CoverUp: Privacy Through "Forced" Participation in Anonymous Communication Networks

David Sommer *ETH Zurich*
Aritra Dhar *ETH Zurich*
Luka Malisa *ETH Zurich*
Esfandiar Mohammadi *ETH Zurich*
Daniel Ronzani *Ronzani Schlauri Attorneys*
Srdjan Capkun *ETH Zurich*

Abstract

The privacy guarantees of anonymous communication networks (ACNs) are bounded by the number of participants.As a consequence, an ACN can only achieve strong privacy guarantees if it succeeds in attracting a large number of active users. Vice versa, weak privacy guarantees renders an ACN unattractive, leading to a low number of users. In this work, we show how to break this vicious circle. We develop CoverUp, a system that "forces" visitors of highly accessed websites to become involuntary participants of an ACN. CoverUp leverages basic browser functionality to execute server-served JavaScript code and to open remote connections to connect all website visitors to an ACN (which we instantiate by a mix server). We build two applications on top of CoverUp: an anonymous feed and a chat. We show that both achieve practical performance and strong privacy guarantees. Towards a network-level attacker, CoverUp makes voluntary and involuntary participants indistinguishable, thereby providing an anonymity set that includes all voluntary and involuntary participants (i.e., all website visitors). Given this, CoverUp provides even more than mere anonymity: the voluntary participants can hide the very intention to use the ACN. As the concept of forced participation raises ethical and legal concerns, we discuss these concerns and describe how these can be addressed.

1 Introduction

Many privacy-enhancing technologies, in particular anonymous communication networks (ACNs) as a key building block, suffer from a lack of a sufficient number of participants. Without high user participation, and therefore without significant *cover traffic*, anonymity networks become vulnerable to traffic analysis attacks or suffer from performance loss in terms of their throughput and latency. The only ACN with a high number of participants (around 1.5 million users [52]) is Tor [35]. Yet, Tor is known to be prone to traffic analysis attacks, as illustrated by a wide variety of passive [40, 58, 51] and active [59, 46, 55] traffic pattern attacks. While other ACNs [32] have been proposed that are even secure against global attackers, they suffer from a low number of participants, since even a perfect ACN can at most hide a user among all participating users. These ACNs are in a vicious circle: the lack of participants leads to low degree of anonymity, and a low degree of anonymity makes these ACNs unattractive for users. Even if the number of users produces a large anonymity set, the mere use of a system can already raise suspicion, which can serve as a powerful deterrent for a wide range of users.

In this work, we show how these issues can be addressed through "forced" user participation. The act of "forcing" users to participate in an ACN enables us to not only achieve strong anonymity properties via a large anonymity set, thus helping in bootstrapping an ACN, but to even hide the intention of those that voluntarily participate in the ACN.

We design a system, called `CoverUp` that triggers users to participate in a centralized, constant-rate mix[1] by leveraging the basic functionality of their browsers to execute (JavaScript) code served by the websites that they visit. `CoverUp` is intended to be used on university or news sites that act as entry servers and serve `CoverUp` code to the end-users' browsers (via an iframe-request to a `CoverUp` server) and therefore make them participate in the ACN. Visitors of these entry servers' websites therefore become (involuntary) participants of an ACN and create cover traffic in which the voluntary participants can hide their traffic. In addition to the JavaScript code that is served to the visitors of entry servers' websites, `CoverUp` includes a browser extension that is used by the voluntary participants of the anonymity network.

[1] While we implement a centralized mix in our case study, our approach can conceptually be extended to a set of distributed mixes or to practically any ACN.

We use CoverUp in two representative applications: a feed (e.g., a Wikileaks newsfeed) Wikileaks documents), and a chat. We evaluate the performance and the privacy guarantees in all three applications. We show that they can be practical, provide a high degree of user anonymity, and enable a user to plausibly deny that it actively uses CoverUp. In particular, we show through our prototypical implementations that a strong adversary cannot distinguish voluntary participants (who additionally installed the CoverUp extension[2]) from involuntary participants, which only run the CoverUp code in their browsers. We assume a strong attacker that can fully observe the network traffic and can control the entry server. The feed can tolerate a malicious CoverUp server (which serves the CoverUp JavaScript code), while the chat requires a trusted CoverUp server.

In the prototype CoverUp the mix servers are implemented as a JAVA servlet API on a Apache Tomcat web server. The CoverUp external application and the browser extension is implemented using Java and Mozilla Firefox web extension API, respectively. After a randomized delay of the first message to hide leakage, the CoverUp downlink and uplink rate of our prototype is 20 Kbit/s and the latency is 36.55 seconds within one site. We evaluate the privacy guarantees provided by CoverUp by analyzing differences in the network delays between voluntary and involuntary users. We show that the attacker can distinguish voluntary and involuntary users with an accuracy of 56.3% with an year's worth of observations, for a realistic usage pattern.[3]

Ethical and legal concerns. "Forcing" participation might appear intrusive, but it can be implemented with commonly-used browser functionality and such that the users are informed about their participation and/or asked for explicit consent (see Appendix 8 for more). Involuntary allocation of resources is nothing unexpected for a visitor of a webpage; it is already done by advertisements or QoS scripts, such as Google Analytics. In particular, webpages that incorporate CoverUp would not cause unexpected behavior on a visitor's browser. The computational overhead of CoverUp is negligible but the traffic overhead for a visitor would be around 7.5MB per day, which is negligible compared to the data load of video streaming services. Our work received a formal approval of our institute's ethics commission. We finally discuss the legal implications for involuntary participants and for the websites that act as a entry server. We elaborate on the liability of the entry server across different jurisdictions. We also discuss that our system is easy to deploy for proxies and that there could even be incentives for participation for both proxies and users. We even argue that, with an increasing demand for privacy protection [21, 22], websites could increase their reputation by supporting a privacy service from their pages, and in turn privacy-supporting visitors would visit those pages. We analyzed to the best possible extent the legal implications of our solutions, but as for similar technologies (e.g., cookies) final judgements are made by the courts in the different jurisdictions.

In summary, we make the following contributions.

- We introduce a novel concept of *privacy by "forced" participation* where we argue that popular services can help in increasing user participation in privacy projects and more specifically in creating cover traffic for anonymous communication. As a result, the anonymity set includes all visitors of a collaborating popular service, thereby achieving strong anonymity.

- We design CoverUp, a web-based system that helps in creating cover traffic for anonymity networks (see Figure 1).

- We instantiate CoverUp in the context of a feed and chat application and show that it offers practical performance (see Table 1) and strong privacy guarantees (see Theorem 1).

- We evaluate the privacy of CoverUp and experimentally evaluate the timing leakage of CoverUp on Windows for, both, voluntary and involuntary participants (see Figure 6 and Figure 7).

- We discuss the legal for the service that acts as a entry server and for the involuntary participants. We elaborate on the liability of the entry server and discuss why the involuntary participants in our view should not experience any legal issues.

Outline of the paper. Section 2 describes the problem. Section 3 provides detailed system design and attacker model of CoverUp. Section 4.1 elaborates on implementation details. Section 4.2 discusses the overhead that CoverUp causes and presents the latency and bandwidth of CoverUp. Section 5 defines a privacy notion that captures involuntary being indistinguishable from voluntary participants and the connections to anonymity notions from the literature. Section 6 analyzes the privacy of CoverUp and shows that the direct privacy leakage of CoverUp solely depends on its timing leakage. Section 7 experimentally evaluates this timing leakage and interprets the privacy results. Section 8, 9, and 10 discuss the

[2]We assume that voluntary users obtain and install the extension in an out-of-band fashion or in another way not observed by the adversary.

[3]As the usage pattern has to be adjusted to the involuntary users, we assume that a usage pattern of 4 times each working day for 10 minutes each time (see Section 6.2).

ethical and legal aspects of CoverUp, and the deployment of CoverUp, respectively. Section 11 discusses related research, and Section 12 concludes the paper and outlines future work.

2 Problem description

For many privacy-enhancing systems, anonymous communication is a key building block. The anonymity that an anonymous communication network (ACN) can provide is limited by the number of participants. In this work, we define and study the following main problem:

> Can an anonymous communication network be strengthened by "forced" participation? What privacy guarantees and performance can such an ACN provide?

It is clear that increased participation in an ACN increases privacy. However, increasing privacy in an ACN through forced participation involves some challenges. The first challenge is to ensure that involuntary participants can become a part of the ACN without any damage to their system. Therefore, here, under "forced" participation we don't mean that users' systems are compelled into participation by infection and malware. Instead, we assume, and show through our solution, that users can be made to participate in an ACN through the legitimate use of existing software and interactions (in our solution, through their browsers). The second challenge is that the involuntary participants should not be distinguishable from voluntary participants. If an attacker can tell involuntary participants apart from the voluntary participants, increased involuntary participation will not have an effect on the privacy properties of the ACN. If, however, the involuntary participants appear just like voluntary participants for an attacker, the resulting system will not only have in a larger anonymity sets but will also provide plausible deniability for voluntary participants. Namely, the voluntary participant can always claim that it was made to participate in the ACN and is not its active user (i.e., that it is an involuntary user).

The requirement that involuntary participants have to look the same as voluntary participants makes this problem particularly challenging. We have to be able to run code on the machine of an involuntary participant, without damaging her system. At the same time, we have to develop a toolchain that enables voluntary participants to use the ACN, while carefully ensuring that they produce the same observable communication patterns. Since such a toolchain might introduce additional computations, it can be prone to producing an observable delay.

To highlight why this is a new type of problem, we contrast it to anonymous communication systems based on covert channels. Covert channels use a piggyback approach to transport data, they depend on existing data streams, resulting in a dependency of the piggybacked system for latency, throughput and privacy. An ACN based on forced participation can create its own cover traffic and therefore control (i.e. tune) the achieved privacy, latency and throughput. All other things equal, such an ACN will also result in higher throughput and lower latency. However, the privacy properties that a forced participation systems achieve differ from those of covert channels. While some covert channels can hide whether communication took place, and thus achieve full deniability, forced participation systems make voluntary and involuntary participants indistinguishable and therefor provide plausible deniability. Section 11 thoroughly discusses the relationship to covert channels such as [50, 45, 38, 60].

3 CoverUp

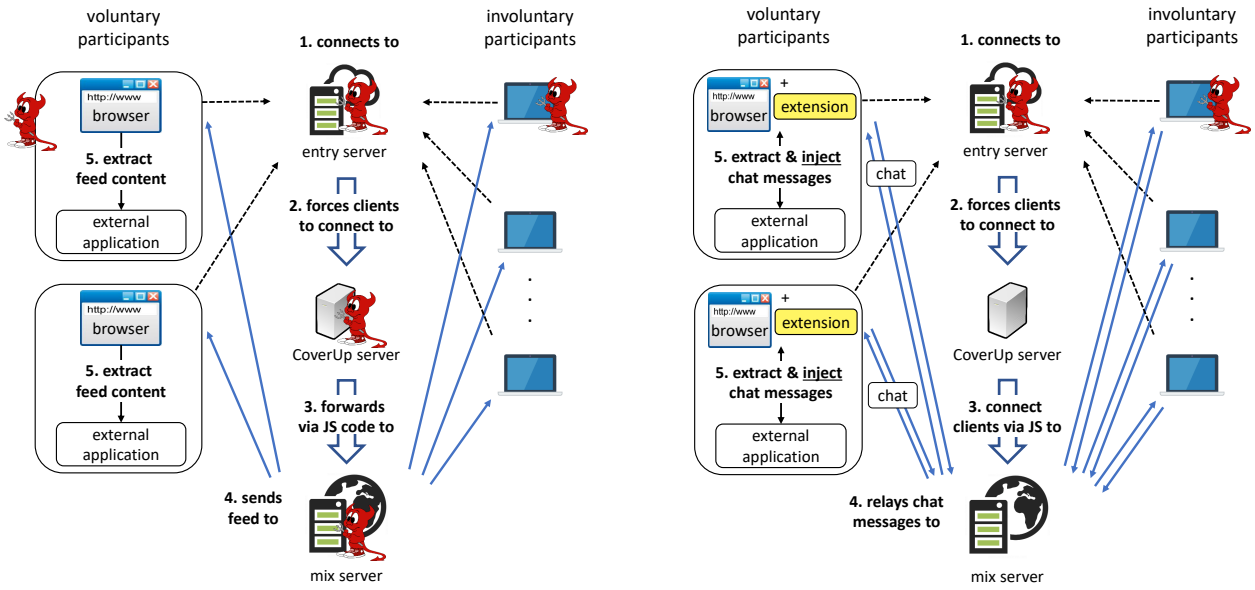
We address the challenges involved in forcing participation by developing a prototype system, which we name CoverUp. Section 3.1 presents the uni-directional channel and how it is used to implement a feed. Section 3.2 shows how this design can be extended to implement a bi-directional channel.

Our system leverages common and widely-used JavaScript-functionality of browsers to cause visitors of a cooperating web site to produce cover traffic for CoverUp users. We call this website the *entry server* and it could be a university, a knowledge site, or a news site. As depicted in Figure 1a (Step 1), this entry server includes in its webpage a request to a dedicated server (the CoverUp *server*). The CoverUp server responds with a JavaScript code snippet (Step 2), which triggers the browsers of the visitors (Step 3) to produce constant rate cover traffic to a dedicated server (the *mix server*). The mix server responds to all parties with the feed (Step 4). The voluntary participants of CoverUp extract the content, e.g., from the feed, via an external application (Step 5). We assume that this external application is retrieved out of band. Involuntary participants do not have an external application installed and will therefore not retrieve the feed; instead, they only generate cover traffic via the JS code served by the CoverUp server.

The privacy provided by CoverUp is tied to the browsing behavior of the involuntary participants. Section 6.2 discusses the resulting limitations and their implications.

3.1 Uni-directional channel (feed)

This set-up constitutes a uni-directional channel, over which the mix server can send information to voluntary participants. The browser stores the data received by the



(a) Uni-directional channel: feed.

(b) Bi-directional channel: chat.

Figure 1: Main components of `CoverUp`. (a) `CoverUp` uni-directional channel enables participants to anonymously download **feeds**, while assuming no trust in any of the servers that they connect to. (b) `CoverUp` bi-directional channel supports **anonymous chat** with minimal trust assumptions on the `CoverUp` and the mix server. The mix server only makes sure that the participant's communication is properly mixed and therefore cannot be subject to traffic correlation attacks.

mix server into a database (the `localStorage`), out of which the external application retrieves the content. With this uni-directional channel, we implement a continuous data feed (in the spirit of a broadcast), which is suited to send information for which a user does not want to be caught reading it (e.g., sensitive medical information, leaked documents, or a leaked e-mail list of an incriminating web service). Our system tolerates a global yet remote network-level attacker (all connections are TLS-secured) that controls all parts of the system (depicted by the red devils in Figure 1a) except for the computer of some the voluntary participants. Additionally, we assume that the content of the feed is signed by an external party for which the external application has the verification key. This external party is also trusted. We defend against a global attacker's traffic correlation capabilities by producing traffic at a constant rate.

We cryptographically protect involuntary participants from accidentally storing parts of the feed's potentially controversial content on their disc by cryptographically ensuring (via an all-or-nothing scheme [53]) that involuntary participants – without actively trying to – do not have sufficient information from which the content of the feed can potentially be reconstructed. This protection is important from a legal perspective (see Section 9).

Dataflow of the uni-directional channel. Figure 1a depicts a high level system design and data flow of the uni-directional channel and the feed application.

1. A voluntary or involuntary participant requests a website from an entry server.

2. The entry server delivers a web-page that contains an iframe-request for the `CoverUp` server. The `CoverUp` server serves the iframe which in turn contains `CoverUp`'s JavaScript snippet. This code gets executed by the participant's web browser.

3. This JavaScript code asks the mix server to deliver a droplet data packet from a randomly chosen fountain or a fountain details list containing description of fountains at the mix server, depending on the page context of the entry server. The mix server delivers a droplet or fountain table depending on the request of the previous step. The script saves the response packet to the web browser's cache database file.

The involuntary participant does not execute any operation beyond this.

4. The voluntary participant installs the external application (received out of band) which periodically checks for a new feed data, i.e., droplet data in the web browser persistent cache, and copies the data into it's own local storage.

5. After accumulating a sufficient amount of droplets, the external application assembles the entire feed ci-

pher text which is encrypted by an all-or-nothing scheme. The external decrypts the message and verify digital signature signed by the mix server to check data authenticity.

3.2 Bi-directional channel (chat)

Based on this uni-directional channel, we construct a bi-directional channel, which enables a voluntary participants to additionally send data over `CoverUp`. As depicted in Figure 1b, the bi-directional channel requires a browser extension (assumed to be retrieved out of band) that modifies the outgoing messages (Step 5), produced by the JavaScript code snippet in the participants' browser. With this bi-directional channel, we implement a chat application, where the mix server relays chat messages between voluntary participants. For this application, it is crucial that the voluntary participants send messages when the unmodified JavaScript (JS) code snippet has been sent by the `CoverUp` server.

Concerning the trust assumptions, we need to trust the integrity check for the JS code snippet, since otherwise an attacker could introduce malicious modifications into the JS code that would enable it to identify voluntary participants. To enable the browser extension to efficiently check the integrity of the JavaScript snipped (which is crucial to prevent timing leakage), we trust the `CoverUp` server in this application. The browser extension simply checks whether the origin of the JavaScript code snippet is as expected. Alternatively, we can tolerate a malicious `CoverUp` server if we check the integrity of the JavaScript code byte for byte, which produces a significant delay. We did not choose this alternative, since – as we show in Section 4.2 and 7 – the efficiency of `CoverUp` directly correlates with the privacy that `CoverUp` achieves.

To gain efficiency, the voluntary participants sends to the mix server the recipient in plaintext. The mix server then relays the chat message to the corresponding receiver. Thus, we trust the mix server. Alternatively (and not done in the prototype), `CoverUp` could implement a bi-directional broadcast channel, where the messages sent by the voluntary participants for the `CoverUp` server are indistinguishable from the messages of the involuntary participants. This variation vastly reduces uplink bandwidth but can tolerate a malicious mix server (see Section 6 for a thorough discussion).

Dataflow of the bi-directional channel.

1. The voluntary participant composes an interactive request x on the external application. x also contains a unique marker to indicate that it is a request for bi-directional communication.

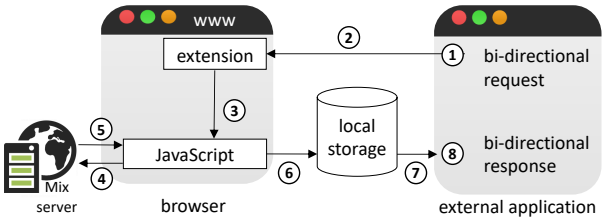


Figure 2: A snapshot of Bi-directional channel from a voluntary participants perspective. The application sends a bi-directional request (encrypted chat) via the `CoverUp` extension to the mix server and receives a response (response chat).

2. The application sends x to the browser extension.
3. The browser extension sets x to the JavaScript snippet to send it as the request payload to the mix server.
4. The snippet sends a request droplet to the mix server with x as the payload. Prior to that, the snippet establishes a TLS session with the mix server.
5. Upon receiving x, mix server responds with a bi-directional data r,
6. The JavaScript snippet stores the response r to the browser's persistent storage.
7. The external application polls the persistent storage in regular intervals.
8. The external application displays r to the voluntary participant on it's interface.

Apart from chat, we also implement interactive browsing mechanism using the bi-directional channel which can be viewed as an enhancement over the feed.

`CoverUp` implements a communication infrastructure on top of HTTP communication that is secure against traffic correlation attacks and even hides whether a user intended to participate, which in turn implies strong privacy properties (see Section 6 for detailed discussion).

4 CoverUp's implementation

This section explains the design choices in and the details of the implementation of the prototype (Section 4.1) and discusses the overhead and performance of the `CoverUp` prototype (Section 4.2).

4.1 CoverUp's implementation

We implemented a prototype and made it available under `http://coverup.tech`. The `CoverUp` implementation consists of five components: a `CoverUp` server, a mix server, an external application, a browser extension, and a short JavaScript code snippet. The `CoverUp` server and

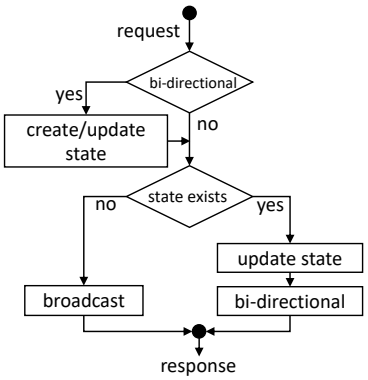


Figure 3: Mix server data flow for bi-directional (chat) channel request. The state denotes to a unique value which corresponds to a specific voluntary user.

the mix server is implemented as a JAVA Servlet, running on an Apache Tomcat web server. The external application is written in JAVA. The browser extension is implemented for Firefox using the JavaScript WebExtensions API. The JavaScript code snippet which fetches feed/interactive data is kept at the `CoverUp` server. The external application and the server implementation consists of about 11 KLOC and the browser extension of about 200 LOC.

We make the following four assumptions about the browser, which are in line with Firefox's explicitly stated security policies. First, iframes are isolated, which we need for the code integrity of `CoverUp`'s JavaScript snippet. Once we checked whether an iframe is loaded from the correct source and the source is in another domain than the parent page, there should not be any way that the parent page of the iframe can change the code, except by changing the source of the iframe [41]. Second, a JS code that is supplied from an arbitrary source is not able to read from or write to another context of a different domain source without its consent. Third, the JS code can only access the browser's localStorage cache and store a limited amount of data. Fourth, the JS code cannot read or write data created by another JavaScript code which originates from a different origin. This property is known as the "same-origin-policy" [42], and all modern browsers publicly claim to enforce this policy.

The system is parametric in the payload size and the droplet request rate. The payload size and request rate is the same, irrespective of the response type (be it a feed, a browsing, or a chat request or response). Our prototype implementation uses a payload size of 75 KB and droplet request rate of once a minute. Section 4.2 evaluates our choices for these system parameters. Increasing the payload increases the traffic overhead, in particular of involuntary user's, and reducing the request rate reduces the latency but decreases the privacy (see Section 7.5). Hence, there is a natural trade-off between the latency and privacy and the amount of traffic overhead cause and bandwidth (i.e., throughput) of the system.

As a next step, we describe the implementation of the applications of our case studies: feed and chat.

4.1.1 Feed

A voluntary user would install an external application, which polls from the firefox local storage cache file. We use fountain codes to deliver the feed data. The mix server keeps such droplets on disk and dispatch them as soon as it receives a request(from both voluntary and involuntary participants). The cache file contains both the table of fountains and the droplets. The `CoverUp` external application reassembles droplets on regular basis and show them when they are ready.

4.1.2 Chat

The messaging protocol is an application on top of the bi-directional channel. The implementation involves indexing the messages as POP (post office protocol [20]) where the indexing is done by the public addresses of the clients. This public address is derived from the *curve25519* public keys (first 48 bits of hashed public key). The chat application assumes that the user added all long term public keys of all his trusted peers. For the cryptographic protection for the messages, the application computes a shared secret from the long term key pairs, e.g., for a prime group generator g, a secret key a of the user and the public key g^b of the recipient the shared secret would be $(g^b)^a = g^{ab}$. While this prototype, in particular, does not achieve forward secrecy, there is conceptually no problem with including a forward-secret key exchange scheme into the chat.

Whenever a new message arrives from a source address, the mix server keeps the message to the index of the destination address. When a uni-directional or a bi-directional request arrives from the destination address, the mix server delivers the message as the response and removes the message from the previously kept index location.

4.2 Overhead, latency, & bandwidth

We estimate `CoverUp`'s overhead, latency, and bandwidth to demonstrate that it can perform reasonably well in a real world scenario, is feasible for deployment in large scale, and does nothing incur an intolerable overhead. `CoverUp` has four adjustable system parameters: request payload size, response payload size, the average request frequency, and the average loading time. A lower request and loading time leaves room for more artificial noise and thus increases privacy.

In our prototype implementation the request/response payload size is 75 KB, the average droplet request frequency is 30 seconds, and the average loading time is 20 seconds for a sub-system that only contains the feed application (and no browser extension), and 5 minutes for the deployment of the full system with chat and feed. Due to the browser extension's high leakage (see Figure 4), we increased the loading time to cover that leakage with noise (see Section 7).

4.2.1 Overhead

We discuss the communication load on the entry server and the involuntary participants. The entry server's traffic overhead is minimal: the size of the iframe tag in its html code. The involuntary participants' traffic overhead depends on the system parameters. To find suitable values for the system parameters, we looked at the Alexa top 15 news sites, in particular since the privacy improvements of `CoverUp`'s "forced" participation approach depends on the entry server's regular number of visitors. The average main-page load-size of the Alexa Top 15 news sites is around 2.2 MB. A few examples are CNN with 5.6 MB, NYTimes with 2.4 MB, HuffingtonPost with 6.1 MB, TheGuardian with 1.8 MB, Forbes with 5.5 MB, BBC with 1.1 MB and Reddit with 0.8 MB,.

Assume an involuntary participants that keeps each working day 10 tabs of the entry server open and each one for 10 minutes. This participants would have 7.5 MB ($= 10 \cdot 5 \cdot 2 \cdot 75$ KB) of data per day and 165 MB ($= 22 * 7.5$ MB) per month. For landline data flat-rates (i.e., for non-mobile visitors), 165 MB is not significant, e.g., in comparison to the traffic caused by streaming videos. Even a heavier usage of 80 hours per day, resulting in 1.65 GB of monthly overhead, would for many users be dominated by their video streaming traffic.

We envision a deployment of `CoverUp` not to include mobile users. In the future, however, the bandwidth budget of mobile users is increasing. For mobile phone contracts where customers have a limited budget for high-speed bandwidth and an unbounded budget for low-speed bandwidth, mobile network providers start increasing the low-speed bandwidth from 64 Kbit/s to 1 Mbit/s [6]. We believe this trend will continue in the following years and, thus, the bandwidth overhead will in the future not even be an issue for mobile users. Excluding mobile users from any overhead could additionally exclude involuntary participants from structurally weak areas.

The computational overhead for involuntary participants is insignificant, for non-mobile visitors. Section 8 further discusses the ethical aspects of using the involuntary participants' resources.

Type	**Session Bandwidth**
feed (chat disabled) downlink	20 Kbits/s
expected goodput per session	11.6 Mbit
feed + chat uplink/downlink	20 Kbits/s
expected goodput per session	6 Mbit
Type	**First Request Delay**
feed (chat disabled)	20 s + RTT
feed + chat	300 s + RTT
Type	**Latency During Session**
feed (chat disabled)	36.55 s + RTT
feed + chat	36.55 s + RTT
chat (best case)	4.5 s

Table 1: `CoverUp`'s payload bandwidth per session, delay of the first request and latency after the first request. A session starts after first request is sent. We assume that each tab is open for 10 minutes, thus a session length of 10 minutes minus the first request delay. The latency is given for the duration of the session. 6.55 seconds is added by the communication with `CoverUp`'s external application.

4.2.2 Latency & bandwidth

We evaluate the performance of `CoverUp` for the duration that a tab is opened, since the usage of `CoverUp` is bound to the visiting patterns of involuntary participants towards the entry server's sites. Depending on the service that the entry server offers, it might not be common to keep the tab open for a long time or to visit the site more than a few times a day. For the performance evaluation, we assume each tab to stays open for 10 minutes, which is in line with recent studies about e-commerce sites [1, 3].

We say that a session starts after the initial request has been sent. Table 1 illustrates the goodput (useful data transmitted), the first request delay, and the latency during a session. As the privacy leakage is lower in the case where the feed functionality is enable but the chat functionality is disabled, we use different request delays for the cases: expected 20 and 300 seconds for first request delay, for feed-only and feed+chat respectively, and expected 36.55 seconds for the latency during the session in both cases. Section 7 explains our choice for the delays. In addition to the delay of the first request `CoverUp`'s also delays the subsequent requests, however with a different rate. As data is only regularly send after this first request, the feed-only variant of `CoverUp` has a higher goodput (11.6 Mbit) per 10 minute-session than the full feed + chat variant (6 Mbit).

Recall that in all measurements the fixed request processing time at the mix server is constant time, fixed at 50 millisecond and doubled when the number of participants exceeds a threshold (see Section 4.1). Moreover, the external application has a polling rate of once every 5 seconds, and the Firefox incurs a 4 seconds-delay before writing data into the localStorage. In our implementation, the minimal latency for the chat is, thus, around

4.5 second. This time is measured from the moment the sender dispatched the encrypted chat from his external application to the time the receiver is able to see the message on his external application (the decryption time is negligible).

5 Privacy notions

This section defines the privacy notion that CoverUp achieves: no attacker can tell whether a participant in a protocol is voluntary or involuntarily (Section 5.1). Section 5.2 discusses the connection of our privacy notion to other known privacy notions from the literature.

5.1 Defining privacy

As a worst-case assumption, the attacker has complete knowledge about the running time distributions of the voluntary and the involuntary participants. The attacker is able to control the content sent by the entry server (i.e., the entry server is untrusted) and has full control over the network link (on the Internet Protocol level). In particular, the attacker is able to request arbitrary data and execute arbitrary JavaScript code in the web browsers context of the entry server and is able to drop, modify, and resend any messages sent over the network.

We overapproximate potential previous knowledge by granting the attacker the capability to control the participant's behavior. In particular, this overapproximation avoids the need to deal with various introduce user behavior profiles, since the attacker can choose the user profiles that maximize the leakage. Of course, a voluntary participant needs a more extended set of input commands than an involuntary participant. To avert any leakage to the attacker, the commands for both kinds are like for the voluntary one, while for the involuntary the surplus is simply ignored. Because this attacker might stress the OS infinitely, we restrict the input rate of these commands to a fixed rate t_{user}. Timing leakage is crucial for the notion that we consider. Since interactive Turing machine, or other computation and network models, such as the UC framework, do properly capture timing leakage, we use TUC [25] (a refinement of UC) as network and computation model (see Appendix 14.1 for a brief description). TUC enables a normal execution of all protocol parties and an adversary that can measure timing leakage. As the accuracy of a TUC adversary is not limited per se and in many deployed operating systems the accuracy of the timestamps is limited to around 1 microsecond[4], we introduce a limit t_{net} on the sample rate of the attacker.

[4]Often the timestamp also contains nanoseconds but the accuracy is nevertheless in microseconds.

Similar to other cryptographic definitions, we use a machine, called the *challenger Ch*, to capture the capabilities and the restriction to the attacker and to define the task that has to be solved by the attacker. This challenger chooses one out of two protocols at random and runs it, one π_I modelling involuntary participant or and the other π_V modelling voluntary participant. The challenger is located between the attacker and the participant, handles all their communication, enforces all restrictions (input rate t_{user} and sampling rate t_{net}). The attacker can intercept any network traffic and controls the entry server. The attacker now has to guess, which scenario the challenger runs. We let the attacker send commands that specify the participant's interaction with the system. As we quantify over all probabilistic poly-time bounded (ppt) machines, we implicitly assume that the attacker has full knowledge about π_I and π_V.

Example 1: Instantiating the model with CoverUp. In the case of CoverUp, the scenario π_I constitutes a Firefox browser together with our external application, which visits an entry server. The attacker determines what the participant does on the entry server. Only the unidirectional without any external application is used. In the other scenario π_V, the CoverUp extension is installed in the Firefox browser, together with our running external application. Here, the user utilizes the external application explicitly. The attacker tries now to distinguish to which scenario applies to a specific participant. To accomplish that, he gives commands to the participant as it would be in π_V. If it is in π_I, the additional commands just get ignored. Appendix 14.2 gives a full description of π_I and π_V. ◇

Along the lines of other indistinguishability-based definitions, we compare the probabilities of two interactions: a ppt machine $\mathscr{A}$ (the attacker) either interacts with the challenger that internally runs (i) π_I or (ii) π_V. We require that no ppt attacker $\mathscr{A}$ can distinguish case (i) from case (ii). Technically, no ppt machine $\mathscr{A}$ shall have a higher probability to output (as a final guess) 0 in case (i) has more than a distance δ away from the probability that $\mathscr{A}$ outputs (as a final guess) 0 in case (ii). In contrast to other indistinguishability-based definition and similar to differential privacy [36], we do not require δ to be negligible in the security parameter, as we also want to capture systems that do have a small amount of leakage, such as CoverUp, which is however even after thousands attacker-observations still small.

Definition 1. *A pair of protocols* (π_I, π_V) δ-hides the intention of participation *if and only if there is a* $\delta \in \{y \mid 0 \leq y \leq 1\}$ *such that for all probabilistic poly-time*

machine $\mathscr{A}$ we have

$$\Big|\Pr[0 \leftarrow \langle \mathscr{A} \mid Ch(\pi_I, t_{user}, t_{net})\rangle)] - \Pr[0 \leftarrow \langle \mathscr{A} \mid Ch(\pi_V, t_{user}, t_{net})\rangle)]\Big| \leq \delta$$

where $b \leftarrow \langle X \mid Y\rangle$ denotes the interaction between the interactive machines X and Y. This interaction stops whenever X stops, b is the output of X after it stopped.

Throughout the paper, we use the notion of an *attacker's accuracy*, i.e., his probability to guess correctly. A notion that is also used in the context of classifiers. The δ from the defintion above can be converted into accuracy by $accuracy = \delta/2 + 0.5$.[5]

Recall that the notion of differential privacy additionally includes a multiplicative factor ε. While our definitions and results could be generalized to such a multiplicative factor ε – ending up with computational differential privacy –, we omitted the ε (thus concentrating on $\varepsilon = 0$) in order to simplify the interpretation of our definition and results.

5.2 Connections to other properties

Our privacy notion implies several well-known notions. We discuss the relations below. While the following arguments and statements can be made more precise, below we present the arguments only informally, in order to illustrate that our privacy notion is suited for anonymous communication networks.

k-(sender) anonymity. The notion of k-anonymity is a common for characterizing the anonymity of an ACN. This notion guarantees that an attacker cannot guess with significantly more than a probability of $1/k$ by whom a message was sent. If k-anonymity is broken, an attacker knows that a message was sent by a particular participant. Hence, the attacker can choose a pair of participants, for which it knows that it was a voluntary participants. Hence, with the set of all involuntary and voluntary participants not controlled by the attacker as an anonymity set, our privacy notion implies (by contraposition) k-anonymity.

By covering all (uncompromised) involuntary participants into the anonymity set, `CoverUp` achieves strong anonymity properties. This is the main strength of this work.

[5]For a set of true positives TP, false negatives FN, true negatives TN, and false positives FN, Definition 1 can be rewritten as $|TP|/|TP \cup FN| - |FP|/|FP \cup TN| \leq \delta$. For $accuracy = |TP \cup TN|/|TP \cup FN \cup FP \cup TN|$, if $|TP \cup FN| = |TN \cup FP|$ then we get $2 * accuracy - 1 \leq \delta$.

Plausible deniability. Plausible deniability is very close to our privacy notion. Plausible deniability means that a voluntary participant can always plausibly deny, by way of presenting evidence for the contrary, that it did not voluntarily participate in `CoverUp`. If plausible deniability would be violated, an attacker would be able to distinguish voluntary and involuntary participants. Hence, by contraposition, our notion implies plausible deniability.

6 Privacy analysis of CoverUp

This section analyzes the privacy of `CoverUp`. Section 6.1 shows that `CoverUp` has solely timing leakage. Section 6.2 discusses potential sources of indirect privacy leakage, and Section 6.3 discusses the implications of a malicious `CoverUp` and a malicious mix server.

6.1 Reduction to timing leakage

For a model of `CoverUp` we show that solely the timing differences in the request and response times leak information. As an intermediary step, we observe that if the involuntary participant would use some delays no information leaks (Lemma 1). Then, we show that the statistical distance of these timing differences fully characterizes the privacy leakage (Theorem 1).

In our model of π_I of voluntary and π_V of involuntary participants we make some simplifying assumptions. We use the two protocols π_I and π_V from Example 1 (the full description can be found in Appendix 14.2). While these protocols exclude many secondary effects by the OS and the browser, we argue that our analysis is still valuable. π_I and π_V exclude effects of the browser caused by running an additional extension, effects of the OS caused by running `CoverUp`'s external application, and effects by the native messages between the external application and the extension. As these ignored effects solely introduce additional timing leakage, it suffices to prove that despite timing leakage there is no leakage, for which our model is sufficient. Section 7 measures this timing leakage on real systems, including these secondary effects.

As a next step, we argue that π_V is indistinguishable from a variant of π_I that includes additional delays. For this observation, we need a technical notion. The *timing transcript* of an interaction is the projection of all messages of the transcript to a constant value, say 0, which models that only the time at which a message was are observable. We say the bucketing of a timing transcript according to a sampling rate t_{net} is the *t_{net}-timing transcript*. Moreover, we call the statistical distance of two distributions (of transcripts) the *timing leakage* of these two distributions with respect to the sampling rate t_{net}. See Appendix 14.5 for the full lemma and a detailed

proof. This observation implies that an active attacker that, e.g., holds back messages, cannot learn more than a passive eavesdropper. Hence, it suffices to concentrate passive eavesdroppers.

We use Lemma 1 to show that the timing leakage between π_I and π_V already completely characterizes how well an attacker can distinguish π_I from π_V. The main proof idea for Theorem 1 is that π_I and $\pi_I + \Gamma$ are solely distinguishable by their timing leakage. Since by Lemma 1 $\pi_I + \Gamma$ and π_V are indistinguishable, π_I and π_V are only distinguishable by their timing leakage (for the full proof see Appendix 14.5).

Theorem 1. *If the timing leakage of π_I and π_V is at most δ for a sampling rate t_{net}, and if π_I and π_V use a secure channel, then for π_I and π_V we have $\delta + \mu$-hide the intention of participation, for some negligible function μ (in the sense of Definition 1).*

Proof. The two protocols $\pi_I + \Gamma$ and π_I are distinguishable by at most δ probability, since their transcripts are exactly the same except for the timing trace. Due to Lemma 1, we know that the timing leakage of $\pi_I + \Gamma$ and the timing leakage of π_V are indistinguishable. Moreover, we know by assumption that the timing leakage of π_V and π_I is at most δ. Hence, the timing leakage of π_I and $\pi_I + \Gamma$ is at most δ.

$$|\Pr[\pi_I] - \Pr[\pi_I + \Gamma]| \leq \delta \tag{1}$$

Plugging our results together, we get

$$\begin{aligned}
&\big|\Pr[\pi_I] - \Pr[\pi_V]\big| \\
&= \big|\Pr[\pi_I] - \Pr[\pi_I + \Gamma] + \Pr[\pi_I + \Gamma] - \Pr[\pi_V]\big| \\
&\leq \underbrace{\big|\Pr[\pi_I] - \Pr[\pi_I + \Gamma]\big|}_{\substack{\text{Equation (1)} \\ \leq \delta}} + \underbrace{\big|\Pr[\pi_I + \Gamma] - \Pr[\pi_V]\big|}_{\substack{\text{Lemma 1} \\ \leq \mu}} \\
&\leq \delta + \mu
\end{aligned}$$

□

6.2 Indirect privacy leakages

This section discusses indirect privacy leakages which are not direct consequences of `CoverUp`.

Browsing privacy. Involuntary users of `CoverUp` potentially reveal their browsing behavior to `CoverUp` server, as a malicious `CoverUp` server can read HTTP header's referer field. This leakage is inherent in our approach to use an entry server and to utilize involuntary participants to produce cover traffic. While this leakage exists, we would like to put it into perspective. Many popular web sites already leak this information to other services, such as advertisement networks or external analytic tools, such as Google Analytics.

Suspicious behavior leaks privacy. Another source of indirect privacy leakage would be that the usage of `CoverUp` may unconsciously influence the behavior of voluntary participants, e.g., if voluntary users spend more time on a specific entry server in order to use `CoverUp` thus significantly reduce the anonymity set. Recent studies show that the average visiting time of e-commerce website is between 8.7 and 10.3 minutes (2016 Q1) [1, 3]. Potentially, such a knowledge can be used by an attacker to distinguish voluntary participants from involuntary participants. To mitigate this `CoverUp` extension can alert users when they spent too much time on the entry server. Another way of mitigating this problem is to adjust the entry server's webpage to ask the visitors of the entry webpage when they are closing the tab "*Do you want to keep the tab open to increase the privacy of* `CoverUp`*? The tab will be automatically closed after X minutes.*" With some piece of JavaScript code it is possible to automatically close the tab after X minutes.

Browser profiling. Potentially, browser profiling methods can be used to learn whether a particular extension is installed [29]. However, measuring these effects is out of scope of this work.

6.3 Malicious mix and CoverUp server

The bi-directional channel, hence the chat, in our prototype requires to trusts the mix server and the `CoverUp` server. For efficiency reasons, the mix server directly learns the intended recipient of a chat message. This trust assumption can be removed by introducing a bi-directional broadcast: the requests of all participants (voluntary and involuntary) would be broadcast to all other participants, letting the involuntary participants send garbage. Such a bi-directional broadcast would, however, reduce the uplink rate by a factor of $1/n^2$, with n being the number of participants.

A malicious `CoverUp` server could serve involuntary participants malicious JavaScript code, which opens various attack vectors. Against attackers that do not have physical access to the `CoverUp` server, one could in principle defend by utilizing trusted components on the server side (e.g., Intel SGX or ARM Trustzone) to ensure that the correct JavaScript code is served even if the rest of the server is compromised. Alternatively, the extension could check byte for byte whether all messages from the `CoverUp` server are as expected and, if not, the extension would not do anything. We decided not to implement this variant, as it incurs high delays and increases the timing leakage.

7 Timing leakage experiments

This section experimentally evaluates the timing leakage of CoverUp. For the evaluation we used 26 systems running Windows Server 2012 operating system. Section 7.1 describes the experimental set-up. Section 7.2 inspects the effects of adding any noise after the measurement is done. Section 7.4 proves an upper bound for distinguishability and Section 7.5 discuss CoverUp privacy evaluation under circumstances such as noise vs. leakage and noise vs. observations. Section 7.6 draws conclusive remarks of the timing leakage experiment.

7.1 The experimental set-up

The goal of the experiments is to measure the leakage from the timing delays of CoverUp. To simulate realistic scenarios, we set up the involuntary and both kind of voluntary participants on 26 identical systems running Windows Server 2012 equipped with Intel Xeon E3-1245 3.4 GHz CPU and 16 GB of main memory. All the systems also run Apache tomcat server with the CoverUp and mix server deployed. The involuntary version of the test set-up only has running Mozilla Firefox browser while for the first scenario the later one executes identical browser and CoverUp's external application to receive the feed. To simulate the second scenario (the chat) the voluntary participant's browser additionally runs our extension and a python script in background which sends automated chat data to the browser extension utilizing native messaging (via STDIO). All of the communications between the server and the browser are executed on loop-back network network interface. We use tshark [10] to capture all such network traffic on loop-back interface. We compare the distributions of timing traces produced by a voluntary participant (both chat and feed) to the distribution produced by an involuntary participant. All the experiments are conducted on these 3 set-ups to investigate the timing leakage from the browser due to the browser extension and the external application.

Reflecting the attacker model. The attacker model (see Section 3) is reflected in our experiments by taking timing traces from the perspective of the attacker who has access to all network traffic. Therefore, we captured the traffic on a corresponding network interface. As a network-level attacker can change the TCP flag for timestamps and compel the victim's operating system to add timestamps to the TCP headers [9], we conduct all measurements in the settings where the browser, the entry server, the CoverUp server, and the mix server resides on a same system. The network traffic dump contains reliable timestamps with a resolution of less than 1 microseconds. Our attacker model also assumed that the attacker has no control over the operating system hence can not determine the existence of the CoverUp external application or the browser extension.

Types of measurements. We have profiled the execution time of the browser extracted from the TCP time stamp of the network packets dispatched by the browser in two separate measurements: the initial *loading* time measurement and the *periodic* execution measurement.

The loading time measurement simulates the case where the browser requests CoverUp's iframe and the browser performance in composing its context internally. We simulate this case by forcing the iframe to refresh on the entry server page in the browser. Before CoverUp's snippet is refreshed, it already has requested a feed packet from the mix server and has received an answer. In the corresponding network traffic dump, we compute the timing difference between the response of the initial iframe html source request to its first droplet request to the mix server, because waiting for the first droplet request after the iframe request enables loading the extension's content script. This timing difference captures any distinguishing feature produced by the extension.

The periodic measurement models the scenario where the voluntary and involuntary participant load the iframe once, followed by one feed request to the server and one response from the mix server. In the network traffic dump, we look for the timing difference for two contiguous feed requests from the browser.

Evaluation details. All the experiments are performed on 26 identical system running Windows Server 2012 standard edition. The systems are equipped with Intel Xeon E3-1245 3.4 GHz CPU and 16 GigaByte of main memory. We deployed CoverUp and mix server on Apache Tomcat web server. We also prepare a custom SSL certificate and configure Tomcat to use SSL all the time (strict transport layer security). All of the communications between the server and the browser are executed on loop-back network network interface. We use tshark [10] to capture all such network traffic on loop-back interface. To obtain the time differences between subsequent requests, we use the time stamp of the first TCP packet (carrying TLS application data) for each request and answer and simply subtract them. To avoid any caching artifacts, we disable the support for HTTP request caching on the server side completely.

7.2 The timing without additional noise

CoverUp does not send the droplet requests at fixed time but rather draws delay noise from a gaussian distribution and accordingly noises the time at which the requests are

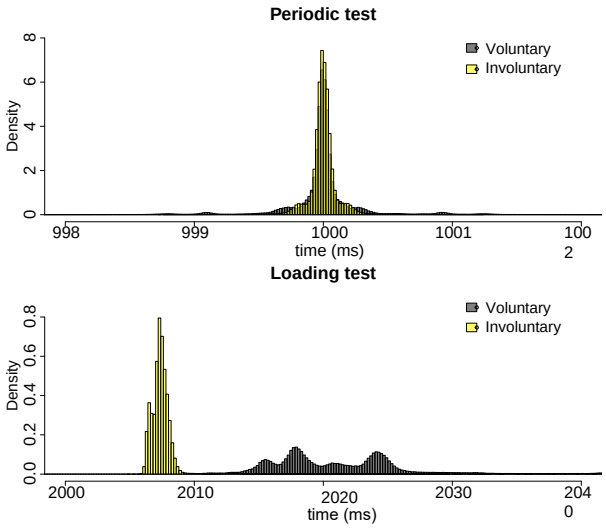


Figure 4: Distribution of timing traces (without additional noise) of 26 windows server systems running periodic and loading testing. Periodic test consists of around 12.9 million timing traces while the loading test around 600K (voluntary chat and involuntary both).

sent. To simplify and accelerate testing, however, the experiments do not draw this noise. We added the noise artificially afterwards. We measured the independence of generating noise inside the experiment and adding it after the experiment. Section 7.3 elaborates on this independence.

For the sake of illustration, we briefly describe the histograms of the timing traces without the additional noise. Figure 4 depicts the differences in the measurements of our implementations on all the systems. The loading process invokes much more computational power and is therefore more noisy, as opposed to sending and receiving a simple network packet as in the periodic case, the distribution in the loading one have a wider range. Remarkable here is that in the periodic case the distributions for interactive and non-interactive requests look very similar. The mean and median differ less than a microsecond.

7.3 Independence of additional noise

In the analysis of the timing leakage, we simulated the additional noise by adding it to the measurement result. To justify this procedure, we conducted separate experiments, similar to the periodic scenario, but instead of waiting 1000ms for the next droplet request, we drew in JavaScript a uniformly distributed random number (using Math.random()) and expanded it in an affine way such that an interval ranges from 200ms to 1800ms. Additionally, we stored each of the drawn random numbers together with an epoch time stamp. Later in the analysis

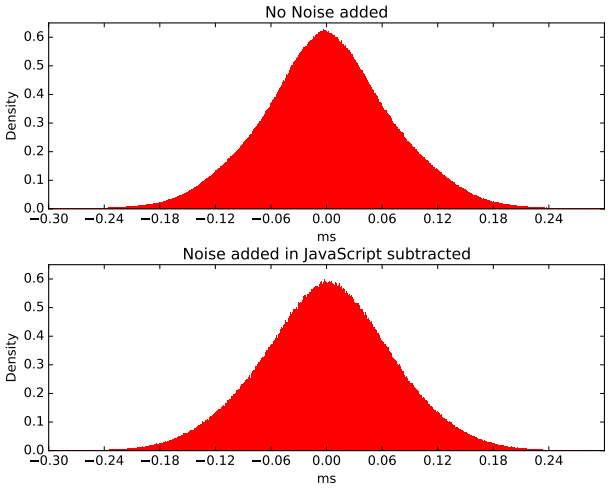


Figure 5: Statistical Independence using uniform noise: Distance: 1.8%

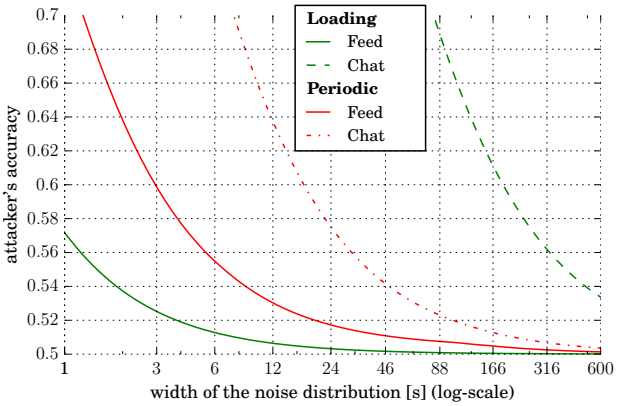


Figure 6: Attacker's accuracy vs the width of the noise distribution, for 20'000 periodic and 1'000 loading observations if only feed is enabled and 10'000 periodic and 1'000 loading observations if chat is enabled. The noise distributions are half-gaussian distributions with mean 0 and a very high standard deviation (10 times the width of the noise).

step, we subtracted the corresponding random number from the network dump measurement. This procedure produced measurements artifacts, caused by the time resolution of our system (which lies slightly under 1us). As we are only interested in the fact whether artificially adding the noise after the experiment is independent of directly adding the additional noise in the experiments, we clustered close histogram bars that are not separated by a significant gap.

Figure 5 shows the resulting distribution. The statistical distance of these two distributions is 1.8% which is an acceptable value.

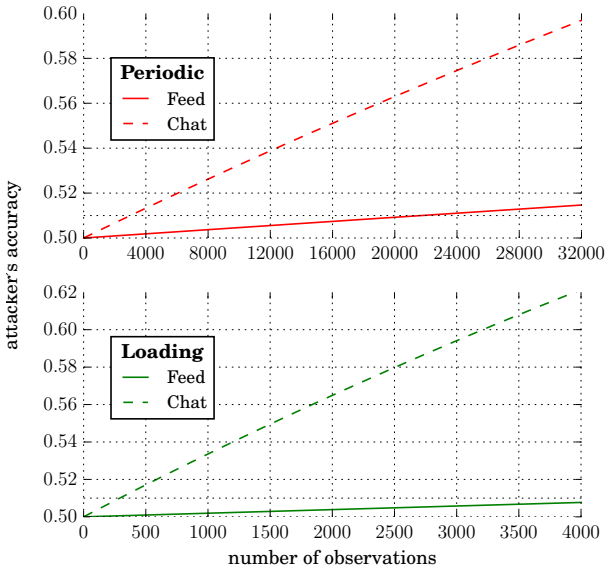


Figure 7: The accuracy of an attacker for periodic and loading leakage, with the following noise widths (with expected request time): feed periodic 60s (30s), chat periodic 60s (30s), feed loading 40s (20s), chat loading 10 min (5 min).

7.4 Attacker accuracy upper bound

Estimating the attacker's pre-knowledge about the delays of a user is challenging. In principle, we cannot exclude that a very strong (e.g., state-funded) attacker runs measurement experiments on many combinations of hardware, browsers, and operating systems. These measurements might be usable by a malicious website to fingerprint the operating system and the hardware, as indicated shown in a recent work [29]. Therefore, our analysis includes a minimal privacy bound against an overly strong attacker that extensively measured the timing leakage for the system of the eavesdropped participant (in the sense of Definition 1). We then compute the accuracy of the optimal attacker with this pre-knowledge as the statistical distance (also called the total variance). Lemma 3 in Appendix 15 recalls the textbook-proof that the statistical distance is an upper bound on the advantage ($= 2 * (\text{accuracy} - 0.5)$) of any (i.e., also potentially unbounded) attacker.

We over-approximate the attacker's advantage δ_i after i subsequent runs of `CoverUp` as follows: $\delta_0 =$ statistical distance after adding noise,$\delta_i = \delta_{i-1} + (1 - \delta_{i-1}) \cdot \delta_0$,. Appendix 16 proves this over-approximates the attacker's advantage. Recall that all our graphs show the attacker's accuracy, which can be computed as advantage$/2 + 0.5$. Moreover, for computing this accuracy, we use 100 microseconds instead of 1 microsecond, since it turns out that after convolution with the noise distribution there is virtually no advantage to use 1 instead of 100 microseconds.

7.5 Evaluating privacy

We evaluate the privacy that different noise distributions of request delays yield against various amounts of attacker observations. The timing leakage is upper bounded by the sum of the leakage caused by loading of the JavaScript (up to requesting the first droplet) and by subsequent periodic requests. Because our two experiments take place at two different stages of `CoverUp`'s procedure, we can capture these leakages separately and quantify them separately, as well. We call the accuracy of an attacker that can distinguish the two loading experiments the loading accuracy and the accuracy of an attacker that can distinguish the two periodic experiments as periodic accuracy. The bound for the attacker's accuracy $=$ loading accuracy $+$ periodic accuracy $- 0.5$.

Leakage vs. noise. Figure 6 plots the attacker's accuracies vs width of the noise distribution, with a log-scale x-axis. As noise distribution, we use a very wide half-Gaussian distribution with mean 0 which we truncate after n seconds. This width n of the noise distribution (x-axis on Figure 6) also determines the standard deviation that we use: we use $10n$ as standard deviation. In this way, we effectively obtain almost a uniform distribution.

Figure 6 shows that the width of the noise is inverse proportional to the accuracy of the attacker. Moreover, it shows for feed-only deployment of the system the attacker's accuracies for up to 20'000 observations and, for the system with chat the attacker's accuracies to 10'000 observations. For the sake of illustration, we will select a few numbers that yield strong privacy. The figure indicates that if only the feed application is deployed, for a width of the noise of 40s the loading accuracy is $\leq 50.2\%$ and with a width of 60s the periodic accuracy is $\leq 50.9\%$. If the chat is additionally enabled and with a noise width of 600 seconds, the loading accuracy is $\leq 53.2\%$ and, with a width of 60 seconds the periodic accuracy is $\leq 53.1\%$.

The figure shows that even less noise would achieve reasonable privacy guarantees. For the chat case, 200s and 1'000 loading observations would yield a loading accuracy of less than 60%, and even the periodic noise could be reduced 30s while having for 10'000 periodic observations a periodic accuracy of $\leq 55\%$.

For feed + chat, a loading noise of 10 minutes (5 minutes expected time) is still practical for a high-latency chat, as the average visiting time of an e-commerce user around 10.3 minutes [1, 3]. With 10 minutes of loading noise (i.e., 5 minutes expected initial request time),

our high-latency chat would still be able to send 10 requests in the expected case. The size of the payload can be increased without violating the privacy of the voluntary participants; a single request and response can be used to send and receive high amount of new messages.

Leakage vs. number of observations. Figure 7 shows the development of the upper bound on the attacker' accuracy if the number of observations increases. For these graphs, we assumed the following request delay noise widths (with expected request time): feed periodic 60 s (30 s), chat periodic 60 s (30 s), feed loading 40 s (20 s), chat loading 10 min (5 min). The leakage increases linearly with an increasing number of observations. The figure shows that with these widths even a 3 times more active voluntary chat participant with 30'000 observations could still count on $\leq 60\,\%$ periodic accuracy and $\leq 60\,\%$ loading accuracy.

7.6 Privacy conclusion

Recall that the usage pattern of a voluntary participant is restricted to the visiting behavior of involuntary participants, as discussed in Section 6.2. While this usage pattern depends on the specific service that the entry server provides, we assume the usage pattern for a popular e-commerce site. As shown in [1, 3] each visit should not be longer than 10.3 minutes. We stress that this usage pattern can be controlled and recommended or even enforced by the extension or the external application. Additionally, we envision a button to set the extension active only when required by the user. This justifies our assumptions of a voluntary participants that visits and utilizes the entry server 4 times per working day, stays each time 10 minutes, and has 4 weeks of holidays per year. In a deployment of feed-only application with noise widths as above (loading = 40 s, periodic = 60 s), a year worth of observations correspond to around 20'000 periodic observations and 1'000 loading observations (see Section 6.2). These parameters results in an attacker's accuracy of $\leq 51.1\%$, which is only 1.1% better than pure guessing. In a deployment of the full system (including the chat) with widths as above (loading = 600s, periodic = 60s), a year worth of observations correspond to around 10'000 periodic and 1'000 loading observations. That results in an attacker's accuracy of $\leq 56.3\%$, which is only 6.3% better than pure guessing.

8 Ethical aspects

While our design takes care not to harm a involuntary participant's system, it causes an involuntary participant's browser to execute our JavaScript code, stores content and forces participation in our protocol. In order words, we propose to use the computation and traffic resources of a visitor to increase the anonymity of an ACN. To inform the visitor about this involuntary allocation of resources, the entry web page can include information for the visitors or even require explicit consent from the visitors.

We argue that the amount of resources that we allocate is not out of scale. Our work received a formal approval of our ETH's ethics commission. First, the involuntary allocation of resources is nothing unexpected for the visitor; it is already done by advertisements or QoS scripts such as Google Analytics. In particular, webpages that incorporate `CoverUp` would not cause unexpected behavior on a visitor's browser. Second, we propose to only incorporate our design into non-mobile version of a webpage, thereby excluding mobile phone visitors and visitors from structurally weak areas from any overhead. Third, we discuss the overhead for a visitor in Section 4.2 in detail. As a summary, the computational overhead of `CoverUp` is negligible but the traffic overhead for a visitor would be around 9MB per day. While this is a non-negligible amount of traffic, a single YouTube Video or even visiting 4 Alexa Top 15 News Sites can cause more traffic. Moreover, we think that in the future traffic will become much cheaper while our system remains highly useful even if solely text is transmitted, i.e., with 9MB per day.

9 Discussion of selected legal questions

One of the challenges in answering the question whether the provision of `CoverUp` and the upload of the JavaScript code by the entry server is legal or not (and many other questions evolving around the use of the Internet) is that, whereas the Internet functions globally, law mostly [43] remains limited by territory because sovereign states put their own legislation into effect [17, 18, 11]. The legal provisions and possible offenses that apply to the technical setup of `CoverUp`, differ from country to country. Moreover, as law is not an exact science and definite legal statements are made by the courts, we conclude the legal discussion herein with an assessment that we consider probable.

In this section we limit the legal analysis to a selected discussion on whether the activity of the provider of the entry server could qualify as cybercrime offense. We do not, for instance, analyse offenses by the provider of the `CoverUp` server or of the mix server, or cover aiding and abetting.

Many countries enforce their own laws and have their own (territorial) jurisdiction, many countries, among others the EU member states and the USA, have ratified [2] in the Convention on Cybercrime [43] (CCC) – the inter-

national treaty on crimes committed via the Internet and other computer networks. This international treaty criminalizes, among others, illegal access (article 2 CCC), data interference (article 4 CCC), and misuse of devices (article 6 CCC). Do these offenses apply to the provider of the entry server?

9.1 Illegal access

Illegal access (article 2 CCC) penalizes the entering of a computer system but does not include the mere sending of an e-mail message or a file to a system. The application of standard tools provided for in the commonly applied communication protocols and programs, is not per se "without right", in particular not if the accessing application can be considered to have been accepted (e.g. acceptance of cookies [19, 14, 15, 16] by client).

However, a broad interpretation of article 2 CCC is not undisputed (refer [43], para. 44 - 50).

Upon request, the entry server delivers a webpage that contains an iframe request for the `CoverUp` server, which then delivers the JavaScript to the browser for the download of the droplets. Not only does the entry server merely send a file (pointer) to the browser, but the request to download the JavaScript from the `CoverUp` server is standard browser functionality for communication. The same would happen if the entry server were financed by online advertising: upon request the entry server would deliver a webpage pointing to the advertising server and trigger the download of the advertising text or pictures to the browser. As this is a standard online process, we conclude that even in a broad interpretation of article 2 CCC, the provider of the entry server should not be illegally accessing the browser.

9.2 Data interference

Data interference (article 4 CCC) penalizes the damaging, deletion, deterioration, alteration or suppression of computer data "without right". This provision protects a computer device from the input of malicious code, such as viruses and Trojan horses as well as the resulting alteration of data. However, the modification of traffic data for the purpose of facilitating anonymous communications (e.g., the activities of anonymous remailer systems) should in principle be considered legitimate protection of privacy (refer [4, 5, 8, 7], [12, Recitals(1) and (35)]), [13, Article 13], and, therefore, be considered as being undertaken "with right" [43, para. 61].

`CoverUp` does not damage, delete, deteriorate, or suppress data on the participant's client. However, it does alter the data on the hard disk: on the one hand the webpage with the iframe uses disk space and thus modifies the participant's data; on the other hand `CoverUp` triggers the download of the JavaScript code and subsequently the droplets from the mix server to the involuntary participant's browser, which again uses disk space and thus modifies the data anew.

However the explanatory report to the Convention on Cybercrime foresees that the file causing data interference be "malicious". Code is malicious if it executes harmful functions or if the functions are undesirable.

As we concluded in the previous subsection, the webpage containing the iframe request for the `CoverUp` server submitted by the entry server is standard core browser functionality. Thus from a technical viewpoint, `CoverUp` is not harmful. Therefore in our view the provider of the entry server not does cause any malicious data interference. We advocate that article 4 should not apply to the provision of the webpage with the iframe by the provider of the entry server.

9.3 Misuse of devices

Misuse of devices (article 6 CCC) penalizes the production, making available, or distribution of a code designed or adapted primarily for the purpose of committing a cybercrime offense, or the possession of such a computer program. It refers to the commission of "hacker tools", i.e. programs that are e.g. designed to alter or even destroy data or interfere with the operation of systems, such as virus programs, or programs designed or adapted to gain access to computer systems. The objective element of offense comprises several activities, e.g. distribution of such code (i.e. the active act of forwarding data to others), or making code available (i.e. placing online devices or hyperlinks to such devices for the use by others) [2, para. 72].

One of the main questions relating to the misuse of devices is how to handle dual use devices (code). Dual use means in our case that the JavaScript code could be used to download legal content, e.g. political information, as well as illegal content, e.g. child pornography. Should article 6 CCC only criminalize the distribution or making available of code that is exclusively written to commit offenses or should it include all code, even if produced and distributed legally? Article 6 CCC restricts the scope to cases where the code is objectively designed primarily for the purpose of committing an offense, thereby usually excluding dual-use devices [2, para. 72 - 73].

First, it is important to note that `CoverUp` was not designed primarily for the purpose of committing an offense. While the main purpose of `CoverUp` is to protect privacy, it can be used to conceal illegal activities.

Second, can the download of criminal information be considered an illegal activity if the information is encrypted? Here we draw a legal analogy to data protection law. Data relating to an identified or identifiable person

is considered personal data [12, article 2(a)], [23, article 4(1)]. If a person is identifiable or identified, data protection law applies. However, if the personal data are pseudomised or anonymised, then data protection law might not apply anymore because the (formerly identiable or identified) person cannot longer be identified.

Recital (83), article 6(4)(e), 32(1)(a) and 34(3)(a) of the new General Data Protection Regulation[6] stipulate that encryption renders the personal data unintelligible and mitigates the risk of infringing the new regulation.

By applying this data protection principle to the encryption of data by `CoverUp` we can argue that the data provided by the mix server in the droplets are not information because the data is unintelligible. Not only does the involuntary participant not have sufficient data to reassemble the droplets to a whole, but the data are encrypted in such manner that it is impossible to make any sense of it. At least from a theoretical viewpoint the encryption of `CoverUp` cannot be breached. We therefore conclude that the JavaScript code (a) with regard to the involuntary participant does not qualify as dual use device because even if it is used for illegal purpose, the data transmitted remain unintelligible and therefore do not qualify as information; and (b) with regard to the voluntary participant can be qualified as dual use device because the encrypted and unintelligible data are decrypted and reassembled to intelligible information.

9.4 Legal conclusion

We discussed the applicability of articles 2 (illegal access), 4 (data interference), and 6 (misuse of device) CCC to `CoverUp`. We conclude that the provider of the entry server is probably not illegally accessing the participant's browser by applying `CoverUp`; that the provider of the entry server probably does not cause any malicious data interference; and that the use of `CoverUp` with regard to the involuntary participant does not qualify as misuse of device. As regards the reassembly of the droplets to a meaningful whole, if the information is illegal, `CoverUp` might qualify as dual use device and fall under article 6 CCC. We conclude that at least with regard to the risk of indictment pursuant to article 6 CCC it seems advisable that the provider of the entry server does not provide the JavaScript code for download.

10 Deployment

We have witnessed a steady rise of concern regarding privacy. Such includes state backed surveillance, web based services collecting huge amount of private information and discrimination of citizens who access sensitive materials such as leaked documents. In recent years, a number of countries reformed their privacy protection laws, which specifically aims to provide protections against the misuse of citizens' private data. One major example is European Union's EU-GDPR and the surveys accompanying it [21, 22] shows that there is a need for privacy-preserving systems. Anonymous communication networks (ACN) is the basic building blocks for many privacy preserving protocols. `CoverUp` provides a strong privacy guarantee for hiding the intention. Our proposed forced participation technique achieves this by hiding the voluntary users in the traffic generated by the involuntary users. Existing systems can easily incorporate `CoverUp` by setting up the entry server in their own service. The code integrating is effortless and requires almost no modification. The host servers only have to include an iframe pointing to the `CoverUp` server.

The external application and the browser extension have to be delivered out-of-band channel. Installing these two components is straight-forward as it only includes to add the extension program to Chrome and run the external application's compiled binary.

11 Related Work

Hiding ones intentions in the internet is done since the beginning. It is closely connected the field of censorship circumvention.

Anonymous communication protocols. There are numerous approaches to hide a user's traffic. Anonymous communication (AC) protocols hide traffic by rerouting and potentially waiting. Low-latency AC protocols, such as Tor [35] or Hornet [31], are vulnerable to traffic correlation attacks. High-latency mix-nets, such as Mixminion [34], which do not require the user client to continuously send messages leak a user's intend to connect to the anonymity network, which might seem suspicious and prevent a user from using the mix-net client. AC protocols that do require the user client to continuously send messages, such as DISSENT [32] or Vuzuvela [57], still require the active participation of the users in the protocol, which can leak the intention. Our solution can be easier deployed and does not require a sophisticated infrastructure.

Ricochet is a related project: an anonymous chat. Based on Tor's hidden service design, Ricochet implements a privacy-preserving instant messenger. As Ricochet is based Tor, it suffers from Tor's weaknesses, such as traffic correlation attacks and website fingerprinting. As our system is a constant-rate communication system, `CoverUp` does not suffer from these kinds of attacks. Tor

[6] Regulation (EU), applicable as of 25.5.2018

and thus Ricochet leak that a user intends to use Tor. Vy CoverUp's indistinguishability of voluntary and involuntary participants enables users to deny the intention to participate in the system.

Covert channels & steganography. Covert channels hide whether communication took place, and thus achieve full deniability. As covert channels typically use a piggyback approach to transport data, they depend on existing data streams, resulting in a dependency of the piggybacked system for latency and throughput. Steganography is another approach which is hiding messages in unsuspicious looking data [47, 37, 24]. But once detected, the origin and therefore the intention is obvious. The same applies to Mixing [48]. Plausible deniability is the possibility to deny the knowledge of actions done by others (e.g., Cirripede [44]). Off-the-record messaging: published MAC key after talk: does not protect against real time monitoring [26].

McPherson et al. proposed CovertCast, a broadcast hidden in normal video streams like YouTube [50]. Che et al. were able to create a deniable communication channel based on different levels of noisy channels [30]. Deploying that system is, however, require a much higher effort by the service provider (e.g., YouTube) and does not provide any interactive communication like CoverUp. Freewave [45] provides a covet channel where the user can modulate his internet traffic signal into acoustic data and transfer it to remote server via VoIP such as skype. Such system have bandwidth limitation and is vulnerable to attacks described in [39]. SWEET [60] describes a covert channel e-mail communication where the user can send the query to the remote server by using any available mail server. Such system suffered from very low bandwidth and high latency, making them practically infeasible for deployment. CloudTransport [28] introduced covert communication which involves publicly accessible cloud servers such as Amazon S3 which acts as the oblivious mix. But services like this does not provide protection against attackers learning intention. Infranet [38] describes a system executing covert communication using image stenography but also suffers from a low bandwidth.

12 Conclusion & future work

We discussed how "forced" participation can improve the privacy of anonymous communication network (ACNs). By adding involuntary participants to the anonymity set, we achieve not only an increased anonymity set but also a plausible deniability: an attacker cannot tell whether an observed communication stream originates from a voluntary or an involuntary participant. We developed a system CoverUp with two features: feed, and high-latency chat. This approach of "forced" participation can help to bootstrapp mid- and high-latency ACNs.

Peer-to-peer variant. Even though the CoverUp performs reasonably well in terms of bandwidth (for serving normal HTML pages, refer Section 4.2.1), the latency is bounded by the droplet request rate from a particular entry server. The latency can be decreases significantly by adding a peer to peer communications between all the participants. Such peer to peer network allows the feed data transferred between participants hence decrease the load of the CoverUp and in particular the mix server. Effectively CoverUp will behave like a bit torrent network enhancing the bandwidth multiple fold.

Mix-server as a full HTTP proxy. CoverUp can be easily extended to make the mix server acts like a full HTTP proxy server. Such modification will allow a voluntary participants to browse through arbitrary web pages. This can be an immediate extension of CoverUp.

Custom browser. Our evaluation of the timing leakage of CoverUp (Section 7) shows that the browser extension, used for the bi-directional channel, has some timing leakages. After many careful observations and experiments, we conjecture that the timing leakages arises from the browser's internal scheduler. The extension operates through several layers of heavy abstractions which makes it non trivial to develop extensions which are such timing sensitive in nature. We think that a set modifications in the browser code will solve such problem where all the modifications can be implemented on native code rather than high level abstraction (such as the JavaScript based API). Such modification is nontrivial and requires a high amount of engineering effort, making it out of scope for our current research. But such modification can be an immediate followup of this work and would be a major contribution towards privacy preserving browsing applications.

References

[1] 2016 q1 demandware shopping index.

[2] Chart of signatures and ratifications of treaty 185.

[3] E-commerce kpi study: There's (finally) a benchmark for that.

[4] European convention on human rights (ehcr).

[5] Fourth amendment.

[6] Heise article (in German): O2 entschärft Drosselung: Neue Tarife immer mit mindestens 1 Mbit/s.

[7] Katz v. united states, 389 u.s. 347 (1967).

[8] Olmstead v. united states, 277 u.s. 438 (1928).

[9] Rfc 7323 - tcp extensions for high performance.

[10] tshark-the wireshark network analyzer 2.0.0.

[11] America's founding documents — national archives, 1776.

[12] Directive 95/46/ec of the european parliament and of the council, Nov 1995.

[13] Federal constitution of the swiss confederation, Apr 1999.

[14] Directive 2002/22/ec of the european parliament and of the council, Apr 2002.

[15] Directive 2002/58/ec of the european parliament and of the council, July 2002.

[16] Regulation (ec) no 2006/2004 of the european parliament and of the council, Dec 2004.

[17] Consolidated version of the treaty on the functioning of the european union, May 2008.

[18] Eur lex, May 2008.

[19] Directive 2009/136/ec of the european parliament and of the council, Dec 2009.

[20] Rfc 918 - post office protocol, Dec 2009.

[21] Attitudes on data protection and electronic identity in the european union, June 2011.

[22] State of privacy report 2015, 2015.

[23] Regulation (ec) no 2006/679 of the european parliament and of the council, May 2016.

[24] ARTZ, D. Digital steganography: hiding data within data. *IEEE Internet computing 5*, 3 (2001), 75–80.

[25] BACKES, M., MANOHARAN, P., AND MOHAMMADI, E. TUC: Time-sensitive and Modular Analysis of Anonymous Communication. In *Proceedings of the 27th IEEE Computer Security Foundations Symposium (CSF)* (2014), IEEE, pp. 383–397.

[26] BONNEAU, J., AND MORRISON, A. Finite-state security analysis of otr version 2.

[27] BOYKO, V. *On the Security Properties of OAEP as an All-or-Nothing Transform*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999, pp. 503–518.

[28] BRUBAKER, C., HOUMANSADR, A., AND SHMATIKOV, V. Cloudtransport: Using cloud storage for censorship-resistant networking. In *International Symposium on Privacy Enhancing Technologies Symposium* (2014), Springer, pp. 1–20.

[29] CAO, Y., LI, S., AND WIJMANS, E. (Cross-)Browser Fingerprinting via OS and Hardware Level Features. In *Proc. 25th Network and Distributed System Security Symposium (NDSS)* (2017), Internet Society.

[30] CHE, P. H., BAKSHI, M., AND JAGGI, S. Reliable deniable communication: Hiding messages in noise. In *Information Theory Proceedings (ISIT), 2013 IEEE International Symposium on* (July 2013), pp. 2945–2949.

[31] CHEN, C., ASONI, D. E., BARRERA, D., DANEZIS, G., AND PERRIG, A. Hornet: high-speed onion routing at the network layer. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (2015), ACM, pp. 1441–1454.

[32] CORRIGAN-GIBBS, H., AND FORD, B. Dissent: accountable anonymous group messaging. In *Proceedings of the 17th ACM conference on Computer and communications security* (2010), ACM, pp. 340–350.

[33] DAEMEN, J., AND RIJMEN, V. *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.

[34] DANEZIS, G., DINGLEDINE, R., AND MATHEWSON, N. Mixminion: Design of a type iii anonymous remailer protocol. In *Proceedings of the Symposium on Security and Privacy (S&P)* (2003), IEEE, pp. 2–15.

[35] DINGLEDINE, R., MATHEWSON, N., AND SYVERSON, P. Tor: The second-generation onion router. Tech. rep., DTIC Document, 2004.

[36] DWORK, C., MCSHERRY, F., NISSIM, K., AND SMITH, A. Calibrating Noise to Sensitivity in Private Data Analysis. In *Proc. 10th Theory of Cryptography Conference (TCC)* (2006), pp. 265–284.

[37] EGGERS, J. J., BAEUML, R., AND GIROD, B. Communications approach to image steganography. In *Electronic Imaging 2002* (2002), International Society for Optics and Photonics, pp. 26–37.

[38] FEAMSTER, N., BALAZINSKA, M., HARFST, G., BALAKRISHNAN, H., AND KARGER, D. R. Infranet: Circumventing web censorship and surveillance. In *USENIX Security Symposium* (2002), pp. 247–262.

[39] GEDDES, J., SCHUCHARD, M., AND HOPPER, N. Cover your acks: Pitfalls of covert channel censorship circumvention. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (2013), ACM, pp. 361–372.

[40] GILAD, Y., AND HERZBERG, A. Spying in the dark: Tcp andtortraffic analysis. In *International Symposium on Privacy Enhancing Technologies Symposium* (2012), Springer, pp. 100–119.

[41] GOOGLE. Content security policy (csp) - google chrome.

[42] GOOGLE. Cross origin xmlhttprequest - google chrome.

[43] GROUP, I. N. W. Convention on cybercrime, budapest, 23.xi.2001, Oct 1984.

[44] HOUMANSADR, A., NGUYEN, G. T., CAESAR, M., AND BORISOV, N. Cirripede: Circumvention infrastructure using router redirection with plausible deniability. In *Proceedings of the 18th ACM Conference on Computer and Communications Security* (New York, NY, USA, 2011), CCS '11, ACM, pp. 187–200.

[45] HOUMANSADR, A., RIEDL, T. J., BORISOV, N., AND SINGER, A. C. I want my voice to be heard: Ip over voice-over-ip for unobservable censorship circumvention. In *NDSS* (2013).

[46] JANSEN, R., TSCHORSCH, F., JOHNSON, A., AND SCHEUERMANN, B. The sniper attack: Anonymously deanonymizing and disabling the tor network. Tech. rep., DTIC Document, 2014.

[47] KAMBLE, M. P. R., WAGHAMODE, M. P. S., GAIKWAD, M. V. S., AND HOGADE, M. G. B. Steganography techniques: A review. *International Journal of Engineering 2*, 10 (2013).

[48] LE BLOND, S., CHOFFNES, D., ZHOU, W., DRUSCHEL, P., BALLANI, H., AND FRANCIS, P. Towards efficient traffic-analysis resistant anonymity networks. In *ACM SIGCOMM Computer Communication Review* (2013), vol. 43, ACM, pp. 303–314.

[49] MACKAY, D. J. Fountain codes. In *Communications, IEE Proceedings-* (2005), vol. 152, IET, pp. 1062–1068.

[50] MCPHERSON, R., HOUMANSADR, A., AND SHMATIKOV, V. Covertcast: Using live streaming to evade internet censorship. *Proceedings on Privacy Enhancing Technologies 2016*, 3 (2016), 212–225.

[51] MOLLAND, H., AND HELLESETH, T. An improved correlation attack against irregular clocked and filtered keystream generators. In *Annual International Cryptology Conference* (2004), Springer, pp. 373–389.

[52] PROJECT, T. T. Tor metrics, Nov 2016.

[53] RIVEST, R. L. *All-or-nothing encryption and the package transform*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997, pp. 210–218.

[54] SHOKROLLAHI, A. Raptor codes. *IEEE transactions on information theory 52*, 6 (2006), 2551–2567.

[55] SUN, Y., EDMUNDSON, A., VANBEVER, L., LI, O., REXFORD, J., CHIANG, M., AND MITTAL, P. Raptor: routing attacks on privacy in tor. In *24th USENIX Security Symposium (USENIX Security 15)* (2015), pp. 271–286.

[56] SUNDARARAJAN, J. K., SHAH, D., AND MÉDARD, M. Arq for network coding. In *Information Theory, 2008. ISIT 2008. IEEE International Symposium on* (2008), IEEE, pp. 1651–1655.

[57] VAN DEN HOOFF, J., LAZAR, D., ZAHARIA, M., AND ZELDOVICH, N. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles* (2015), ACM, pp. 137–152.

[58] WANG, T., AND GOLDBERG, I. Improved website fingerprinting on tor. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society* (2013), ACM, pp. 201–212.

[59] WANG, X., CHEN, S., AND JAJODIA, S. Network flow watermarking attack on low-latency anonymous communication systems. In *2007 IEEE Symposium on Security and Privacy (SP'07)* (2007), IEEE, pp. 116–130.

[60] ZHOU, W., HOUMANSADR, A., CAESAR, M., AND BORISOV, N. Sweet: Serving the web by exploiting email tunnels. *arXiv preprint arXiv:1211.3191* (2012).

13 Preliminaries

In this section we describe existing tools and techniques that have been used in our proposed system `CoverUp`.

13.1 Fountain Code

Fountain codes [49, 56] are a class of forward error correction (FEC) codes with the following properties

- Arbitrary sequence of encoding symbols can be generated form a given set of source symbols i.e., input data.
- Original source symbols can be recovered from any subset of encoding symbols with size more than a threshold value T.
- Encoding symbols can be delivered regardless of specific order.
- Fountain codes does not show fixed code rate.

In this paper, we have used a bit-wise `XOR` ($\oplus$) based fountain code with error detection mechanism.

In a simple analogy, one can consider an empty glass for water. A fountain emits the input data encoded in a large amount of droplets in a steady stream. Anyone can collect them in a glass alternately and if one thinks the glass is filled enough, one may try to assemble the data from the water (data stored in the glass). If the amount of droplets is insufficient to reassemble the data, one has to wait longer to collect more droplets and retries later.

Our specific fountain code implementation is not optimal. There exists efficient fountain codes such as *Raptor* [54] in the literature but most of them are protected by intellectual property rights.

13.2 All-or-nothing transformation

All-or-nothing transformation is an encryption mode in which the data only can be decrypted if all the encrypted data is known. More precisely: "An AONT is an unkeyed, invertible, randomized transformation, with the property that it is hard to invert unless all of the output is known."[27].

We modified the *all-or-nothing scheme* proposed by Rivest [53] which encrypts all data with a symmetric key cryptography algorithm (in our implementation, we use `AES-128` [33]) in Cipher Block Chaining (CBC) mode and appends a new block in which the encryption key is XOR'ed ($\oplus$) with the 128 bit truncated `SHA-256` hashes of all the encrypted blocks. This guarantees that one needs all encrypted data (or at least its hash) to extract the decryption key from last block.

1. Input message block: $m_1, m_2, \ldots, m_n$
2. Chose random key $\mathscr{K} \xleftarrow{R} \{0,1\}^{128}$ for `AES-128`.
3. Compute output text sequence $m'_1, m'_2, \ldots, m'_n, m'_{\text{key}}$ as follows:
   - Let $m'_i = Enc(\mathscr{K}, m_i)\ \forall\, i \in 1, \ldots, n$ with CBC mode.
   - Let $m'_{\text{key}} = \mathscr{K} \oplus h_1 \oplus h_2 \oplus \ldots \oplus h_n$ where $h_i = \mathscr{H}_i[1, \ldots, 128]; \mathscr{H}_i = \texttt{SHA-256}(m_i) \quad \forall i \in 1, \ldots, n$
   - Send $m' = m'_1 || \ldots || m'_n || m'_{\text{key}}$

The receiver can recover the key $\mathscr{K}$ only after receiving all message blocks. He executes the following steps

- $\mathscr{K} = m'_{\text{key}} \oplus h_1 \oplus h_2 \oplus \ldots \oplus h_n$.
- $m_i = Dec(\mathscr{K}, m'_i)\ \forall\, i \in 1, \ldots, n$.

14 Detailed privacy analysis

14.1 TUC: A time-sensitive model for networks of machines

For quantifying the privacy guarantees of `CoverUp`, we use the TUC framework as a time-sensitive model for network of machines. TUC constitutes a model for networks of machines that is time-sensitive. In TUC, time is represented as a rational number, and there is a global time, on which the time of each machine depends. Each machine has a local clock that is a function t in the global time. This function represents potential delays or inaccuracies of the local timer. Moreover, TUC assigns to each machine a speed s. Hence, a machine is after c step at the global time c/s and the local timer of that machine shows $t(c/s)$.

The execution of a network of machines in TUC is conducted by a single machine, called the execution, that runs all participating machines as submachines. This execution sequentially activates each machine, counts the steps that each machine performs, and coordinates the timely sending and receiving of messages between the submachines. Due to the sequential activation of machines, it can happen that one machine is already far in the future compared to all other machines. It is shown [25] that all reasonable activation strategies lead to the same results. As a consequence, we ignore that TUC internally uses sequential activation and treat all machines as if they are executed in parallel and run according to their speed.

A party can consist of several parallel machines (e.g., several CPUs) that communicate to each other.

14.1.1 Timeless environment and attacker

As in the UC framework, TUC includes an environment and an adversary. This environment and this adversary can consist of several machines that work in parallel. A

natural way of modeling this capability is to represent the environment and the adversary as a set of parallel machine. While such a model is more accurate, we decided for the sake simplicity to over-approximate this strength of the environment and the adversary by allowing both parties to make an arbitrary (but poly-bounded) amount of computation steps in one time-step.

14.1.2 Internet topology

As in the UC framework, TUC models how two machines directly communicate to each other. The internet can, thus, be represented by a network of intermediary machines that (if honest) relay the message from the sender to the destination. A partially global attacker can, of course, compromise several of these machines. Hence, we can abstract this network of machines by the information which connections between protocol parties leaks the message to the attacker. In addition to the previous model, we additionally need to specify the delay of the network, i.e., how much time the connection between two parties takes.

14.2 Description of protocols

We use two protocols namely π_I and π_V to denotes the sequence of executions performed by the involuntary and voluntary participants respectively. A protocol π is defined as a quadruplet $\{B, \Pi, S_{\texttt{CoverUp}}, S_{mix}\}$. Where $B, \Pi, S_{\texttt{CoverUp}}$ and S_{mix} denotes a browser, a set of operations, `CoverUp` server and the mix server respectively. A browser B is defined as a interactive Turing machine. B has three ports namely user port, network port and output port. B takes input from a user (and/or an attacker $\mathscr{A}$) via the user port. All requests and responses to and from remote servers such as $S_{\texttt{CoverUp}}$ and S_{mix} is done via the network port which provides a secure communication channel. The output port is utilized to send any output result to the user e.g. data received from S_{mix}. Additionally a browser can also execute a JavaScipt code which in turn can send request over network to a specific recipient (e.g. S_{mix}) via the network port. Upon receiving response, browser can send it to the user (and/or the attacker). We specify two instance of browser, namely B_I and B_V corresponding to π_I and π_V respectively. The browser instance B_V is B_I with an extension E installed in it. E is also an interactive Turing machine which introduces additional transitions to B. Hence $B_V = B_I \| E$.[7]

Algorithm 1: Challenger $Ch(\pi_b, t_{net}, t_{user})$

Notation: Ch challenger, $\mathscr{A}$ adversary, π_b protocol ($b \in \{I, V\}$), $p \in \{\mathsf{network}, \mathsf{user}\}$ the interface over which the message comes

Upon Initialization
begin
  Initialize two empty FIFO queues Q_{net}, Q_{user}

Upon Receiving m from the $\mathscr{A}$ over interface p
begin
  if $p = \mathsf{user}$ **then**
    $Q_{user}.\mathsf{push}(m)$
  else if $p = \mathsf{network}$ **then**
    $Q_{net}.\mathsf{push}(m)$

Invoke every t_{net} point in time **begin**
  $(m_1, m_2) \leftarrow Q_{net}.\mathsf{pop}()$
  Send (m_1, m_2) over the network port to π_b

Invoke every t_{user} point in time **begin**
  $(m_1, m_2) \leftarrow Q_{user}.\mathsf{pop}()$
  Send (m_1, m_2) as user inputs to π_b

Whenever π_b outputs m over the interface p **begin**
  Send (m, p) to $\mathscr{A}$

14.3 The challenger

Algorithm 1 describes the challenger $Ch(\pi, t_{net}, t_{user})$. We consider a message in form of a pair (m, p) where m is the message itself in bitstring format and p denotes to the port where m arrives. In our definition there can be three types of ports:

1. u : Denotes to the user port where there can be incoming and outgoing data flow from the browser due to user activities. We denotes the activities as Command (A) and Command (B) which relates to mouse click events on the website of the entry server and the `CoverUp`/mix server (specified in the command), respectively.
2. N : Denotes to the network port where network leakage in terms of traffic pattern is observed.
3. CS : This the port where the entry server sends and receives data. The outgoing data can be the programmed java script codes and HTML pages. The incoming data consists of the response from the java script code and the HTML page.

The challenger Ch_b relies on two *FIFO* queues Q_{NET} and $Q_{Browser_\pi}$ for input which are populated by network traffic and the browser B_b respectively. Ch_b polls to both of these queues in a predefined time interval t_q.

[7] $\|$ is defined as the combination of two state machine as described in `https://www.cs.cmu.edu/afs/cs/academic/class/15671-f95/www/handouts/shared-memory/node1.html`

14.4 Hybrid Games & main theorem

Protocol 1 B_I : abstraction of the browser in π_I (feed)

Upon Connecting to the entry server and receiving an iframe
begin
  Compose request r from the iframe
  Send r to `CoverUp` server via secure channel
Upon Receiving a JavaScript code `code` from `CoverUp` server
begin
  Execute `code` **begin**
    $h_f \leftarrow$ feed
    $data \leftarrow \{0\}^k$
    send $(h_f \| Data)$ via the secure channel to the mix server
Upon Receiving D from mix server over the secure channel
begin
  Send D to the user

Protocol 2 B_V : abstraction of the browser in π_V (bi-directional channel)

Upon Connecting to the entry server and receiving an iframe
begin
  Compose request r from the iframe
  Send r to `CoverUp` server via secure channel
Upon Receiving a JavaScript code `code` from `CoverUp` server
begin
  Execute `code` **begin**
    $Data \leftarrow$ `readBiDirectionalData()`
    Set $h_f \leftarrow$ bi − directional
    Set $ID_{int} \leftarrow \{0,1\}^k$
    Send $(h_f \| ID_{int} \| Data)$ to the mix server
Upon Receiving D from mix server over the secure channel
begin
  Send D to the user

Games 1-3 describe hybrid games which incorporate small changes over Game 0 or $Ch(\pi_I)$ (protocol executed by involuntary participants) and transform the protocol to $Ch(\pi_V)$ (protocol executed by voluntary participants) by adding up a small amount of delay.

Protocol 3 `CoverUp` (r): `CoverUp` server side computation

Upon Receiving a request r from a browser $B_{I/V}$
begin
  `code` ← JavaScript code snippet
  Send `code` to $B_{I/V}$

Protocol 4 mix server(h_f) : the mix server side constant time computation

Upon Receiving $(h_f \| ID_{int} \| Data)$ from the secure channel
begin
  $FixedExecutionTime \leftarrow x$
  $start \leftarrow$ `timeNow()`
  if $h_f = bi-directional$ **then**
    Initialize $state$ with ID_{int}
  if `stateExists()` = TRUE **then**
    Set $state \leftarrow$ `getState`(ID_{int})
    $D \leftarrow$ `covertData`$(state, Data)$
    Call `UpdateState`$(state)$
  else
    $D \leftarrow$ `broadcast`
  Sleep for $(x - (\texttt{timeNow}() - \texttt{start}))$
  Send D over the secure channel

Game 1

Ch: **Upon** Receiving (m, p) from $\mathscr{A}$ for client
begin
  if $p = u$ & $m = (\mathsf{Command}(m_1), \mathsf{Command}(m_2))$ **then**
    if $b = 0$ **then**
      Send $(\mathsf{Command}(m_1), \mathsf{Command}(m_2))$ over u to Browser(ϕ)
    else
      Send $(\mathsf{Command}(m_1), \mathsf{Command}(m_2))$ over u to Browser(ϕ)
  else
    Send m over p to Browser(ϕ)
π_I^1: **Upon** receiving a bi-directional request R from ω_U
begin
  `RequestQueue.push`(R)

Game 2

π_V^2: **Upon** Initialization at Ch side
begin
  $Data \leftarrow$ `readBiDirectionalData()`
  Set $h_f \leftarrow$ bi − directional
  Set $ID_{int} \leftarrow \{0,1\}^k$
  Send $(h_f \| ID_{int} \| Data)$ to the mix server

Game 3

π_V^3: **Upon** Receiving a request $R = (h_f \| ID_{int} \| Data)$ at the mix server side
begin
  $D \leftarrow$ mix server(h_f) (the Protocol 4 for the mix server)
  Send D to Ch

14.5 Analysis of hybrid games

Lemma 1. *Assume that π_I and π_V established a secure channel in TUC (i.e., TLS in the real implementation).*[8]

[8] Formally, a functionality $\mathscr{F}_{SCS}$, as in the UC framework, but secure in TUC (see Appendix 14.1).

Assume that all cryptographic primitives in π_V and π_I and secure in the TUC framework. Let $\pi_I + \Gamma$ be some protocol that behaves just like π_I, except that it incurs additional delays, which add up to Γ. Then, there is a Γ such that $\pi_I + \Gamma$ and π_V are indistinguishable in the sense of Definition 1 with a $\delta = \mu$ for a function μ that is negligible in the security parameter. Moreover, the timing leakage of $\pi_I + \Gamma$ and π_V is 0 for any sampling rate.

Game 0 is defined as $Ch(\pi_I, t_{net}, t_{user})$ which is the challenger choosing protocol with only feed capability (no interactive capability supported). Where as $Ch(\pi_V, t_{net}, t_{user})$ is the challenger who picks protocol instance supporting interactive communication (browsing and chatting). We assume that the operation that performs on `RequestQueue` introduces Δ_0 delay. Reading of bi-directional data from the external application imposes Δ_1 delay and modifying variables in the request payload introduces Δ_2 delay. We define S_i to be the number of operation executed in game i and $\Pr[S_i]$ denotes the probability that an attacker can distinguish game i by observing the total execution time.

Game 1-3 introduce small modification over base protocol, i.e., the broadcast channel. Every game add some small timing delay Δ to the previous game which is the only information available to $\mathscr{A}$. Henceforth we define the following notations

Notation 1.

$$\Pr[S_i + \Delta] := \Pr[0 \leftarrow \langle \mathscr{A} \mid \textit{Game i with delay } \Delta \rangle]$$

Notation 2.

$$\Pr[S_i - \Delta] = \Pr[S_j] :\Longleftrightarrow \Pr[S_i] = \Pr[S_j + \Delta]$$

We have also used the following relation throughout our proof which can be proved easily.

$$\begin{aligned}
&\Pr[S_j] = \Pr[S_i + \Delta] \wedge \Pr[S_i] = \Pr[S_k + \Delta'] \\
&\Longleftrightarrow \Pr[S_j] = \Pr[S_k + \Delta + \Delta'] \\
&\Pr[S_i + \Delta] = \Pr[S_j + \Delta + \Delta'] \\
&\Longleftrightarrow \Pr[S_i] = \Pr[S_j + \Delta'] \quad \text{(trivial using Notation 2)}
\end{aligned}$$

14.5.1 Game 0 and Game 1

Game 1 only include one operation on `RequestQueue` which imposes Δ_0 timing delay. Hence

$$\Pr[S_1] = \Pr[S_0 + \Delta_0].$$

$$\frac{\Pr[S_0]}{\Pr[S_1]} = \frac{\Pr[S_0]}{\Pr[S_0 + \Delta_0]}$$

14.5.2 Game 1 and Game 2

Game 2 adds the request intercept which executes in the browser extension at client side. This includes one call to `readBiDirectionalData()` method which reads bi-directional data sent by the external application. This incurs Δ_1 timing delay. Moreover Game 2 adds statements which modify the payload content such as the header h_f to $\mathsf{bi-directional}$ and data field to the bi-directional request. This introduce Δ_2 timing delay. Remember that ll the communications are done via a secure channel, and all the modification of the packet data ensures constant data size. Hence we can ensure indistinguishability in spite of the data modification.

$$\Pr[S_2] = \Pr[S_1 + \Delta_1 + \Delta_2].$$

$$\frac{\Pr[S_1]}{\Pr[S_2]} = \frac{\Pr[S_1]}{\Pr[S_1 + \Delta_1 + \Delta_2]} = \frac{\Pr[S_1]}{\Pr[S_0 + \Delta_0 + \Delta_1 + \Delta_2]}$$

14.5.3 Game 2 and Game 3

In Game 3 all the statements remain same, only the parameter to the reactive machine mix server(h_f) changes as the Ch now sends $\mathsf{bi-directional}$ as the packet header. As mix server(h_f) guarantees constant time execution irrespective of the input parameter, Game 3 does not introduce any additional timing delay.

$$\Pr[S_3] = \Pr[S_2 + \Delta_1 + \Delta_2].$$

$$\frac{\Pr[S_2]}{\Pr[S_3]} = \frac{\Pr[S_2]}{\Pr[S_2 + \Delta_1 + \Delta_2]} = \frac{\Pr[S_2]}{\Pr[S_0 + \Delta_o + \Delta_1 + \Delta_2]}$$

Proof. Game 3 adds total $\Delta_1 + \Delta_2$ delay (cumulative from Game 1 to 3) to Game 0 or $Ch((\pi_I), noise, T_{user}, T_{net})$. Hence

$$\Pr[S_3] = \Pr[S_0 + \Delta_0 + \Delta_1 + \Delta_2] \qquad \text{(eq 2)}$$

holds from eq 1 □

Lemma 2. *Game 3 is equivalent to the challenger $Ch(\pi_V, T_{user}, T_{net})$ who picks the* `CoverUp` *instance with covert communication mode.*

$$\Pr[S_3] = \Pr[0 \leftarrow \langle \mathscr{A} \mid Ch(\pi_V, t_{user}, t_{net}) \rangle]$$

Proof. From Lemma 1 we get $\Pr[S_6] = \Pr[S_0 + \Delta_0 + \Delta_1 + \Delta_2]$

$$\begin{aligned}
S_0 &= \mathsf{Step}(\phi + \pi_V) = \mathsf{Step}(\pi_V) \\
S_3 &= \mathsf{Step}(\pi_I^3) = \mathsf{Step}(\pi_V) + \Delta_0 + \Delta_1 + \Delta_2 \\
\Pr[S_3] &= \Pr[S_0 + \Delta_0 + \Delta_1 + \Delta_2] \qquad \text{(from eq 2)}
\end{aligned}$$

$$\begin{aligned}\Pr[S_3] &= \Pr[0 \leftarrow \langle \mathscr{A} \mid \text{Game } 3\rangle] \\ &= \Pr[0 \leftarrow \langle \mathscr{A} \mid \text{Game } 0+\Delta_0+\Delta_1+\Delta_2\rangle] \\ &= \Pr[0 \leftarrow \langle \mathscr{A} \mid \text{Game } 0+\Delta\rangle] \\ &= \Pr[0 \leftarrow \langle \mathscr{A} \mid Ch(\pi_I, t_{user}, t_{net})+\Delta\rangle] \\ &= \Pr[0 \leftarrow \langle \mathscr{A} \mid Ch(\pi_I^3, t_{user}, t_{net})\rangle] \\ &= \Pr[0 \leftarrow \langle \mathscr{A} \mid Ch(\pi_V, t_{user}, t_{net})]\end{aligned}$$

□

15 Statistical distance and the optimal attacker

Definition 2 (Discrete distributions over finite domains)**.** *A discrete distribution X is* a discrete distribution over a finite domain *if there is a natural number n such that $|supp(X)| = n$, $supp$ denotes the support. For pair X,Y of discrete distributions over finite domains, the* join domain $\Omega_{X,Y}$ *(abbreviated as Ω if uniquely determined in the context) be defined as $\Omega_{X,Y} := supp(X) \cup supp(Y)$.*

Definition 3 (Negligible leakage against unbounded attackers)**.** *Let X,Y be two families of discrete distributions over a finite domain with a joint Domain Ω. Then, the X* and *Y* have negligible leakage against unbounded attackers *if there is a negligible function μ such that for all Turing machines A and sufficiently large $\eta \in \mathbb{N}$ we have*

$$\begin{aligned}&|\Pr[b = 1 : b \leftarrow A(w), w \leftarrow X_\eta] \\ &-\Pr[b = 1 : b \leftarrow A(w), w \leftarrow Y_\eta]| \leq \mu(\eta)\end{aligned}$$

Definition 4 (Statistical distance over finite domain)**.** *Let X,Y be two discrete distributions over a finite domain with a joint Domain Ω. Then, the* statistical distance *d of X and Y is defined as*

$$d(X,Y) := \frac{1}{2}\sum_{a \in Q}(|p_X(a) - p_Y(a)|)$$

Definition 5 (Statistical indistinguishability)**.** *Let X,Y be two families of discrete distributions over a finite domain with a joint Domain Ω. X and Y are* statistically indistinguishable *if there is a negligible function μ such that for sufficiently large $\eta \in \mathbb{N}$ the statistical distance of X_η and Y_η is negligible in η, i.e., $d(X,Y) \leq \mu(\eta)$.*

Lemma 3 (Definition 3 $\Leftrightarrow$ Definition 5)**.** *Let X,Y be two discrete distributions over a finite domain with a joint Domain Ω. Then, X,Y have negligible leakage against unbounded attackers if and only if X,Y are statistically indistinguishable.*

Proof. If X and Y have negligible leakage against unbounded attackers, then X and Y are statistically indistinguishable. Otherwise, an unbounded attackers could just check for each sample w that it receives whether $p_{X_\eta}(w)$ or $p_{Y_\eta}(w)$ is large, and output 1 in one case and 0 in the other case. The resulting advantage is exactly the statistical distance, which contracts the assumption that X and Y are not statistically indistinguishable.

For the converse direction (statistical indistinguishability implies negligible leakage against unbounded attackers), we assume that statistical indistinguishability holds and there is an unbounded attacker A such that (for infinitely many η)

$$\begin{aligned}&|\Pr[b = 1 : b \leftarrow A(w), w \leftarrow X_\eta] \\ &-\Pr[b = 1 : b \leftarrow A(w), w \leftarrow Y_\eta]| \geq p(\eta)\end{aligned}$$

for some polynomial η. For the proof, we use the more general definition of statistical distance[9]

$$\begin{aligned}d(X_\eta, Y_\eta) := \max_{S \subseteq \Omega} &|\Pr[w \in S : w \leftarrow X_\eta] \\ &-\Pr[w \in S : w \leftarrow Y_\eta]|\end{aligned}$$

Statistical indistinguishability and the finiteness of the domain then implies (for some negligible function μ) for sufficiently large η that there is a set S' such that

$$|\Pr[b = 1 : b \Leftrightarrow (w \in S'), w \leftarrow X_\eta] \quad (2)$$

$$-\Pr[b = 1 : b \Leftrightarrow (w \in S'), w \leftarrow Y_\eta]| \leq \mu(\eta) \quad (3)$$

We observe, however, that the attacker A also gives rise to a set $S_A := \{w \mid A(w) = 1\}$. Then, we have

$$\begin{aligned}&|\Pr[b = 1 : b \Leftrightarrow (w \in S_w), w \leftarrow X_\eta] \\ &-\Pr[b = 1 : b \Leftrightarrow (w \in S_w), w \leftarrow Y_\eta]| \geq p(\eta) > \mu(\eta)\end{aligned}$$

Which contradicts (1), since S' was assumed to be the set that maximizes $|\Pr[w \in S : w \leftarrow X_\eta] - \Pr[w \in S : w \leftarrow Y_\eta]|$ and already for S' we have

$$|\Pr[w \in S : w \leftarrow X_\eta] - \Pr[w \in S : w \leftarrow Y_\eta]| \leq \mu(\eta)$$

for sufficiently large η, however, for the set S_A we have

$$|\Pr[w \in S : w \leftarrow X_\eta] - \Pr[w \in S : w \leftarrow Y_\eta]| \geq p(\eta) > \mu(\eta)$$

□

16 Composition theorem

We recall a known result for the statistical distance of two product distributions D_0^i and D_1^i with finite domains.

[9] For a proof of the equivalence to Definition 4 can be found here: `https://wiki.cc.gatech.edu/theory/images/b/b2/Lec5.pdf`

Lemma 4. *Let D_0 and D_1 a pair of distributions with finite domains and a statistical distance (i.e., total variance) of δ. Let product distributions $D_0^i := D_0 \times \cdots \times D_0$ and $D_1^i := D_1 \times \cdots \times D_1$ be the respective product distributions, resulting from i iterative self-compositions. Then, for all $i \in \mathbb{N}$ the statistical distance of D_0^i and D_1^i is given by the following recursive formula:*

$$\begin{aligned}\delta_0 &:= \delta \\ \delta_i &:= \delta_{i-1} + (1-\delta_{i-1}) \cdot \delta\end{aligned}$$

Proof. Recall the definition of statistical distance from the proof of Lemma 3

$$SD(D_0^i, D_1^i) \big| \Pr[x \in S : x \leftarrow D_0^i] - \Pr[x \in S : x \leftarrow D_0^i] \big|$$

We will show that $SD(D_0^i, D_1^i) \leq \delta_{i-1} + (1-\delta_{i-1}) \cdot \delta$ for all $i > 1$. By induction, the statement of the lemma then follows.

$$\begin{aligned}
&\Pr[\underbrace{x \in S}_{=:E_0^i} : \underbrace{x \leftarrow D_0^i}_{=:\Omega_0^i}] \\
&\overset{(1)}{=} \Pr[E_0^{i-1} \vee (\neg E_0^{i-1} \wedge E_0) : \Omega_0^i] \\
&= \Pr[E_0^{i-1} : \Omega_0^{i-1}] + (1 - \Pr[E_0^{i-1} : \Omega_0^i]) \cdot \Pr[E_0 : \Omega_0] \\
&\quad - \underbrace{\Pr[E_0^{i-1} \wedge (\neg E_0^{i-1} \wedge E_0)] : \Omega_0^i}_{=0} \\
&\Rightarrow | \Pr[E_0^i : \Omega_0^i] - \Pr[E_0^i : \Omega_1^i] | \\
&= \big| \Pr[E_0^{i-1} : \Omega_0^{i-1}] + (1 - \Pr[E_0^{i-1} : \Omega_0^i]) \cdot \Pr[E_0 : \Omega_0] \\
&\quad - \Pr[E_0^{i-1} : \Omega_1^{i-1}] - (1 - \Pr[E_0^{i-1} : \Omega_1^i]) \cdot \Pr[E_0 : \Omega_1]) \big| \\
&= \big| \Pr[E_0^{i-1} : \Omega_0^{i-1}] - \Pr[E_0^{i-1} : \Omega_1^{i-1}] + \\
&\quad (1 - \Pr[E_0^{i-1} : \Omega_0^i]) \cdot \Pr[E_0 : \Omega_0] \\
&\quad - (1 - \Pr[E_0^{i-1} : \Omega_1^i]) \cdot \Pr[E_0 : \Omega_1]) \big| \\
&\leq \delta_{i-1} + \delta - | \Pr[E_0^{i-1} : \Omega_0^i]) \cdot \Pr[E_0 : \Omega_0] \\
&\qquad\qquad - \Pr[E_0^{i-1} : \Omega_1^i]) \cdot \Pr[E_0 : \Omega_1] |
\end{aligned}$$

(1) holds since D_b^i is a product distribution.

Let $a := \Pr[E_0^{i-1} : \Omega_0^{i-1}]$, $b := \Pr[E_0 : \Omega_0]$, $c := \Pr[E_0^{i-1} : \Omega_1^{i-1}]$, and $d := \Pr[E_0 : \Omega_1]$. Assume w.l.o.g. that $b \geq d$ holds. Then, also $a \cdot b \geq c \cdot d$ and $a \geq c$ holds. Since $a > c$ holds, $ab - cd \geq a(b-d)$. Since $b > d$ holds, $|ab - cd| \geq |a(b-d)|$. Again since $a > c$ holds, we have $a(b-d) \geq |a-c| \cdot |b-d|$, and thus

$$|ab - cd| \geq |a-c| \cdot |b-d| = \delta_{i-1} \cdot \delta$$

An analogous argumentation holds for the other case, i.e., $d > b$. Since $|ab - cd| < \delta_{i-1} + \delta$, we get

$$\begin{aligned}
&\delta_{i-1} + \delta - \\
&\underbrace{| \Pr[E_0^{i-1} : \Omega_0^i]) \cdot \Pr[E_0 : \Omega_0] - \Pr[E_0^{i-1} : \Omega_1^i]) \cdot \Pr[E_0 : \Omega_1] |}_{\geq \delta_{i-1} \cdot \delta} \\
&\leq \delta_{i-1} + (1 - \delta_{i-1})\delta = \delta_i
\end{aligned}$$

□