronacher. Welcome—flask (a python microframework). URL: http://flask.pocoo.org/(visited on 02/28/2017), 2010.

[143] P. Rösler, C. Mainka, and J. Schwenk. More is Less: On the End-to-End Security of Group Chats in Signal, WhatsApp, and Threema. In 2018 IEEE European Symposium on Security and Privacy (EuroS&P), pages 415–429, April 2018.

[144] Markus Rückert. Lattice-based blind signatures. pages 413–430, 2010.

[145] Michael Schliep and Nicholas Hopper. End-to-end secure mobile group messaging with conversation integrity and deniability. In Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society, WPES'19, pages 55–73. ACM, 2019.

[146] Michael Schliep, Eugene Vasserman, and Nicholas Hopper. Consistent Synchronous Group Off-The-Record Messaging with SYM-GOTR. Proceedings on Privacy Enhancing Technologies, 2018(3):181–202, 2018.

[147] Will Scott, Thomas Anderson, Tadayoshi Kohno, and Arvind Krishnamurthy. Satellite: Joint analysis of cdns and network-level interference. In 2016 USENIX Annual Technical Conference (USENIX ATC 16), pages 195–208, 2016.

[148] Adi Shamir. How to share a secret. Communications of the ACM, 22(11):612–613, 1979.

[149] Ryan Shandler. Measuring the political and social implications of government-initiated cyber shutdowns. In 8th USENIX Workshop on Free and Open Communications on the Internet (FOCI 18), 2018.

[150] Kushagra Singh, Gurshabad Grover, and Varun Bansal. How india censors the web. In 12th ACM Conference on Web Science, pages 21–28, 2020.

[151] Markus Stadler, Jean-Marc Piveteau, and Jan Camenisch. Fair blind signatures. pages 209–219, 1995.

[152] Emily Stark, Michael Hamburg, and Dan Boneh. Symmetric cryptography in javascript. In Computer Security Applications Conference, 2009. ACSAC'09. Annual, pages 373–381. IEEE, 2009.

[153] Emil Stefanov, Marten Van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path oram: an extremely simple oblivious ram protocol. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, pages 299–310. ACM, 2013.

[154] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. The first collision for full sha-1.

[155] AB Stubblefield and Dan S Wallach. Dagster: censorship-resistant publishing without replication. Rice University, Technical Report TR0l-380, 2001.

[156] Ram Sundara Raman, Prerana Shenoy, Katharina Kohls, and Roya Ensafi. Censored planet: An internet-wide, longitudinal censorship observatory. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pages 49–66, 2020.

[157] S. Thomson, C. Huitema, Ksinant V., and Souissi M. DNS Extensions to Support IP Version 6 . Internet Engineering Task Force, 2003. Accessed: 2022-01-27.

[158] M. C. Tschantz, S. Afroz, Anonymous, and V. Paxson. SoK: Towards Grounding Censorship Circumvention in Empiricism. In 2016 IEEE Symposium on Security and Privacy (SP), pages 914–933, May 2016.

[159] Twitter. Removal requests. https://transparency.twitter.com/en/removal-requests.html, 2017.

[160] Nirvan Tyagi, Yossi Gilad, Derek Leung, Matei Zaharia, and Nickolai Zeldovich. Stadium: A distributed metadata-private messaging system. In Proceedings of the 26th Symposium on Operating Systems Principles, pages 423–440. ACM, 2017.

[161] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith. SoK: Secure messaging. In 2015 IEEE Symposium on Security and Privacy, pages 232–249, May 2015.

[162] Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: Scalable private messaging resistant to traffic analysis. In Proceedings of the 25th Symposium on Operating Systems Principles, SOSP '15, pages 137–152. ACM, 2015.

[163] Benjamin VanderSloot, Allison McDonald, Will Scott, J Alex Halderman, and Roya Ensafi. Quack: Scalable remote measurement of application-layer censorship. In 27th USENIX Security Symposium (USENIX Security 18), pages 187–202, 2018.

[164] Marc Waldman and David Mazieres. Tangler: a censorship-resistant publishing system based on document entanglements. In Proceedings of the 8th ACM conference on Computer and Communications Security, pages 126–135. ACM, 2001.

[165] Marc Waldman, Aviel D Rubin, and Lorrie Faith Cranor. Publius: A robust, tamper-evident censorship-resistant web publishing system. In 9th USENIX Security Symposium, pages 59–72, 2000.

[166] Matthäus Wander, Christopher Boelmann, Lorenz Schwittmann, and Torben Weis. Measurement of globally visible dns injection. IEEE Access, 2:526–536, 2014.

[167] Shiyuan Wang, Divyakant Agrawal, Amr El Abbadi, Lamia Youseff, Rich Wolski, Fang Yu, Tevfik Bultan, Oscar H Ibarra, Xia Zhou, Alessandra Sala, et al. Generalizing PIR for practical private retrieval of public data. DBSec, 10:1–16, 2010.

[168] Yang Wang, Mark Manulis, Man Ho Au, and Willy Susilo. Relations among privacy notions for signcryption and key invisible "Sign-then-Encrypt". pages 187–202, 2013.

[169] Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. nature, 393(6684):440, 1998.

[170] Michael Weinberg. Universal music group and youtube agree to forget about fair use. https://www.publicknowledge.org/news-blog/blogs/universal-music-group-and-youtube-agree-to-forget-about-fair-use, 2014.

[171] Zachary Weinberg, Diogo Barradas, and Nicolas Christin. Chinese wall or swiss cheese? keyword filtering in the great firewall of china. In Proceedings of the Web Conference 2021, pages 472–483, 2021.

[172] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper, 2014.

[173] Eric Wustrow, Colleen Swanson, and J Alex Halderman. Tapdance: End-to-middle anticensorship without flow blocking. In USENIX Security, pages 159–174, 2014.

[174] Xueyang Xu, Z Morley Mao, and J Alex Halderman. Internet censorship in china: Where does the filtering occur? In International Conference on Passive and Active Network Measurement, pages 133–142. Springer, 2011.

[175] Tarun Kumar Yadav, Akshat Sinha, Devashish Gosain, Piyush Kumar Sharma, and Sambuddho Chakravarty. Where the light gets in: Analyzing web censorship mechanisms in india. In Proceedings of the Internet Measurement Conference 2018, pages 252–264, 2018.

[176] Tarun Kumar Yadav, Akshat Sinha, Devashish Gosain, Piyush Kumar Sharma, and Sambuddho Chakravarty. Where the light gets in: Analyzing web censorship mechanisms in India. In Internet Measurement Conference. ACM, 2018.

Appendix A

Proof of Theorem 1

Consider first an arbitrary non-associate Charlie, with probability r_c of appearing in a random or target epoch. We first analyze the probability that Alice appears above Charlie in the ranking after n random and target epochs.

Recall that the attack maintains a "score" for each user, increasing by 1 each time the user appears in a target epoch, and decreasing by 1 each time the user appears in a random epoch. Define $2n$ random variables $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_n$, corresponding to the signed difference in Alice and Charlie's scores during each of the n random epochs (X_i's) and target epochs (Y_i's). So each $X_i, Y_i \in \{-1, 0, 1\}$ and the sum $\bar{X} = \sum_{1 \leq i \leq n}(X_i + Y_i)$ is the difference in Alice and Charlie's score at the end of the attack. We wish to know the probability that $\bar{X} > 0$.

By the stated probability assumptions, we know the expected value of all of these random variables: $\mathbb{E}[X_i] = r_c - r_a$, $\mathbb{E}[Y_i] = t_a - r_c$, and therefore by linearity of expectation, $\mathbb{E}[\bar{X}] = n(t_a - r_a)$. Crucially, note that this is **independent of Charlie's probability** r_c because we included the same number of random and target epochs.

We can now apply Hoeffding's inequality [80] over the sum of these $2n$ independent, bounded random variables X_i, Y_i to conclude that $\Pr[\bar{X} \leq 0] \leq \exp(-n(t_a - r_a)^2/4)$.

Noting that this bound does not depend on the particular non-associate Charlie in any way, we can apply a simple union bound over all $\leq m$ non-associates to obtain the stated result.

Appendix B

Proof Of Security For One-Way Sealed Sender Conversations

We now give a proof that the protocol in §3.6.2 realizes the ideal functionality in Figure 3.6. As mentioned, we give this proof in the standalone model with static corruptions.

We define the simulator Sim as follows:

Setup: At startup, Sim generates long-term key pairs $(\mathsf{pk}_{P_i}, \mathsf{sk}_{P_i})$ for each honest user $P_i \in P_H$. Next, Sim receives a public key pk_{P_j} for each corrupt user $P_j \in P_C$ from the adversary.

Sim initializes an empty table $\mathbb{T}$ with format

$$(\mathtt{cid}, P_s, \mathsf{pk}_{s,\mathtt{cid}}, \mathsf{sk}_{s,\mathtt{cid}}, P_r, \mathsf{pk}_{r,\mathtt{cid}}, \mathsf{sk}_{r,\mathtt{cid}})$$

where P_s is the identity of the conversation initiator, $(\mathsf{pk}_{s,\mathtt{cid}}, \mathsf{sk}_{s,\mathtt{cid}})$ is the keypair used by P_s in conversation $\mathtt{cid}$, P_r is the initial receiver, and $(\mathsf{pk}_{r,\mathtt{cid}}, \mathsf{sk}_{r,\mathtt{cid}})$ is the keypair used by P_r in conversation $\mathtt{cid}$. Some elements in these entries may be empty if Sim does not know the value. We will represent unknown elements with $\cdot$.

Sim also initializes an empty message table $\mathbb{M}$ with format

$$(\mathtt{cid}, \mathtt{mid}, c)$$

Note that the definition presented in Figure 3.6 is in terms for **pull notifications**, while the protocol in §3.6.2 is in terms of **push notifications**. However, the push notification in the protocol, modeled after how Signal actually works, are essentially a sustained pull. That is, opening a longterm connection is equivalent to having the receiver continuously sending pull requests to the

server. To bridge this gap, the simulator maintains a list of open connections. At each time step, the simulator iterates through the list of open connections are sends a ReceiveMessage to the ideal functionality of that players part. Similarly, we expect that honest users will do this if they want push-style notifications.

(1) **Honest user starts a conversation with an honest user.** When Sim receives the message $(\mathsf{ApproveNewConvo}, P_r, \mathtt{cid})$ from the ideal functionality, samples $(\mathsf{pk}_{s,\mathtt{cid}}, \mathsf{sk}_{s,\mathtt{cid}}) \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSKeyGen}(1^\lambda)$. Sim retrieves the longterm information for user P_r, **i.e.** $\mathsf{pk}_{P_r}, \mathsf{sk}_{P_r}$. Add the entry

$$(\mathtt{cid}, \cdot, \mathsf{pk}_{s,\mathtt{cid}}, \mathsf{sk}_{s,\mathtt{cid}}, P_r, \mathsf{pk}_{P_r}, \mathsf{sk}_{P_r})$$

to $\mathbb{T}$ and then does the following:

(a) Encrypt $c \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSEnc}(\texttt{"init"} \| \mathsf{pk}_{s,\mathtt{cid}}, \mathsf{sk}_{s,\mathtt{cid}}, \mathsf{pk}_{P_r})$

(b) Send c to P_{service}

If Sim gets c' from P_{service} for P_r and $c = c'$, Sim performs the following

(a) sends an acknowledgment to P_{service} on behalf of P_r for P_s

(b) receives the acknowledgment on behalf of P_s

(c) sends (Approve) to ideal functionality

Otherwise, Sim sends (Disapprove) to the ideal functionality

(2) **Honest user starts a conversation with an corrupt user.** When Sim receives the message $(\mathsf{ApproveNewConvoCorrupt}, P_s, P_r, \mathtt{cid})$ from the ideal functionality, samples $(\mathsf{pk}_{s,\mathtt{cid}}, \mathsf{sk}_{s,\mathtt{cid}}) \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSKeyGen}(1^\lambda)$. Sim retrieves the longterm information for user P_r, **i.e.** pk_{P_r}. Add the entry

$$(\mathtt{cid}, P_s, \mathsf{pk}_{s,\mathtt{cid}}, \mathsf{sk}_{s,\mathtt{cid}}, P_r, \mathsf{pk}_{P_r}, \cdot)$$

to $\mathbb{T}$ and then does the following:

(a) Encrypt $c \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSEnc}(\texttt{"init"} \| \mathsf{pk}_{s,\texttt{cid}}, \mathsf{sk}_{P_s}, \mathsf{pk}_{P_r})$

(b) Send c to P_{service}

If Sim gets an acknowledgment from P_{service} for P_s, Sim sends (Approve) to ideal functionality. Otherwise, Sim sends (Disapprove) to the ideal functionality.

(3) **Corrupt user starts a conversation with an honest user.** When Sim receives a message $c \| \mathsf{pk}_e$ from P_{service} for an honest player P_h, Sim retrieves the longterm information for that player, **i.e.** $\mathsf{pk}_{P_s}, \mathsf{sk}_{P_s}$. Sim then does the following:

(a) Decrypt and verify $(\texttt{"init"} \| \mathsf{pk}_e, x, \mathsf{pk}_c) \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSDecVer}(\mathsf{sk}_{P_h}, c)$ On failure, Sim halts.

(b) Find a player P_c with longterm public key $\mathsf{pk}_{P_c} = \mathsf{pk}_c$. If no such player exists, Sim halts.

(c) Send $(\mathsf{StartConvo}, P_h)$ to the ideal functionality on behalf of P_c and receive $(\mathsf{ApproveNewConvoCorrupt}, P_c, P_h, \texttt{cid})$ in return. Sim responds with (Approve). Sim drops the resulting notification.

(d) Generate an acknowledgment message using pk_e and sk_h and send it to P_{service} on behalf of P_h for the identity pk_e

Finally, Sim adds the entry

$$(\texttt{cid}, P_h, \mathsf{pk}_{P_h}, \mathsf{sk}_{P_h}, P_c, \mathsf{pk}_e, \cdot)$$

to $\mathbb{T}$

(4) **Anonymous honest user sends a message to another honest user.** When Sim receives the message $(\mathsf{NotifyAnonymousSendMessage}, \texttt{cid}, \texttt{mid}, |m|)$ from the ideal functionality, Sim looks up the entry

$$(\texttt{cid}, \cdot, \mathsf{pk}_{s,\texttt{cid}}, \mathsf{sk}_{s,\texttt{cid}}, P_r, \mathsf{pk}_{r,\texttt{cid}}, \mathsf{sk}_{r,\texttt{cid}})$$

in $\mathbb{T}$ and performs the following:

(a) Samples $m_0 \leftarrow_{\$} \{0,1\}^{|m|}$

(b) Computes $c \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSEnc}(m_0, \mathsf{sk}_{r,\mathtt{cid}}, \mathsf{pk}_{s,\mathtt{cid}})$

(c) Sends c to the P_{service} for $\mathsf{pk}_{s,\mathtt{cid}}$ from $\mathsf{pk}_{r,\mathtt{cid}}$

(d) Records the entry $(\mathtt{cid}, \mathtt{mid}, c)$ in $\mathbb{M}$

(5) **Non-anonymous honest user sends a message to another honest user.** When Sim receives the message $(\mathsf{NotifySendMessage}, \mathtt{cid}, \mathtt{mid}, P_r, |m|)$ from the ideal functionality, Sim looks up the entry

$$(\mathtt{cid}, \cdot, \mathsf{pk}_{s,\mathtt{cid}}, \mathsf{sk}_{s,\mathtt{cid}}, P_r, \mathsf{pk}_{r,\mathtt{cid}}, \mathsf{sk}_{r,\mathtt{cid}})$$

in $\mathbb{T}$ and performs the following:

(a) Samples $m_0 \leftarrow_{\$} \{0,1\}^{|m|}$

(b) Computes $c \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSEnc}(m_0, \mathsf{sk}_{s,\mathtt{cid}}, \mathsf{pk}_{r,\mathtt{cid}})$

(c) Sends c to the P_{service} for $\mathsf{pk}_{r,\mathtt{cid}}$ from $\mathsf{pk}_{s,\mathtt{cid}}$

(d) Records the entry $(\mathtt{cid}, \mathtt{mid}, c)$ in $\mathbb{M}$

(6) **Honest user sends a message to a corrupt user.** When Sim receives the message $(\mathsf{NotifySendMessageCorrupt}, \mathtt{cid}, \mathtt{mid}, m, P_h, P_c)$ from the ideal functionality, Sim looks up the entry

$$(\mathtt{cid}, P_h, \mathsf{pk}_{h,\mathtt{cid}}, \mathsf{sk}_{h,\mathtt{cid}}, P_c, \mathsf{pk}_{c,\mathtt{cid}}, \cdot)$$

in $\mathbb{T}$ and performs the following:

(a) Computes $c \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSEnc}(m, \mathsf{sk}_{h,\mathtt{cid}}, \mathsf{pk}_{c,\mathtt{cid}})$

(b) Sends c to the P_{service} for $\mathsf{pk}_{c,\mathtt{cid}}$ from $\mathsf{pk}_{h,\mathtt{cid}}$

(c) Records the entry $(\mathtt{cid}, \mathtt{mid}, c)$ in $\mathbb{M}$

(7) **Anonymous honest user receives a message from an honest user.** When Sim receives a set of messages

$$\{(\mathsf{ApproveAnonymousReceiveMessage}, \mathtt{cid}, \mathtt{mid}_i, |m_i|)\}_{i \in [k]}$$

from the ideal functionality, Sim looks up

$$(\mathtt{cid}, \cdot, \mathsf{pk}_{s,\mathtt{cid}}, \mathsf{sk}_{s,\mathtt{cid}}, P_r, \mathsf{pk}_{r,\mathtt{cid}}, \mathsf{sk}_{r,\mathtt{cid}})$$

in $\mathbb{T}$. Additionally, for each message, Sim looks for an entry $(\mathtt{cid}, \mathtt{mid}_i, c_i)$ in $\mathbb{M}$. The ideal functionality authenticates to P_{service} with the identity $\mathsf{pk}_{s,\mathtt{cid}}$ and receives messages $\{a'_j \| c'_j\}_{j \in [k']}$ in return. Sim does the following:

(a) For each message $(\mathsf{ApproveAnonymousReceiveMessage}, \mathtt{cid}, \mathtt{mid}_i, |m_i|)$ and associated entry $(\mathtt{cid}, \mathtt{mid}_i, a_i, c_i)$, if P_{service} sent a message $a'_j \| c'_j$ such that $a'_j = a_i$ and $c'_j = c_i$, sends $(\mathsf{Approve}, \mathtt{mid})$. If no such $a'_j \| c'_j$ exists, Sim sends $(\mathsf{Approve}, \mathtt{mid})$.

(b) For each message $a'_j \| c'_j$, if there does not exist and entry $(\mathtt{cid}, \mathtt{mid}, a'_j, c'_j)$ for some value of $\mathtt{mid}$, Sim decrypts $(m_j, \mathsf{pk}_j) \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSDecVer}(\mathsf{sk}_{s,\mathtt{cid}}, c_i)$. If $\mathsf{pk}_j = \mathsf{pk}_{r,\mathtt{cid}}$, the simulator aborts with an error.

(8) **Non-anonymous Honest user receives a message from an honest user.** When Sim receives the set of messages

$$\{(\mathsf{ApproveReceiveMessage}, \mathtt{cid}, \mathtt{mid}_i, |m|, P_r)\}_{i \in [k]}$$

from the ideal functionality, Sim looks up

$$(\mathtt{cid}, \cdot, \mathsf{pk}_{s,\mathtt{cid}}, \mathsf{sk}_{s,\mathtt{cid}}, P_r, \mathsf{pk}_{r,\mathtt{cid}}, \mathsf{sk}_{r,\mathtt{cid}})$$

in $\mathbb{T}$. Additionally, for each message, Sim looks for an entry $(\mathtt{cid}, \mathtt{mid}_i, c_i)$ in $\mathbb{M}$. The ideal functionality authenticates to P_{service} with the identity $\mathsf{pk}_{r,\mathtt{cid}}$ and receives messages $\{c'_j\}_{j \in [k']}$ in return. Sim does the following:

(a) For each message $(\mathsf{ApproveReceiveMessage}, \mathtt{cid}, \mathtt{mid}_i, |m_i|, P_r)$ and associated entry $(\mathtt{cid}, \mathtt{mid}_i, c_i)$, if P_{service} sent a message c'_j such that $c'_j = c_i$, sends $(\mathsf{Approve}, \mathtt{mid})$. If no such c'_j exists, Sim sends $(\mathsf{Approve}, \mathtt{mid})$.

(b) For each message c'_j, if there does not exist and entry $(\mathtt{cid}, \mathtt{mid}, c'_j)$ for some value of $\mathtt{mid}$, Sim decrypts $(m_j, \mathsf{pk}_j) \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSDecVer}(\mathsf{sk}_{r,\mathtt{cid}}, c_i)$. If $\mathsf{pk}_j = \mathsf{pk}_{s,\mathtt{cid}}$, the simulator aborts with an error.

(9) **Honest user receives a message from a corrupt user.** When Sim receives the message $(\mathsf{ApproveReceiveMessageCorrupt}, \mathtt{cid}, P_s, P_r)$ from the ideal functionality, it looks up

$$(\mathtt{cid}, P_h, \mathsf{pk}_{h,\mathtt{cid}}, \mathsf{sk}_{h,\mathtt{cid}}, P_c, \mathsf{pk}_{c,\mathtt{cid}}, \cdot)$$

in $\mathbb{T}$. Sim authenticates to P_{service} with $\mathsf{pk}_{h,\mathtt{cid}}$ and gets a set of messages $\{c_i\}_{i\in[k]}$ from P_{service}. For each c_i Sim does the following:

(a) decrypts $(m_i, \mathsf{pk}_i) \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSDecVer}(\mathsf{sk}_{h,\mathtt{cid}}, c_i)$. If it fails, the message is dropped.

(b) send the tuple $(\mathtt{cid}, P_c, P_h, m_i)$ to the ideal functionality

Although the simulator is quite involved, the security argument is quite straight forward hybrid argument, starting with the real experiment $\mathcal{H}_0$. In $\mathcal{H}_1$, conversation opening messages between honest parties take the ephemeral secret key instead of the sender's longterm secret key. Due to the ciphertext anonymity of Π_{ssenc}, the distance between $\mathcal{H}_0$ and $\mathcal{H}_1$ is negligible. In $\mathcal{H}_2$, the plaintext contents of messages between honest users are replaced with random messages of the same length. Due to the security of Π_{ssenc}, the distance between $\mathcal{H}_1$ and $\mathcal{H}_2$ is negligible. In $\mathcal{H}_3$, if the service provider delivers a message on behalf of an anonymous honest user that the honest user did not send, the experiment aborts. Due to the authenticity property of Π_{ssenc}, the distance between $\mathcal{H}_2$ and $\mathcal{H}_3$ is negligible. In $\mathcal{H}_4$, if the service provider delivers a message on behalf of a non-anonymous honest user that the honest user did not send, the experiment aborts. Due to the authenticity property of Π_{ssenc}, the distance between $\mathcal{H}_3$ and $\mathcal{H}_4$ is negligible. Finally, in $\mathcal{H}_5$ keys are generated randomly by the simulator instead of the honest parties. Because the keys are sampled at random, the distributions of $\mathcal{H}_4$ and $\mathcal{H}_5$ are the same. $\mathcal{H}_5$ and the simulator above are distributed identically, so the proof is done.

Appendix C

Protocols for Two-Way Sealed Sender Conversations

Notation	Type	Meaning	Anonymous
P_s	User	Sender/Initiator	-
P_r	User	Receiver	-
$(\mathsf{pk}_s, \mathsf{sk}_s)$	Π_{ssenc} Keys	Sender/Initiator key	N
$(\mathsf{pk}_r, \mathsf{sk}_r)$	Π_{ssenc} Keys	Receiver key	N
$(\mathsf{pk}_e, \mathsf{sk}_e)$	Π_{ssenc} Keys	Ephemeral key	Y
$(\mathsf{pk}_{es}, \mathsf{sk}_{es})$	Π_{ssenc} Keys	Sender/Initiator eph. key	Y
$(\mathsf{pk}_{er}, \mathsf{sk}_{er})$	Π_{ssenc} Keys	Receiver eph. key	Y

Figure C.1: Notation for two-way sealed sender protocols

This appendix provides more details for the two-way sealed sender solution discussed in §3.6.3.

Recall how this solution works: after an initiator sends a sealed sender message to the long-term identity of the receiver communicating the sender's ephemeral identity, the receiver generates a fresh, ephemeral identity of their own and sends it to the sender's ephemeral identity via sealed sender. After this initial exchange, the two users communicate using only their ephemeral identities and sealed sender messages, in both directions (two-way).

The protocol proceeds as follows: When some conversation initiator P_s wants to start a conversation with a user P_r, the initiator executes **Initiate Two-Way Sealed Conversation** (see below). P_s starts by generating a keypair $(\mathsf{pk}_{es}, \mathsf{sk}_{es})$ and registering pk_{es} with the service provider. P_s then runs the **Change Mailbox** protocol (see below), which informs the receiver of pk_{es} by sending a message to pk_r. The receiver P_r then generates a keypair $(\mathsf{pk}_{er}, \mathsf{sk}_{er})$ and registers pk_{er} with the service provider. Finally, the P_r runs the **Change Mailbox** protocol, informing P_s about pk_{er} by sending a message to pk_{es}. P_s and P_r communicate using **Send message**, **Open**

Connection, and **Push Message** as in §3.6.2 (for brevity, these protocols have not been replicated below).

Initiate Two-Way Sealed Conversation to P_r:

(1) P_s looks up the P_r's long-term key pk_r

(2) P_s generates keys $(\mathsf{pk}_{es}, \mathsf{sk}_{es}) \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSKeyGen}(1^\lambda)$ and opens a mailbox with public key pk_{es}

(3) P_s runs the subroutine **Change Mailbox**$(P_r, \mathsf{pk}_{es}, \mathsf{sk}_s, \mathsf{pk}_r)$

(4) P_r generates keys $(\mathsf{pk}_{er}, \mathsf{sk}_{er}) \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSKeyGen}(1^\lambda)$ and opens a mailbox with public key pk_{er}

(5) P_r runs the subroutine **Change Mailbox**$(P_s, \mathsf{pk}_{er}, \mathsf{sk}_r, \mathsf{pk}_{es})$

(6) P_s records $(P_r, \mathsf{pk}_{er}, \mathsf{pk}_{es}, \mathsf{sk}_{es})$ and P_r records $(P_s, \mathsf{pk}_{es}, \mathsf{pk}_{er}, \mathsf{sk}_{er})$ in their respective conversation tables

(7) Both P_s and P_r use **send message** to send a read-receipt acknowledgment to pk_{er} and pk_{es} respectively.

Change Mailbox$(P_r, \mathsf{pk}_e, \mathsf{sk}_s, \mathsf{pk}_r)$:

(1) User changing mailbox P_s does the following (note that this user may be the conversation initiator or the conversation receiver)

  (a) encrypts

  $c \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSEnc}(\texttt{"init"} \| \mathsf{pk}_e, \mathsf{sk}_s, \mathsf{pk}_r)$

  (b) connects to the server provider anonymously and sends $c \| \mathsf{pk}_e$ to the service provider addressed to pk_r.

(2) The service provider opens a mailbox with public key pk_e and delivers c to pk_r (sealed sender)

(3) When the other user P_r calls receive message, it decrypts and verifies

$(\texttt{"init"} \| \mathsf{pk}_e, \mathsf{pk}_s) \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSDecVer}(\mathsf{sk}_r, c)$.

Appendix D

Full list of countries / resolvers

Country	Resolver pairs	IPv4 A	IPv4 AAAA	IPv6 A	IPv6 AAAA	Avg.
United States (US)	1228	1.40	0.94	0.49	0.85	0.92
Germany (DE)	753	1.00	0.92	0.75	0.81	0.87
Korea (KR)	632	2.33	1.06	1.24	1.22	1.46
France (FR)	560	0.56	0.47	0.39	0.54	0.49
Russia (RU)	312	5.51	4.78	4.49	4.50	4.82
Iran (IR)	277	25.12	24.49	21.95	21.45	23.25
Viet Nam (VN)	252	1.53	0.89	1.42	0.67	1.13
Taiwan (TW)	248	1.16	1.03	0.86	0.95	1.00
India (IN)	226	1.27	1.52	1.04	1.16	1.25
Canada (CA)	199	0.74	0.36	0.23	0.30	0.41
United Kingdom (GB)	196	1.14	0.92	0.77	0.97	0.95
China (CN)	194	29.27	32.32	28.41	32.08	30.52
Thailand (TH)	186	8.25	1.18	1.13	0.93	2.87
Brazil (BR)	160	2.67	1.67	1.99	2.08	2.10
Japan (JP)	152	1.12	1.18	0.56	1.01	0.97
Mexico (MX)	150	3.78	2.05	1.75	1.94	2.38
Turkey (TR)	114	1.29	0.96	1.27	1.02	1.14
Netherlands (NL)	97	1.17	0.78	0.72	0.70	0.85
South Africa (ZA)	93	2.41	1.60	1.18	1.33	1.63
Australia (AU)	72	1.56	0.70	0.95	0.66	0.97
Hong Kong (HK)	67	7.00	5.57	4.91	5.24	5.68
Chile (CL)	65	2.67	0.70	1.10	0.79	1.32

Switzerland (CH)	60	0.32	0.34	0.28	0.34	0.32
Indonesia (ID)	56	6.46	2.99	2.41	2.26	3.53
Lithuania (LT)	52	0.95	0.80	0.51	0.78	0.76
Singapore (SG)	50	1.76	0.86	0.64	0.78	1.01
Malaysia (MY)	50	4.92	2.03	1.27	1.35	2.39
Spain (ES)	49	0.91	0.75	1.40	1.94	1.25
Poland (PL)	48	2.00	1.13	2.90	0.96	1.75
Argentina (AR)	47	5.20	3.51	2.22	1.28	3.05
Romania (RO)	44	2.30	1.03	0.83	0.93	1.27
Czechia (CZ)	41	1.60	1.06	0.50	0.51	0.91
Italy (IT)	38	2.36	2.10	2.25	2.52	2.31
Ukraine (UA)	35	6.49	3.05	2.64	2.87	3.76
Sweden (SE)	34	0.36	0.29	0.51	0.25	0.35
Finland (FI)	34	0.68	0.59	0.43	0.52	0.56
Belgium (BE)	31	1.56	1.26	0.95	0.95	1.18
Bulgaria (BG)	30	3.24	2.91	1.15	1.06	2.09
Bangladesh (BD)	29	6.42	1.28	0.89	0.81	2.35
Colombia (CO)	27	3.39	3.28	1.45	1.14	2.31
Saudi Arabia (SA)	24	2.21	1.81	1.60	1.60	1.81
Pakistan (PK)	23	3.74	1.59	4.64	1.86	2.96
Hungary (HU)	22	0.29	0.27	0.23	0.31	0.27
Ecuador (EC)	21	3.29	2.94	0.38	0.77	1.84
Greece (GR)	20	0.49	1.15	0.41	0.53	0.64
Kazakhstan (KZ)	19	2.90	2.01	2.42	2.21	2.38
Philippines (PH)	18	4.34	2.36	2.45	1.96	2.78
Norway (NO)	16	0.27	0.32	0.31	0.40	0.32
Slovakia (SK)	14	0.22	0.24	0.22	0.26	0.24
Egypt (EG)	14	5.59	4.12	3.05	3.47	4.06
Denmark (DK)	13	1.68	1.26	0.93	1.05	1.23
Portugal (PT)	11	2.16	0.39	0.23	0.27	0.76
Peru (PE)	11	2.04	0.81	0.69	0.65	1.05
Nigeria (NG)	9	3.42	1.38	1.65	1.43	1.97
Austria (AT)	8	0.11	0.19	0.07	0.14	0.13
Kenya (KE)	8	7.79	1.40	1.14	1.07	2.85
Moldova, Rep. of (MD)	7	5.46	3.58	2.90	3.04	3.75

Serbia (RS)	7	0.18	0.12	0.14	0.10	0.14
Nepal (NP)	7	1.10	1.22	0.80	0.98	1.03
Slovenia (SI)	7	3.48	0.24	0.14	0.16	1.01
New Zealand (NZ)	7	4.30	0.20	0.46	0.16	1.28
Utd Arab Emirates (AE)	6	1.61	0.86	1.10	0.89	1.11
Estonia (EE)	6	1.61	0.42	0.28	0.42	0.68
Costa Rica (CR)	6	8.57	2.45	2.19	2.10	3.83
Uruguay (UY)	6	0.70	0.42	0.23	0.19	0.39
Macao (MO)	6	4.44	4.48	4.44	4.51	4.46
Mongolia (MN)	5	1.88	0.67	0.59	0.70	0.96
Libya (LY)	5	1.40	0.76	0.39	0.42	0.74
Sudan (SD)	5	4.59	4.48	2.38	2.32	3.45
Venezuela (VE)	5	6.67	1.62	0.81	0.73	2.46
Belarus (BY)	5	1.01	0.73	0.45	0.62	0.70
Panama (PA)	5	3.75	3.98	0.31	0.45	2.12
Armenia (AM)	5	0.87	0.64	0.59	0.73	0.71
Uzbekistan (UZ)	4	0.53	0.85	0.82	0.74	0.74
Belize (BZ)	4	5.92	0.53	0.42	0.39	1.81
Latvia (LV)	4	0.11	0.11	0.14	0.14	0.12
Luxembourg (LU)	4	1.86	0.07	0.07	0.21	0.55
Albania (AL)	4	0.70	0.56	0.42	0.42	0.53
Iraq (IQ)	4	3.57	2.45	2.14	2.14	2.57
Bosnia & Herzegovina (BA)	4	2.21	0.70	0.63	0.70	1.06
Guatemala (GT)	4	0.60	0.56	0.39	0.21	0.44
Jordan (JO)	4	0.28	0.21	0.14	0.42	0.26
Bolivia (BO)	4	1.16	0.53	0.32	0.28	0.57
Lebanon (LB)	3	0.61	0.33	0.28	0.33	0.39
Dominican Republic (DO)	3	4.76	0.75	0.28	0.47	1.56
Honduras (HN)	3	1.49	1.45	0.65	0.61	1.05
Croatia (HR)	3	0.14	0.23	0.19	0.14	0.18
Afghanistan (AF)	3	2.24	1.12	1.91	1.82	1.77
Cambodia (KH)	3	3.17	0.90	0.47	0.43	1.24
El Salvador (SV)	2	0.84	0.35	0.14	0.21	0.39
Nicaragua (NI)	2	3.78	0.28	0.35	0.07	1.12
Lao (LA)	2	9.24	0.56	0.21	0.28	2.57

Oman (OM)	2	5.46	0.70	0.42	0.70	1.82
Israel (IL)	2	2.73	2.94	2.73	2.66	2.77
Guam (GU)	2	8.05	0.35	0.00	0.35	2.19
Georgia (GE)	2	1.19	1.26	1.40	0.98	1.21
Trinidad & Tobago (TT)	2	0.35	0.14	0.07	0.00	0.14
Chad (TD)	2	2.73	1.05	0.63	1.12	1.38
North Macedonia (MK)	2	9.52	6.44	4.69	4.83	6.37
Ethiopia (ET)	2	0.14	0.28	25.56	0.21	6.55
Uganda (UG)	2	0.49	0.49	0.56	6.65	2.05
Tunisia (TN)	2	0.35	0.42	0.35	0.49	0.40
Cyprus (CY)	2	4.83	0.70	0.35	0.21	1.52
Paraguay (PY)	1	3.26	1.56	1.28	1.13	1.81
Kyrgyzstan (KG)	1	0.57	0.71	0.43	0.85	0.64
Seychelles (SC)	1	0.28	0.43	0.28	0.57	0.39
Global	7,428	3.10	1.83	1.77	1.60	2.07

Table D.1: Full list of measured rates of DNS censorship across both query types and network interfaces. Rates are expressed as the percentage of censored DNS queries over total number of DNS queries sent. Darker shaded cells indicate a higher rate of DNS censorship (compared to the country's average) and lighter shaded cells indicate a lower rate of DNS censorship. The average rate of censorship for a country is computed across all four IP/query combinations. The global row contains the mean of each column. These means weigh the contribution of each country equally, rather than weighted by the number of resolvers used in tests.

Appendix E

Censorship rates by network end-point types

Country	Resolver pairs	IPv4 A	IPv4 AAAA	IPv6 A	IPv6 AAAA	Avg.
Corporate censorship (>25 resolvers)						
Iran (IR)	48	25.50	24.66	22.34	21.69	23.55
Hong Kong (HK)	37	6.20	5.44	4.55	5.38	5.39
Russian Federation (RU)	117	4.67	4.36	4.01	4.00	4.26
Thailand (TH)	43	9.28	1.18	0.99	0.95	3.10
Malaysia (MY)	30	6.07	2.25	0.97	1.10	2.60
Spain (ES)	28	1.26	0.97	2.24	2.97	1.86
Korea (KR)	177	2.49	1.37	1.38	1.57	1.70
Australia (AU)	35	2.53	1.16	1.77	1.14	1.65
South Africa (ZA)	74	2.09	1.62	1.10	1.31	1.53
Japan (JP)	79	1.71	1.82	0.90	1.57	1.50
Viet Nam (VN)	40	2.16	1.01	0.85	0.86	1.22
Brazil (BR)	34	1.50	1.10	1.01	1.17	1.19
India (IN)	101	1.19	1.76	0.81	0.99	1.19
Chile (CL)	53	2.10	0.73	1.12	0.77	1.18
United States (US)	757	1.62	1.16	0.54	0.96	1.07
Singapore (SG)	41	1.56	0.85	0.65	0.78	0.96
United Kingdom (GB)	186	1.07	0.94	0.78	0.97	0.94
Netherlands (NL)	71	1.08	0.94	0.84	0.85	0.92
Germany (DE)	717	0.98	0.91	0.75	0.80	0.86
Lithuania (LT)	51	0.95	0.80	0.51	0.78	0.76
Finland (FI)	31	0.74	0.64	0.47	0.56	0.60

France (FR)	470	0.60	0.45	0.40	0.46	0.48
Canada (CA)	165	0.84	0.39	0.26	0.33	0.45
Non-corporate censorship (>25 resolvers)						
China (CN)	179	29.46	32.68	28.82	32.58	30.89
Iran (IR)	229	25.04	24.46	21.87	21.40	23.19
Hong Kong (HK)	30	8.00	5.74	5.35	5.08	6.04
Russian Federation (RU)	195	6.02	5.03	4.78	4.80	5.16
Argentina (AR)	34	5.47	4.22	2.56	1.31	3.39
Indonesia (ID)	40	6.27	2.84	2.17	2.04	3.33
Thailand (TH)	143	7.94	1.18	1.17	0.93	2.80
Mexico (MX)	140	3.90	2.03	1.70	1.89	2.38
Bangladesh (BD)	29	6.42	1.28	0.89	0.81	2.35
Brazil (BR)	126	2.99	1.82	2.25	2.33	2.35
Korea (KR)	455	2.26	0.94	1.19	1.09	1.37
Poland (PL)	32	1.71	0.91	1.93	0.74	1.32
Romania (RO)	38	2.32	1.10	0.89	0.97	1.32
India (IN)	125	1.34	1.32	1.23	1.30	1.30
Belgium (BE)	27	1.59	1.25	0.92	0.91	1.17
Viet Nam (VN)	212	1.42	0.86	1.52	0.64	1.11
Taiwan (TW)	242	1.16	1.04	0.87	0.96	1.01
Germany (DE)	36	1.31	1.02	0.81	0.89	1.01
Turkey (TR)	89	1.17	0.86	1.29	0.70	1.01
United States (US)	471	1.05	0.58	0.42	0.68	0.68
Netherlands (NL)	26	1.44	0.37	0.41	0.31	0.63
France (FR)	90	0.39	0.57	0.32	0.94	0.56
Japan (JP)	73	0.49	0.50	0.19	0.40	0.39
Australia (AU)	37	0.64	0.26	0.17	0.20	0.32
Switzerland (CH)	46	0.27	0.35	0.28	0.37	0.32
Canada (CA)	34	0.27	0.25	0.09	0.14	0.19
Censorship on resolver pairs with native IPv6 (> 25 resolvers)						
China (CN)	163	29.38	32.94	28.95	32.84	31.03
Russian Federation (RU)	57	3.96	4.01	3.89	4.11	3.99
Thailand (TH)	74	8.86	1.45	1.60	1.34	3.31
Mexico (MX)	103	4.58	2.52	2.24	2.55	2.97
Brazil (BR)	66	3.15	2.12	3.11	3.37	2.94

India (IN)	51	0.99	1.13	1.11	1.36	1.15
Viet Nam (VN)	27	1.29	0.77	1.19	1.32	1.14
Japan (JP)	111	1.02	1.34	0.53	1.14	1.01
Taiwan (TW)	230	1.09	1.01	0.84	0.93	0.97
Germany (DE)	510	1.11	1.02	0.84	0.86	0.96
United Kingdom (GB)	69	0.67	0.76	0.66	1.35	0.86
Singapore (SG)	26	0.95	0.80	0.58	0.80	0.78
Australia (AU)	35	1.58	0.53	0.40	0.41	0.73
Belgium (BE)	26	0.65	0.88	0.61	0.75	0.72
United States (US)	762	0.74	0.85	0.36	0.91	0.72
France (FR)	196	0.52	0.65	0.55	0.94	0.66
Lithuania (LT)	45	0.44	0.74	0.43	0.73	0.59
Canada (CA)	53	0.69	0.37	0.23	0.37	0.41
Netherlands (NL)	54	0.35	0.37	0.51	0.39	0.41
Switzerland (CH)	54	0.34	0.37	0.30	0.38	0.35

Censorship on resolver pairs with 6to4 IPv6 addresses (> 25 resolvers)

China (CN)	31	28.78	29.06	25.65	28.07	27.89
Iran (IR)	272	25.26	24.59	21.96	21.44	23.31
Hong Kong (HK)	54	7.41	5.20	4.94	4.78	5.58
Russian Federation (RU)	255	5.86	4.95	4.63	4.59	5.01
Ukraine (UA)	31	6.12	2.72	2.45	2.62	3.48
Indonesia (ID)	51	5.66	2.35	1.83	1.52	2.84
Argentina (AR)	44	4.65	3.21	1.87	0.90	2.66
Thailand (TH)	112	7.84	1.00	0.82	0.67	2.58
Malaysia (MY)	30	5.82	2.28	0.97	1.12	2.55
Bangladesh (BD)	27	6.29	1.35	0.89	0.78	2.33
Poland (PL)	44	2.14	1.18	3.12	1.00	1.86
South Africa (ZA)	93	2.41	1.61	1.18	1.35	1.64
Brazil (BR)	94	2.34	1.35	1.20	1.18	1.52
Korea (KR)	628	2.34	1.05	1.25	1.23	1.47
Spain (ES)	39	0.99	0.80	1.63	2.31	1.43
Netherlands (NL)	43	2.21	1.30	0.99	1.10	1.40
Chile (CL)	63	2.63	0.71	1.12	0.80	1.31
India (IN)	175	1.36	1.63	1.02	1.11	1.28
United States (US)	466	2.48	1.08	0.70	0.76	1.26

Australia (AU)	37	1.54	0.86	1.46	0.89	1.19
Turkey (TR)	112	1.30	0.96	1.29	1.04	1.15
Viet Nam (VN)	225	1.57	0.91	1.45	0.60	1.13
Mexico (MX)	47	2.02	1.00	0.67	0.60	1.07
United Kingdom (GB)	127	1.40	1.01	0.82	0.78	1.01
Japan (JP)	41	1.39	0.76	0.64	0.66	0.86
Germany (DE)	243	0.78	0.72	0.58	0.72	0.70
France (FR)	364	0.59	0.47	0.31	0.41	0.44
Canada (CA)	146	0.76	0.36	0.23	0.27	0.41

Table E.1: Measured rates of censorship for countries with more than 25 resolver-pairs with corporate, non-corporate, native IPv6, and 6to4-bridged IPv6 end-points. Rates are expressed as the percentage of censored DNS queries. Darker shaded cells indicate a higher rate of DNS censorship (compared to the country's average computed over all network and query types).