Deniable Upload and Download via Passive Participation

David Sommer *ETH Zürich*
Aritra Dhar *ETH Zürich*
Esfandiar Mohammadi *ETH Zürich*
Daniel Ronzani *Ronzani Schlauri Attorneys*
Srdjan Čapkun *ETH Zürich*

Abstract

Downloading or uploading controversial information can put users at risk, making them hesitant to access or share such information. While anonymous communication networks (ACNs) are designed to hide communication meta-data, already connecting to an ACN can raise suspicion. In order to enable plausible deniability while providing or accessing controversial information, we design CoverUp: a system that enables users to asynchronously upload and download data. The key idea is to involve visitors from a collaborating website. This website serves a JavaScript snippet, which, after user's consent produces cover traffic for the controversial site / content. This cover traffic is indistinguishable from the traffic of participants interested in the controversial content; hence, they can deny that they actually up- or downloaded any data.

CoverUp provides a feed-receiver that achieves a downlink rate of 10 to 50 Kbit/s. The indistinguishability guarantee of the feed-receiver holds against strong global network-level attackers who control everything except for the user's machine. We extend CoverUp to a full upload and download system with a rate of 10 up to 50 Kbit/s. In this case, we additionally need the integrity of the JavaScript snippet, for which we introduce a trusted party. The analysis of our prototype shows a very small timing leakage, even after half a year of continual observation. Finally, as passive participation raises ethical and legal concerns for the collaborating websites and the visitors of the collaborating website, we discuss these concerns and describe how they can be addressed.

1 Introduction

Access to and distribution of sensitive and controversial information often comes at risk for users. Due to the risk of being observed, users might be reluctant to download or upload certain content. Even if the content itself is end-to-end encrypted, the fact that the user accessed a particular domain or used an anonymity network might already indicate his interest in the particular content. Since Edward Snowden's revelations, we know that surveillance is mostly based on meta-data, such as source and destination IP, timestamps, and the size of the data [54].

Solutions like anonymous communication networks (ACN) are designed to hide such meta-data. Despite that, even the strongest ACNs in literature [68, 66, 40, 60] do not protect against global network attackers and do not hide users' participation in the ACN, except for the brute-force method of continuously producing artificial traffic [30]. This participation in an ACN alone can appear suspicious. Participation time can be used in long-term statistical disclosure attacks to re-identify the user, thereby downgrading the anonymity properties of an ACN [43, 44].

In this paper, we aim to solve this issue in the case of asynchronous upload and download and therefore address the following problem: how to allow users to safely download and upload content without the fear of their intentions being identified. This problem is different from the more general problem of anonymous communication. Namely, content upload and download is asynchronous, typically allows for high latency, and is therefore much less vulnerable to timing correlations. Additionally, we aim to achieve a stronger anonymity property: we require that the participation (time) of users is protected.

Our approach to solving this problem is to draw in visitors (*Passive Participants*) of highly accessed websites and trigger them via JavaScript to create *cover* traffic to the controversial content server. Additionally, we ensure that the passive participants' traffic is indistinguishable from active participants', who are genuinely interested in downloading/uploading the content, thereby enabling deniable communication.

Finally, prior work [41, 62] neglected vital implementation details, such as system timing leakage. They left, in particular, three major challenges unsolved. (i) How to construct a downlink connection (using the browser) that relays data to an external program with minimal timing leakage. (ii) How to relay data from an external program to the uplink connection (using the browser) with minimal timing leakage. (*iii*) How long can such a system be safely used before the timing leakage renders active participants clearly distinguishable?

Contributions. We address these three challenges.

- We design a deniable uni-directional channel to a broadcast server, CU:Feed, that can deliver data to an external program, called COVERUP-Tool, for highly anonymity-sensitive users (Challenge (*i*)) with a throughput of 10 Kbit/s and up to 50 Kbit/s. Such message feeds are suited for the transmission of information that a user does not want to be caught reading (e.g., sensitive medical information, leaked documents, or an e-mail list of an incriminating web service). We protect passive participants from potentially incriminating information by enforcing that a participant's machine never contains enough data chunks to reconstruct any incriminating information. CU:Feed achieves deniability against a global network-level attackers that controls all parties except for the user' machine.
- We extend CU:Feed to a deniable bi-directional channel CU:Transfer, which enables selective data download and data upload to a content server. Active users install for CU:Transfer a browser extension that replaces dummy traffic with upstream data and communicates with COVERUP-Tool (Challenge (*ii*)). CU:Transfer assumes an honest-but-curious party (the COVERUP *server*) that serves the active and passive participants' JavaScript code. CU:Transfer achieves deniability against a global network-level attacker that controls all parties except for the user's machine, the COVERUP server, and the Transfer server.
- For both channels, we implemented a prototype that carefully minimize the timing leakage (Challenge (*iii*)). The prototype includes an entry server, the COVERUP server that serves the COVERUP JavaScript client code. For active participants, we additionally provide a browser extension and an external tool that enable participants to interact with the COVERUP system. The COVERUP downlink and uplink rate of our prototype is between 10 and 50 Kbit/s, depending on the bandwidth overhead, and the expected latency is 60 seconds.
- We experimentally evaluate the timing-leakage of our prototypes by measuring the differences between active users and cover-traffic generating passive users in the network timing-delays (Challenge (*iii*)). We show that their usage patterns are hard to distinguish, and for half a year of continual observation[1] with a latency of 60s we are able to bound the attacker's advantage of distinguishing these usage patterns with $2 \cdot 10^{-3}$, i.e., the chance of successfully deciding whether a user is active or passive is 50.001%.[2]

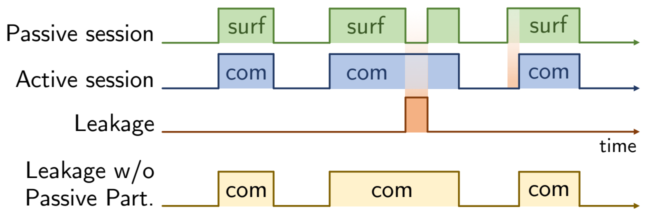

Figure 1: Passive participation can hide the participation time for a file-sharing protocol. The x-axis is the time, and the y-axis show whether at that time surfing or protocol-communicating behavior is expected. Only communicating activity which is not covered by the *expected* surfing behavior creates leakage. Active participants that produce protocol-communication only produce leakage during time where they would normally not surf.

- We discuss ethical and selected legal questions w.r.t. the entry server and the passive participants.

2 Problem description

The goal of this paper is to enable users to safely up- and download content without the fear of their intentions being identified. The concrete problem is to enable users to hide their up- and download activities among traffic that is produced by other normal web users. This problem is different from the more general problem of anonymous communication, as our goal is to utilize the traffic of normal web surfers.

2.1 Passive participation

One approach for utilizing the traffic of normal web surfers of highly accessed websites (the *entry server*) is passive participation: compel web surfers (*passive participants*) to create cover traffic to the content server in a non-invasive manner and such that their traffic is indistinguishable from *active participants* (which are genuinely interested in downloading/uploading the content). As a result, active participants can deny that they up- or downloaded any data during their normal surfing time on the entry server websites, protecting their *participation time* in the file-sharing protocol.

The degree of plausible deniability depends on whether the active participants manage to let their surfing behavior towards the entry server unchanged. For users that are willing to make a paradigm shift, COVERUP offers strong guarantees. Instead of activating COVERUP whenever a deniable up- or download channel is needed, COVERUP gives the highest degree of privacy if users let it run in the background. For asynchronous up- and downloads, COVERUP can just up- and download opportunistically, whenever an active participant is anyway visiting the entry server. Moreover, the wider COVERUP is deployed, the lower is the need

[1]We assume a usage pattern of at most 50 times a day, and at most 5 hours per day in total.

[2]This advantage is very low, since (in contrast to some usages of cryptographic schemes) COVERUP is a system that has limited exposure. Thus, an attacker cannot get arbitrarily many samples to amplify his or her chance to guess correctly to a clear decision.

of a user to adapt its behavior to gain more throughput.

Even with imperfect behavior, this approach provides partial cover and delays a potential detection. First, consider the case where the browsing behavior of the active participants towards the entry server does not change. There, using CoverUp can provide deniability for the act of utilization. In contrast, a slightly altered user behavior leaks its *difference* to the unaltered behavior. However, this difference is smaller than the full leakage without CoverUp, as connecting directly to a service already reveals intention. Figure 1 illustrates this property by focusing on the participation time.

2.2 Challenges

We consider an attacker that controls the network but the user's machine and the Transfer server (the file server) are honest, and a dedicated party (the CoverUp server) that serves the protocol code for active and passive participants as a JavaScript snippet is honest but curious. Even in the presence of such an honest-but-curious CoverUp server and an honest Transfer server, the browser's processing time of active and passive participants can potentially leak information. This problem is amplified, since a network-level attacker can change the TCP flag for timestamps and compel the victim's operating system to add OS-level timestamps to the TCP headers [25]; hence, there is no hope of network-noise blurring the timing leakage. This leads us to three major challenges that we study in this work. (*i*) How to construct a *deniable* downlink connection that relays data to an external program with minimal timing leakage? (*ii*) How to relay data from an external program to a *deniable* uplink connection with minimal timing leakage. (*iii*) How long can such a system be safely used before the timing leakage renders active participants clearly distinguishable?

2.3 Non-goals

In the problem area of passive participation, two challenges remain that are out of scope of this work.

Behavior-changes towards the entry server. The usage of CoverUp may unconsciously influence the behavior of active participants, e.g. if active users spend more time on a specific entry server in order to use CoverUp. We believe, however, that these behavior changes do not cause a large amount of leakage as CoverUp is meant for asynchronous up- and download of files; hence, it is less prone to timing correlations (e.g., intersection attacks) than synchronous applications, such as messaging. As a consequence, the only source of leakage would be users that keep the tabs longer open in the background with CoverUp. Recent studies show that many users keep tabs open in the background anyway [53]; hence, CoverUp would not cause significant privacy leakage for these users. Properly understanding these behavior changes requires a thorough user study, which is out of scope of this work.

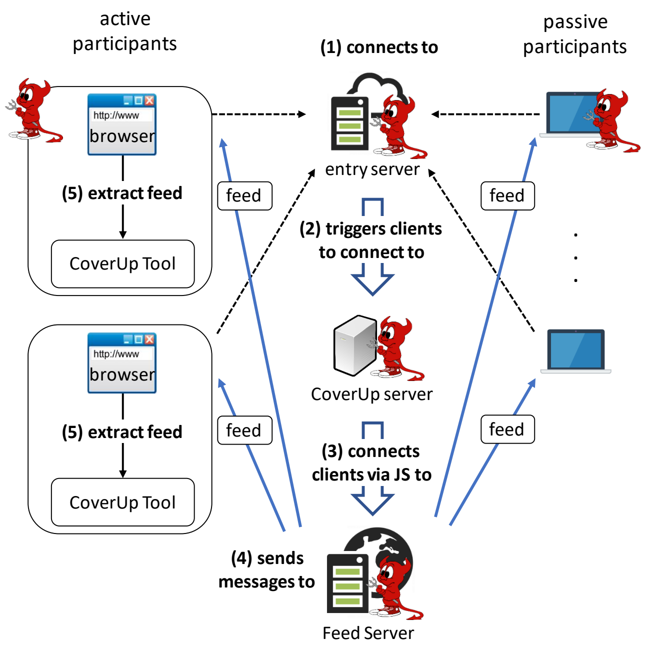

Figure 2: Main components of CoverUp for CU:Feed. All visitors of an entry server are redirected to the CoverUp server, triggered to send (dummy) requests to the Feed server, and then receive an encoded piece of a uni-directional message feed (4), which is extracted (5) by active participants via the CoverUp-Tool.

Browsing time of passive participants. Passive participants potentially reveal their browsing behavior to the CoverUp infrastructure, as a malicious server can read HTTP header's referrer field. While this leakage exists, we would like to put it into perspective. Many popular websites already leak this information to other services, such as advertisement networks or external analytic tools, such as Google Analytics. A deeper analysis of this leakage is out of scope.

3 CoverUp

Passive participation raises the challenge of utilizing passive participants to produce cover traffic with unintrusive technologies while asking for not more than an informed consent[3] and while keeping the traffic of active and passive participants indistinguishable. This section details how we overcome these technical challenges and presents the system design of CoverUp. We split CoverUp into two parts based on their features: a uni-directional broadcast-receiver channel and a fully bi-directional channel. We call them CU:Feed and CU:Transfer respectively.

[3]We discuss the challenges of an informed consent in Section 6.

3.1 CU:Feed

The uni-directional channel CU:Feed implements a deniable feed-receiver for a feed that is broadcast by a dedicated *Feed server*. CU:Feed triggers visitors (the *passive* participants) of cooperating websites (the *entry* server) to produce cover traffic, after they give an informed consent. CU:Feed leverages unintrusive widely-used JavaScript functionality of browsers. For *active* participants, which are interested in the feed, CU:Feed performs the same steps, but we additionally provide the external application COVERUP-Tool. With this application, the feed's content can be extracted from the browser's cache. As *active* users are indistinguishable from *passive* ones for all involved parties except their own machine, they cannot chose the feed they are listen to. The entry server could be a university, a knowledge, or a news site.

As illustrated in Figure 2, CU:Feed performs as follows:

(1) The user connects to the entry server. The entry server embeds in its HTML-code an iframe to a dedicated server (the COVERUP *server*) from a different domain.
(2) The COVERUP server responds with a JS code snippet.
(3) This JS snippet triggers the browser of the entry server's visitors to send requests to the *Feed server*.
(4) The *Feed server* responds with CU:Feed packets. This effectively produces cover traffic to and from the Feed server. The COVERUP JS snippet then stores the most recent CU:Feed packet in the browser's `localStorage` cache, thereby overwriting the old one.

The passive participants of COVERUP stop here. The rest of the protocol is only executed by the active participants.

(5) An active participant uses a previously obtained external application, called COVERUP-Tool[4], to extracts these CU:Feed packets from the browser's cache, which is stored on the disk.

CU:Feed executes the same steps for active and passive participants except that active participants additionally install COVERUP-Tool on their computer to extract the feed. This makes the active and the passive participants indistinguishable to a network level adversary who does not compromise a user's system. As the CU:Feed has no strict latency requirements, the browser behavior of active participants can be kept exactly the same, thus avoiding timing leakage.

With regard to the privacy of participants, the JS snippet from the COVERUP server is in an isolated context and thus can not learn anything from other contexts (including the page the iframe is embeeded in) due to the SAME-ORIGIN-POLICY [8]. Hence, the COVERUP server can only learn when a participant visited the entry server, by the requests.

The content of the Feed could be controversial. To deflect potential legal harm to the passive participants, we cryptographically protect them from accidentally storing meaningful parts of the CU:Feed on their disc by utilizing an ALL-OR-NOTHING SCHEME [61] and only storing one CU:Feed packet in their `localStorage`. Without actively trying to, passive participants do not have sufficient packets collected to potentially reconstruct any content of the feed.

After applying the All-or-Nothing protection, we use error-correcting FOUNTAIN CODE (see Section 4.1) on the protected feed content. This splits the content in many packets and enables COVERUP-Tool to assemble these CU:Feed packets in an arbitrary order and with potentially missing packets, as the feed content might be too big for a single request. Thereby, the Feed server does not need to know which packet has reached a user and in which order. As there is no difference in feed packets for an *active* and *passive* participant, CU:Feed does not require TLS. The authenticity of the feed can be achieved by signing the content, assuming a PKI.

Trust assumptions and attacker capabilities. The CU:Feed is resistant against a global network-level active attacker that controls all parts of the system except the active participant's hardware, operating system, and its running applications, as the only difference between active and passive participants is COVERUP-Tool that reads browser's cache (`localStorage`). This attacker is active, so he can modify, drop or delay any number of messages, which includes the creation of an arbitrary number of participants – passive or active as individual participants are independent of each other. As we focus on guaranteed anonymity and not on integrity, COVERUP is not censorship resistant as it cannot protect from denial of service.

3.2 CU:Transfer

We extend CU:Feed to CU:Transfer, which enables user to upload content to and download content from a file server (Transfer server). Active CU:Transfer participants have to additionally install the COVERUP browser extension that establishes a channel to the external COVERUP-Tool, which can be used to upload and download content.

The protocol of CU:Transfer is almost the same as CU:Feed. While passive participants only transmit dummy and receive CU:Feed messages, active participants of CU:Transfer also transmit dummy messages unless they need to transmit content (Figure 3, Step 1). In those cases, they use native messaging to connect from the COVERUP-Tool to the browser extension (Step 2). The browser extension then replaces a CU:Feed request with a real message content (Step 3). All messages (Step 4) are encrypted by TLS, are of the same size, and are transmitted at regular time intervals. Hence, messages of passive participants are indistinguishable from messages of active participants for a global network-level adversary. Upon receiving the encrypted message (Step 5), the browser extension records it (Step 6), and sends it via native messaging (Step 7) to the COVERUP-Tool

[4]COVERUP-Tool could be obtained off-the-record or as part of the CU:Feed. There, a small program including explanation could be distributed in clear text and without any encoding which could be extracted from the cache manually. This program assembles the full COVERUP-Tool delivered by the encoded feed.

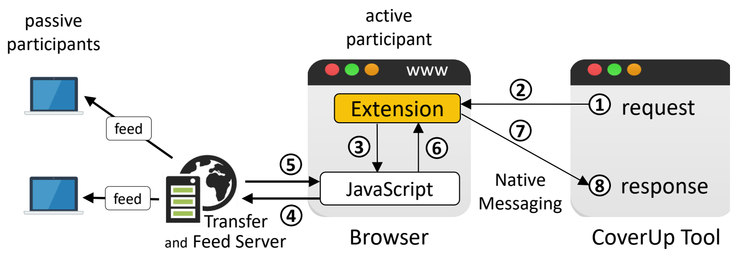

Figure 3: CU:Transfer in combination with CU:Feed. Once the JS snippet has been received, all participants request CU:Feed packets. An active CU:Transfer participant can use the extension to replace these requests to the Transfer server with custom requests. To render the traffic from passive and active users indistinguishable, we use TLS encryption at step 4 and 5, and at all connections by passive participants - in contrast to CU:Feed. For CU:Transfer the dummy messages do not need to contain feed content; they can also purely contain garbage.

which decrypts the content (Step 8).

As a result, both active and passive participants constantly send requests to the Transfer server, and the Transfer server responds with a constant-size data chunk; in particular, larger files are sent in smaller chunks. In general, the Transfer server does not need to be a centralized entity. Traffic sharing solutions (content distribution networks) could be used.

CU:Transfer trust assumptions. CU:Transfer is resistant against global network-level attacker that control all parts of the system except for the active participant's machine, the CoverUp server, and the Transfer server. The CoverUp server has to be trusted because CU:Transfer relies on the integrity of the JS snippet; otherwise an attacker can inject malicious JS code that can detect active participants (e.g., by testing for the existence of the extension[5]). To enable the browser extension to check the integrity of the JavaScript snippet with minimal timing leakage, we trust the CoverUp server to be honest-but-curious for CU:Transfer and the browser extension simply checks whether the origin of the JavaScript code snippet is as expected. If the check fails, the browser extension does not hijack any packets.[6]

We trust the Transfer server as hiding access pattern is a non-trivial problem for which current solutions either require prohibitively high communication complexity or are unsuitable in a bandwidth limited multi-user setting [30, 64].

3.3 Timing leakage

In CU:Feed active participants need to run CoverUp-Tool to extract information from the browser. While this application is external to the browser and does not directly interact with it, they share system-wide computation resources, which influences the browser's computation time. In CU:Transfer, the client additionally installs a browser extension and hence directly influences the browser's computation time. In both cases, the timing pattern of the issued web requests is influenced (in the order of milliseconds) and this is noticeable by a network-level attacker. While CU:Feed causes minor timing leakages, CU:Transfer causes significantly more timing leakage.

This timing leakage cannot be countered by introducing deterministic delays, as a JavaScript program cannot measure the sending processing time of the systems outside of its context. For the same reason, it cannot precisely enforce a delay. Therefore, we introduce random delays and show in Section 5 that these random delays significantly reduce the timing leakage. To limit the amplification of the timing leakage, we additionally limit the number of requests for which the browser extension (of an active participant) is active. This limits the risk of malicious entry servers triggering excessive amounts of page-loads, which would enable an attacker to dramatically increase the number of observations and in turn increase the timing leakage.

Figure 4 illustrates all potential observations of a network-layer attacker and the timeline of how messages are sent, received, processed in the browser, and when random delays (i.e., noise) are added. The system delay s_i in this figure refers to the system's computation time (including delays caused by the OS, the browser, and the network card). Any computation – and for CU:Transfer the communication with the extension – takes place in c_i with minimal interference.

In the rest of the paper, we concentrate on two time measurements that an attacker can perform: i) *Loading measurements* denote the time between the reception of the JavaScript snippet from the CoverUp server and the first outgoing request to the Feed/Transfer server, and ii) *Periodic measurements* denote the time between subsequent CoverUp requests to the Feed/Transfer server. For the CU:Transfer case, Figure 5 shows distributions of timing delays of active and passive participants for Loading and for Periodic measurements, without adding delays. It illustrates the importance of adding random delays; without these delays, already the naked eye can distinguish the distributions.

4 Prototype & performance

This section describes the CoverUp prototype implementation and presents its performance. We stress that main purpose of the prototype is the timing leakage evaluation. Hence, our implementation only contains a dummy feed

[5]Modern browser claim to prevent any page-loaded JavaScript from checking for installed WebExtensions unless the extension wants to reveal itself. Specifically, any content-scripts run undetectable by page-loaded JS in an isolated context [4] and any access to resources of an extension must be allowed explicitly by the extension [19].

[6]Trusting the CoverUp server can be circumvented by checking the integrity of the JavaScript code byte for byte. This would eliminate any costs associated with running such an honest-but-curious CoverUp server, but the timing leakage by such a solution would be significantly higher.

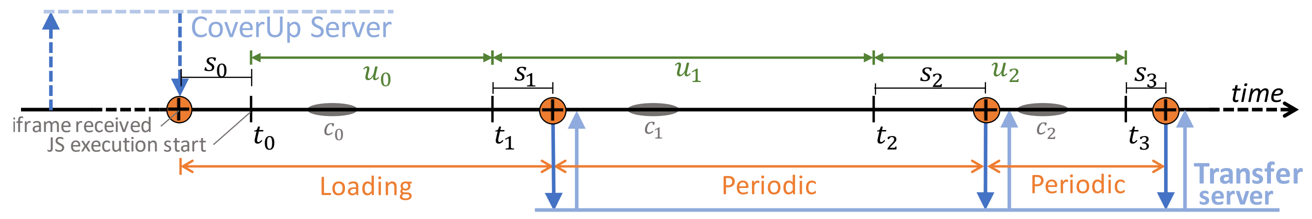

Figure 4: The timeline of an active or passive participant in the browser, starting at a request to the CoverUp Server for a JavaScript code snippet from an iframe. The code is executed and makes continuous requests to the Feed/Transfer server. The attacker can measure network timestamps of the requests (⨁). To decrease the leakage s_i of the system or browser internals, we add randomly chosen delays u_i to the sending times t_i. There are two main sources of leakage: the set-up of the iframe context (Loading) and the interval between the consecutive requests (Periodic). Any comprehensive computation c_i inside the script or by the browser extension (for active participants) is done between the sending intervals when all components are idle.

u_i = artificially added noise
t_i = XMLHttpRequest.send() call
s_i = system noise
c_i = computation inside the script
⨁ = time-stamp measurement

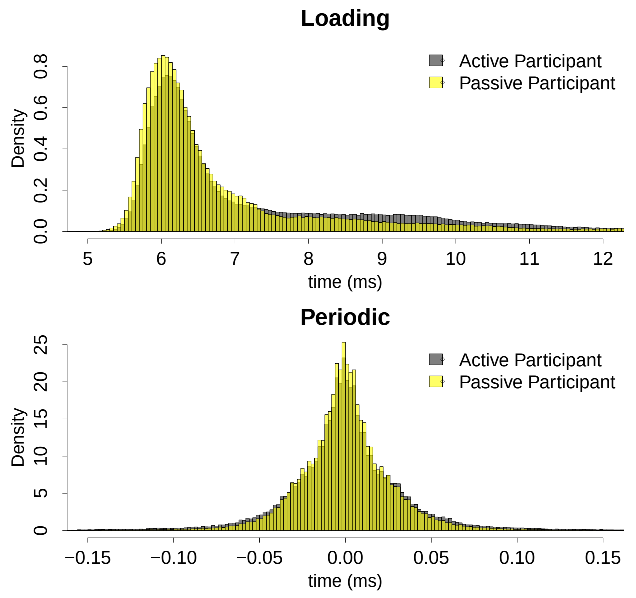

Figure 5: Distribution of timing (without additional noise) of Loading and Periodic measurements run on Linux. Each of the graphs overlays the timing distributions of active and passive participants. For the Periodic measurements, we substraced the expectation value (it is centered around 0).

server and a dummy Transfer server.

4.1 Prototype implementation

We implemented a prototype and made it available under `http://coverup.tech`. It delivers a feed and the upload and download system, for which we implemented a high-latency mailbox. The CoverUp implementation consists of five components: a CoverUp server, a dummy central server that acts as the Transfer server, in the protocol the mailbox server, the message relay and the broadcaster, an external application (CoverUp-Tool), a browser extension, and a short JS code snippet. The CoverUp server and the Feed/Transfer 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 in Google Chrome browser using the JS WebExtensions API. The JS code served by the entry server is kept at the CoverUp server. The CoverUp-Tool and the server implementation consists of about 14 KLoC and the browser extension of about 200 LoC.

We make the following four assumptions about the browser, which are in line with Chrome's explicitly stated security policies. 1. iframes are isolated, which we need for the code integrity of CoverUp's JS snippet. The parent page of the iframe cannot modify the iframe if the iframe is originated (domain) from a source other than the parent [6]. 2. a JS code cannot read from or write to another context of a different domain source without its consent. 3. the JS code can write a small amount of data to the browser's localStorage cache and this cache cannot be accessed by another JS code which originates from a different origin. This property is known as the "same-origin-policy" [8], and all modern browsers claim to enforce it.

CU:Feed. For the CU:Feed, all the users of the entry server receive identical broadcast content which is encoded with a fountain code [56]. Such encoding ensures that any out of order threshold amount of broadcast packet can recover the data successfully. Our prototype implementation uses an `XOR` based fountain code (for details see Appendix A.1). The JS snippet served by the CoverUp server stores the fountain pieces in the cache database file located on the mass storage (known as browser `localstorage`). The CoverUp-Tool collects and assembles the fountain pieces from the `localstorage`. Our implementation also employs an All-or-Nothing-Encryption scheme (one similar to [61]) which ensures that one needs threshold amount of pieces of the fountain (i.e. the entire source data) to decrypt it. The JS snippet only keeps one fountain piece in the `localstorage` to ensure that the passive users do not have any sensitive content on their disk in decipherable form.

CU:Transfer. We extend the uni-directional CU:Feed to the bi-directional CU:Transfer (recall Section 3.2), which provides a up- and download channel for arbitrary data. As a example protocol, we implemented a high-latency mailbox protocol which enables the secure exchange of messages between active users. The implementation of the protocol involves indexing the messages as POP (post office protocol [26]) where the indexing is done by public addresses of the clients. This public address is derived from the *curve25519* [34] public keys (first 48 bits of the hashed public key). Additionally, the Transfer server indexes these public addresses by the SSL identifier of the TLS client request channel. The SSL identifier is an unique id that is pseudo randomly generated during the start of a TLS session and dependent on the session key. The Transfer server uniquely identifies a sender/receiver of an incoming request by its SSL identifier without the overhead of sending an additional identification token. This prevents the attacker also from hijacking the session. The CU:Transfer 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 (using Diffie-Hellman key exchange) from the long-term key pairs. The current prototype of COVERUP does not provide forward secrecy, but one can easily integrate such feature into the messenger. Whenever a new message arrives from a source address, the Transfer server keeps the message to the index of the destination address. When a request arrives at the destination address, the Transfer server delivers the message as the response and removes the message from the previously kept index location.

4.2 COVERUP performance

We estimate COVERUP's overhead, latency, and throughput to demonstrate that it can perform reasonably well in a real-world scenario, is feasible for deployment in large scale and does not incur an intolerable overhead. COVERUP has three adjustable system parameters: request payload size, response payload size and the average request frequency, which is the average requesting rate for CU:Feed packets after adding artificial noise. Increasing the payload increases the traffic overhead for passive participants; and a lower request frequency leaves room for more artificial noise and thus increases privacy. Hence, there is a natural trade-off between the latency and privacy and the amount of traffic overhead caused and throughput of the system. In our prototype implementation, the request/response payload size is in the range of 75 KB to 375 KB. We send a request every 60 seconds in average for CU:Transfer and CU:Feed. Section 5 evaluates our choices for these system parameters.

Computational overhead. The computational overhead of COVERUP's JS executed in the Browser is negligible. Our implementation of the COVERUP-Tool takes around 50 MB of main memory and less than 1% CPU time. Similarly, the COVERUP browser extension incurs an almost unnoticeable amount of memory and CPU consumption.

Traffic overhead. The traffic overhead of CU:Feed and CU:Transfer are identical, as they are indistinguishable by design. The entry server's overhead is minimal: Only the size of the iframe tag in its HTML code. The passive participants' traffic overhead depends on the system parameters. We based our estimation of the system parameters on the Alexa top 15 news sites, in particular since the privacy improvements of COVERUP's passive-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 and will grow in near future. 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.

COVERUP is parametric in the packet size. Once fixed, the traffic overhead for the passive users is proportional to this packet length. We generously assume a passive participant that has a daily connected to the entry server for 5 hours each day. This participants would have 22MB ($\sim 5 \cdot 60 \cdot 60s \cdot \frac{1}{60s} \cdot 75$KB) to 110 MB ($\sim 5 \cdot 60 \cdot 60s \cdot \frac{1}{60s} \cdot 375$KB) of data overhead per day and 660MB ($= 30 \cdot 22$ MB) to 3.3 GB ($= 30 \cdot 110$MB) per month. For landline data flat-rates (i.e., for non-mobile visitors), 22 MB is not significant, e.g., in comparison to the traffic caused by streaming videos. We envision a deployment of COVERUP not to include mobile users. But it may be possible in near future due to the increased bandwidth of the mobile networks. Section 6 further discusses the ethical aspects of using the passive participants' resources after an explicit consent.

Latency & throughput. 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 passive participants towards the entry server's sites. Depending on the service that the entry server offers, it might be common to keep the tab open (in the background) for a long time, or to visit the site more than a few times a day, and even to switch to another entry server if multiple are available. COVERUP achieves 10 to 50 Kbits/s (for different the fixed length packet size system parameter 75 to 375 KB) throughput and a latency of around 60 seconds in average (delay between consecutive messages). As the future size of websites and the Internet infrastructure will evolve, COVERUP's packet size system parameter can be increase and thus data usage can be adapted to deliver a higher throughput. Section 5 explains our choice for the delays.

Scalability. For the participants the workload of the COVERUP channel itself is independent of the number of participants, and for the Transfer server the workload linearly increases. Hence, for the participants COVERUP scales well, and for the Transfer server an infrastructure at the scale of the entry server suffices, rendering COVERUP practical with current infrastructure.

5 Timing leakage experiments

How much timing leakage does COVERUP cause? To answer this question, we set up an experiment that measures the timing leakage. These measurement-experiments produce histograms that we use as models for the underlying processes. We use these models to estimate the privacy leakage under continual observation.

5.1 Experimental set-up

We assume that the dominant part of the timing leakage will be visible from two kinds of measurements: *Loading* and *Periodic* measurements, as depicted by the orange arrows in Figure 4. In Loading measurements, we force the iframe to refresh on the entry server page in the browser. In the corresponding TCP dump, we measure the timing difference between the response of the initial iframe HTML source request and its first ("passive") request to the Feed/Transfer server. This forces to load the extension's content script and thus captures any distinguishing feature (any timing delay added by the existence of the browser extension) produced by the extension.

The Periodic measurements model the scenario where the active and passive participants load the iframe once, followed by JavaScript generated periodic requests to the Feed/Transfer server and their response. In the network traffic dump, we look for the timing difference for two contiguous CU:Feed/CU:Transfer requests from the browser. Section 5.5 discusses the choice to concentrate on these measurements. For both measurements, we compare the timing measurement distributions of a passive participant against the distribution of an active participant.

To simulate realistic scenarios, we set up the passive and both kinds (CU:Transfer and CU:Feed) of active participants on 12 identical systems running Windows 10 and Ubuntu 16.04 (both x86-64 and in dual-boot configuration) equipped with an Intel Core i5-2400 3.1 GHz CPU and 8 GB of main memory. Additionally, the COVERUP and a dummy implementation of a Feed/Transfer server run as an Apache Tomcat web server instance (works as an ACN, a messenger relay and a Feed server) on a separate machine in the same subnet connected by a 10 Gbps switch. We ended up with 3.8 million measurements in total.

All of the communications between the server and the browser are executed over a local GigaBit Ethernet network. We use *tshark* [28] to capture all such network traffic on the participant's network interface. We compare the distributions of timing traces produced by an active participant to the distribution produced by a passive participant. All the experiments are conducted on these set-ups to investigate the timing leakage of the browser caused by COVERUP's browser extension and external COVERUP-Tool.

Reflecting the attacker model. The attacker model (see Section 3.1) 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 [25], we conduct all measurements in the settings where participants, the COVERUP server, and the Feed/Transfer server are in the same GigaBit Ethernet switched network. According to our measurements, the accuracies of the added time-stamps are $4000\mu s$ for Linux, and $400\mu s$ for Windows respectively.

Test modes. We emulate primarily three different user scenarios by using various combinations of the browser extension and the COVERUP-Tool. We use Google Chrome browser v57.0 to run our extension. The extension and the COVERUP-Tool communicate via the native messaging interface (via STDIO). The three different test modes include:

1. *Passive participant*: Google chrome with no extension and no COVERUP-Tool running.
2. *Active CU:Transfer participant*: Google Chrome with the extension installed and the COVERUP-Tool running which communicates with the aforementioned browser extension by the native messaging interface.
3. *Active CU:Feed participant*: Google chrome with no extension and COVERUP-Tool running assembling CU:Feed chunk from the browser `localStorage`.

These are repeated for both Loading and Periodic measurements (they are described in Section 5.3).

Interfering processes. Additionally we constructed one user profile in Linux to understand how the execution of other browsing tabs influences the timing leakage. To demonstrate a simple profile we additionally open another tab in the Google Chrome which is running a high definition (720p) video in a loop (see Figure 8).

Data sanitization. Our test setup was unstable with frequently freezing machines (e.g., networkcard stopped working and power outages). We repeatedly ran the same set-up; hence, we expect the measurement-chunks generated from the same machine to be fairly consistent. While we kept significantly represented outliers, we also measured 150 widely scattered outliers in 3 million measurements. These outliers are too few to be representatives of real effects. However, such widely scattered outliers distort our timing leakage analysis, since in theory real outlier effects that only happen in one configuration and not in the other configuration heavily amplify privacy leakage under continual observation.

To get a representative model of the underlying response-delay distributions from the measurements, we removed the above-mentioned unrepresentative, scarcely scattered outliers. To minimize the bias to the model, however, we dismissed entire batches of 6h measurements-blocks if they contained clear outliers w.r.t. to the rest of the (sub-)histograms for the same scenario, e.g., periodic active participants. Removing all such 6h blocks led to dismissing 20% of all measurements, leaving us with 3 million measurements in total.

5.2 Adding noise

To simplify and accelerate testing, our experiments send requests at fixed intervals, while the actual CoverUp implementation includes random delays. These delays are chosen from a Gaussian distribution $\mathcal{N}_{[0,2\mu]}(\mu,\sigma)$ with mean μ and standard-deviation[7] $\sigma = \frac{2}{10}\mu$, restricted to the interval $[0,2\mu]$, and add this delay to the minimum delay of one second. The expected delay is therefore $\mathbb{E}\left[1+\mathcal{N}_{[0,2\mu]}(\mu=\mu,\sigma=\frac{2}{10}\mu)\right] = 1+\mu$. We added noise artificially after the measurements by convolving the resulting measurement-histograms with a gaussian distribution.[8]

Adding the noise after the experiments is reasonable for our purpose. Recall that we use the measurement-experiments to produce a model for the underlying process, which we use to estimate the timing leakage under long-term attacker-observation. We evaluated the effect of separately adding the noise (see Appendix B) and found that does not significantly distorts our model.

5.3 Estimating the advantage

Our goal is to give an upper bound on the advantage for the task of distinguishing active and passive participants. This section explains the estimators that we use. We assume that the dominant part of the timing leakage will be visible from two kinds of measurements: *Loading* and *Periodic* measurements, as depicted by the orange arrows in Figure 4.

To quantify the timing leakage we use a quantitative variant of statistical indistinguishability of two distributions. For a pair of distributions X,Y and a random sample either from X or from Y, statistical indistinguishability requires that no attacker can tell whether the sample was chosen from X or from Y with more than an advantage δ, which can be represented as follows: $\delta(X,Y) := \frac{1}{2}\sum_{a\in\Omega}(|p_X(a)-p_Y(a)|)$.[9]

In other words, we provide a bound on a distinguisher's advantage. Specifically, n continual collected observations can be modeled by considering $\delta_{n,\{X,Y\}} := \delta(X^n,Y^n)$ for the product distribution X^n and Y^n. Another way of looking at the definition is that it requires the advantage of the interaction with active versus with passive participants to be bounded by a number close to 0.

Recall that the advantage quantifies an attacker's success in distinguishing active from passive participants after n observations, while having perfect knowledge of underlying response-delay distributions of the active and passive participants of type $type \in \{loading, periodic\}$. Therefore, we use $\delta_{n,type}$ as an estimator for the atacker's advantage.

Our analysis relies on three assumptions. First, all the measurement samples are independent. Second, Loading and Periodic measurements are independent. Third, the measured distributions represent the accurate underlying distributions. We believe that the first two assumptions hold in a deployed system because we assume a very high waiting time between requests (around 60s). The third assumption is of theoretical nature. While we conducted extensive measurements (around 3 million measurements in total) to render the model more representative, such measurements can only result in an approximation of the underlying processes. Using standard composition results (see Appendix Lemma 1), these assumptions enable us to bound the advantage of CoverUp with $\texttt{total}_{n,m} := \delta_{n,loading} + \delta_{m,periodic}$, after attacker that makes n Loading observations and m Periodic observations for either Linux or Windows. Here we explicitly use the assumptions that Periodic and Loading measurements are independent.

Moreover, under these assumption we use the Privacy-Buckets-tool [58][10] to compute $\delta_{n,loading}$ and $\delta_{n,periodic}$ from $\delta_{1,loading}$ and $\delta_{1,periodic}$ (for Linux and Windows, respectively), which we get from the sanitized measurement-histograms. Since the Ratiobuckets-tool is most precise when $n = 2^x$ for some x, the number of compositions that we use are powers of 2.

5.4 Timing leakage results

This section plots the results of our timing leakage estimation. For our evaluation, we overapproximate the connection pattern to the entry server by at most 50 site-loads and at most 5 hours of left open tabs (in the background) of visited entry servers per day. We consider an attacker that is able to continuously collect such data for half a year, i.e., 7 days a week for 26 weeks. We assume that the usage pattern of an active participant is identical to the visiting behavior of passive participants (see Section 5.5 for a deeper discussion on visiting behavior). We would like to stress that the same analysis applies to a continuous observation over 2.5 years for users that only make 10 site-loads at the entry server per

[7]There is no specific reason for this σ, but we wanted to prevent hard noise-distribution cut-offs as they increase δ_n.

[8]If we view the histogram as the probability mass function (pmf) for the timing delays, convolving this pmf with a gaussian distribution the histogram corresponds to addition of the corresponding random variables, i.e., adding the noise within the experiment.

[9]This advantage is also known as *total variation* or *statistical distance* and is connected to the classification-accuracy: $\texttt{acc} = (\delta/2)+0.5$.

[10]A publicly available numerical tool that computes a provable upper bound for the advantage under continual observation of a given pair of discrete distributions.

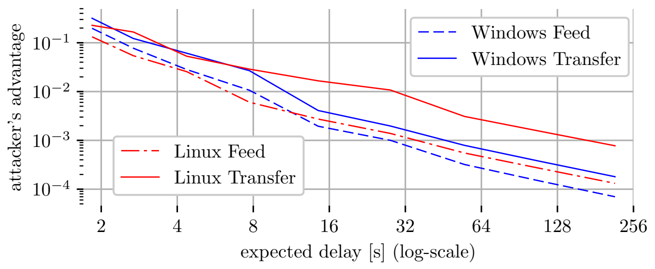

Figure 6: Latency versus upper bound on advantage for observation of half a year, with at most 5 hours of visiting the entry server (Periodic-observations) and at most 50 connecting to the entry server (Loading-observations) per day.

day and are connected for at most 1 hour per day to the entry server.

Latency vs timing leakage. Fixing the observation time to half a year and the connection pattern as described above, Figure 6 plots how $\texttt{total}_{n,m}$ increases with decreasing delays. Looking at the graph, we recommend 60s expected delay as system parameters to achieve an overall advantage of less than $2 \cdot 10^{-3}$ after 6 months of continual measurements of the user's timing patterns with daily 50 Loading observations and daily 5 hours worth of Periodic observations. We stress that despite the limits of our evaluation, the bounds that we present are highly over-approximated: we assume a global network-level attacker that has very precise information about the state of the system such as which processes are running and how they influence the measurements.

Observation-length vs timing leakage. The next angle is the length of the observation versus the degree of privacy: Figure 7. We fix the expected latency to 60s and plot for an increasing number of observations the functions $\delta_{n,loading}$ and $\delta_{m,periodic}$. This graph lets us study different usage behaviors. E-mail service, such as Google mail or Hotmail, as an entry server, e.g., would lead to significantly longer sessions than e-commerce entry servers. E-mail services would, hence, lead to less Loading observations and more Periodic observations. This graph shows that the leakage grows at most linearly with the number of observations. While Loading needs more time in Linux for the CU:Transfer (presumably because it invokes the extension each time), it produces less Periodic leakage while running. The graphs show that in many cases the Feed produces more timing leakage than Transfer. We believe that this is not true and an artifact of the Ratiobuckets-tool. The initial leakage for all Feed histograms is lower than their Transfer counterparts. The Ratiobuckets-tool also produces lower bounds (omitted here for readability) and those show that the Transfer upper bounds are much tighter than the Feed upper bounds.

Distorting effects of concurrent activities. The experiments of which we saw the results so far do not let any other program run in the background. In contrast, Figure 8 overlays the histogram of the vanilla experiments (without

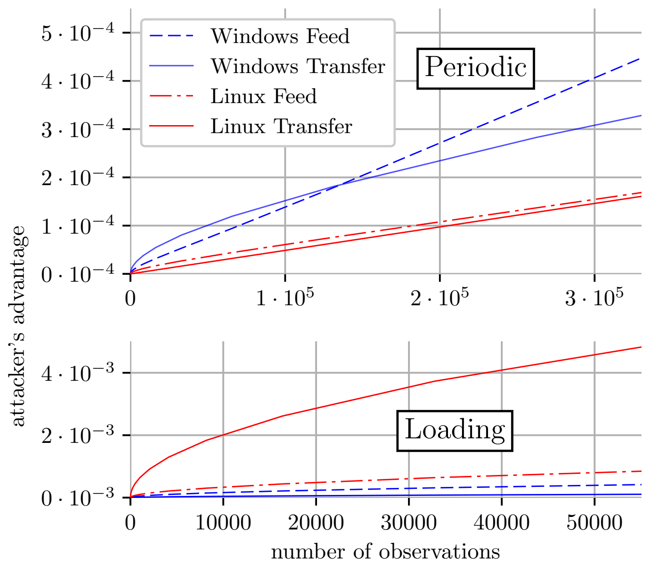

Figure 7: Number of observations versus upper bound on the advantage for Periodic and Loading leakage, evolving over numerous observations, with a 60s expected delay. The right end of the x-axes correspond to 3 years of observation.

any other programs running in the background) and experiments where the browser is rendering a 720p video on Linux. The experiments are conducted with Loading observations, as those produce more leakage. We can clearly see that rendering the video has some impact on the measurement (red line vs. blue line in Figure 8). Hence, it will be hard for an attacker to get such clean measurements like those that we use in our evaluation. This is another reason why we have some confidence that our privacy bounds give a good impression of the degree of privacy that CoverUp can offer, and maybe even provide a significant over-approximation.

5.5 Limits of our evaluation

This section discusses the limits of our evaluation of the timing leakage. While we do not claim that our evaluation offers provable bounds for the timing leakage of CoverUp, we believe that it captures the dominant part of the leakage of CoverUp and is a good indicator of the privacy that CoverUp offers.

Pairs of requests. We stick to pairs of requests since the autocorrelation is low and exploring all possible combinations for a higher number of contiguous requests increases the number of required measurements exponentially. To reduce potential effects from longer sequences of contiguous requests, we incorporate into our recommended delays a minimum of 1s between pairs of requests.

Unnoised measurements. We accelerated our measurements by not adding any additional noise, as we want to evaluate CoverUp with different amounts of noise. During the

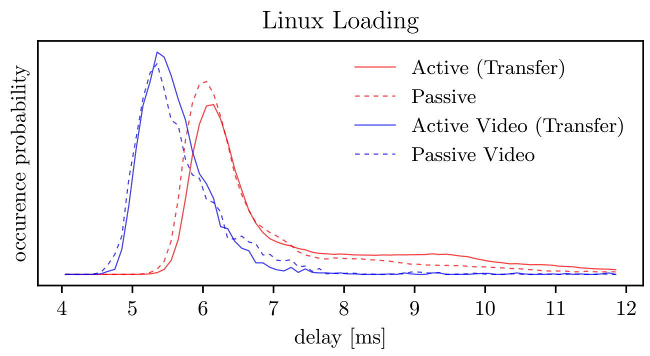

Figure 8: Different computation loads lead to different timing distributions. In the blue video plots, Google Chrome additionally renders a high definition (720*p*) video in a separate tab. Loading measurement. No randomly chosen delays added.

analysis phase, we introduce noise by computing the convolution of the resulting histograms with ideal Gaussian noise. To justify this we additionally construct an experiment with two scenarios: one with added artificial noise and another without where we add the artificial noise after the samples are collected. Figure 9 in Appendix B shows the timing distributions with total variation 1.8%.

Browser profiling. Evaluating our method against profiling attacks that are designed to detect whether a particular extension or a specific application is running [38] are out of scope of this work.

Neglecting the sampling error. Our experiments are limited to 3 million measurements. Hence, the histograms that we analyze do not exactly represent the underlying distribution. As our timing-leakage-bounds are computed on the histograms, they are not hard bounds but rather bounds that hold with high confidence.

6 Ethical & legal considerations

"Passive" participation has to be carefully implemented to avoid ethical and legal issues. We address potential ethical and legal considerations that stem from triggering visitors of some webpage into passively participating in a system like CoverUp. Our work received formal approval of our Institutional Review Board (IRB).

Even consenting passive participants that have been informed can experience unexpected consequences, e.g., by misunderstanding the consequences or by accidental consent. We are aware of the difficulties of informing website visitors in a way that they do not ignore the message and understand the consequences of consenting. Prior research [59, 33] suggests that one can design the CoverUp-dialog such that it minimizes the risks of misunderstanding and of agreeing by accident, e.g. deny by default and consenting in two phases, and highlight the network/battery-activities. Moreover, concepts like nudging and soft paternalism have the potential to let users act for benefits for themselves and for others [29].

Are computation and bandwidth resources of passive participants unwittingly utilized? No, only after an informed consent does CoverUp turn an entry server visitor to a passive participant.

Is CoverUp *harmful to passive participants?* No, CoverUp utilizes standard browser functionality.

Does CoverUp *store potentially incriminating data on the machine of passive participants?* No, we carefully incorporated an All-or-Nothing scheme such that passive participants never contain any useful information on their machine, as long as they do not actively extract and collect the CoverUp data packets from the browser's local storage.

Does CoverUp *trigger passive participants to open potentially suspicious connections?* After an informed consent, CoverUp does trigger a connection to the Feed/Transfer server, which some parties (e.g., an employer) could indeed view as suspicious.We do not consider this an ethical issue since this connection is only opened after its nature was stated and an informed consent was received.

Does the CoverUp *server collect information about the browsing behavior of the entry server's visitors?* No, while each iframe request of every entry server's visitor includes the visitor's IP address, the CoverUp server does not collect or store this information in any form.

Legal considerations for passive participants. Due to the AON scheme (see above), potentially illegal information distributed via the feeds never reaches a passive participant's machine. Additionally, as CoverUp is not primarily designed for the purpose of committing a cybercrime offense, receiving the CoverUp JS snippet is not an offense.

Legal considerations for the entry server. As the JavaScript code is provided by a third party, the entry server has no knowledge about the content, under EU and US legislation and case law the provider's liability privilege should apply. As a result, the entry server should not be held liable for the JavaScript code and thus the content of the feed.

Appendix C provides a more differentiated and thorough discussion of the legal topics.

7 Related work

Extending the anonymity set via JavaScript. There are previous research works on utilizing visitors of a collaborating website to produce anonymizing cover traffic via JavaScript. Conscript [41] and Adleaks [62] describes upload only uni-directional channel from the users to the mix network. In contrast, CoverUp provides a transport private bi-directional channel. Conscript mentioned timing leakage based side channel attacks but evaluation details are missing except power consumption. Conscript additionally has deployment hurdles, since it trusts the entry server to achieve code integrity. While previous work suggests mitigating this trust assumption by letting the extension check all dynamic content to achieve code integrity against a malicious entry server, such dynamic checks will tremendously increase the timing leakage, and thus rendering the active participants clearly distinguishable from passive ones. The need to trust the entry server gives the entry server more responsibility and requires a careful evaluation of the entry servers. The implementation of Adleaks requires a patched version of the browser. This reduces the set of possible browsers and therefore reduces the anonymity set massively. Detailed privacy analysis is not described in the paper including timing leakages. The paper [32] describes how to include unwilling users to cover server to server communication. All transport between the servers (by passive clients) is not encrypted. This means an inspection of the HTTP body reveals intention. Moreover, the paper lacks any implementation details. Additionally, previous works lack a legal aspects discussion of "passive" participation.

Anonymous uploads and downloads. While COVERUP at its core provides a bi-directional transport channel on which ACNs could run, COVERUP has distinctly other goals than traditional ACNs or systems like Pung [30]: COVERUP's goal is to enable users to hide their traffic in the traffic of normal web surfers, i.e., to extend the potential anonymity set to normal web surfers.

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 [52, 51, 31]. But once detected, the origin, thus the intention, is obvious. The same applies to Mixing [55]. Off-the-record messaging publishes the MAC key after each talk, rendering it vulnerable against real-time monitoring [35].

McPherson et al. proposed CovertCast, a broadcast hidden in normal video streams like YouTube [57]. Che et al. were able to create a deniable communication channel based on different levels of noisy channels [39]. 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 [49] provides a covet channel where the user can modulate his internet traffic signal into acoustic data and transfer it to a remote server via VoIPs such as Skype. Such system has bandwidth limitation and is vulnerable to attacks described in [48]. SWEET [50] 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 inherently very low bandwidth and high latency, making them practically infeasible for deployment. CloudTransport [37] introduced covert communication which involves publicly accessible cloud servers such as Amazon S3 which acts as the oblivious mix. However, such services do not provide protection against attackers learning intention. Infranet [45] describes a system executing covert communication using image stenography, but it also suffers from inherently low bandwidth.

Censorship circumvention. There exist several censorship circumvention tools that allow users to reach websites which are otherwise unreachable due to local policies. Flash Proxies [46] provides a browser-based proxy that connects to a tor bridge. Its implementation uses WebSocket and JavaScript to create many, generally ephemeral bridge IP addresses, effectively surpassing the censor's ability to block them. It is now outdated and replaced by Snowflake [27] which is a Tor pluggable transport [22] with a design principle identical to Flash Proxies. Other pluggable transports such as Tor's meek [20] relay data through a third-party server that is hard to block, for example a CDN, using a mechanism called domain fronting [47]. COVERUP is orthogonal to the aforementioned papers. COVERUP does not provide any form of censorship circumvention, as the censor can disable COVERUP by blocking all the web requests to either the entry server, the COVERUP server, or the feed/Transfer server.

8 Conclusion

We discussed how the concept of passive participation can improve the privacy of accessing information in an anonymous and deniable manner. By drawing in passive participants to create cover traffic, we achieve participation deniability: an attacker cannot tell whether an observed request to a Feed/Transfer server originates from a active participant which is interested in its content, or from a passive participant which is only surfing on the entry server.

We leverage this concept with COVERUP, which can operate in two modes: CU:Feed, distributing an uni-directional broadcast, and CU:Transfer, providing a deniable up- and download channel. Given our implementation, we experimentally evaluated the degree of privacy COVERUP can guarantee. For both, CU:Transfer and CU:Feed, we found that the timing leakage is acceptable (an advantage under $2 \cdot 10^{-3}$) within a half a year of continual observation. Even for a state-level agency a half a year of continual observation (on sub-ms-level granularity) incurs a significant cost.

This work introduces a practical solution for deniable upload and download, and deniable feed-receivers. The present analysis clearly shows that the passive-participation-approach can provide sufficient cover for highly anonymity-aware users. We conclude that the line of research on passive participation is a promising direction for deniable communication.

References

[1] 17 u.s. code para. 512. https://www.law.cornell.edu/uscode/text/17/512.

[2] America's founding documents — national archives. https://www.archives.gov/founding-docs.

[3] Chart of signatures and ratifications of treaty 185. http://tinyurl.com/h8ketgj.

[4] Chome scripts - google chrome. https://developer.chrome.com/extensions/content_scripts#execution-environment.

[5] Consolidated version of the treaty on the functioning of the european union. http://eur-lex.europa.eu/resource.html?uri=cellar:41f89a28-1fc6-4c92-b1c8-03327d1b1ecc.0007.02/DOC_1&format=PDF.

[6] Content security policy (csp) - google chrome. https://developer.chrome.com/extensions/contentSecurityPolicy.

[7] Convention on cybercrime, budapest, 23.xi.2001. http://www.europarl.europa.eu/meetdocs/2014_2019/documents/libe/dv/7_conv_budapest_/7_conv_budapest_en.pdf.

[8] Cross origin xmlhttprequest - google chrome. https://developer.chrome.com/extensions/xhr.

[9] Directive 2000/31/EC of the European Parliament and of the Council of 8 June 2000 on certain legal aspects of information society services, in particular electronic commerce, in the Internal Market ('Directive on electronic commerce'), 2000 O.J. L 178.

[10] Directive 2002/22/ec of the european parliament and of the council. http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32002L0022&from=EN.

[11] Directive 2002/58/ec of the european parliament and of the council. http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:32002L0058:en:PDF.

[12] Directive 2009/136/ec of the european parliament and of the council. http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2009:337:0011:0036:en:PDF.

[13] Directive 95/46/ec of the european parliament and of the council. http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:31995L0046&from=EN.

[14] Eur lex. http://eur-lex.europa.eu/legal-content/EN/ALL/?uri=OJ%3AC%3A2012%3A326%3ATOC.

[15] European convention on human rights (ehcr). http://www.echr.coe.int/Documents/Convention_ENG.pdf.

[16] Federal constitution of the swiss confederation. https://www.admin.ch/opc/en/classified-compilation/19995395/index.html.

[17] Fourth amendment. https://www.law.cornell.edu/constitution/fourth_amendment.

[18] Katz v. united states, 389 u.s. 347 (1967). https://supreme.justia.com/cases/federal/us/389/347/case.html.

[19] Manifest: Web accessable resources - google chrome. https://developer.chrome.com/extensions/manifest/web_accessible_resources.

[20] meek: Tor bug tracker and wiki.

[21] Olmstead v. united states, 277 u.s. 438 (1928). https://supreme.justia.com/cases/federal/us/277/438/case.html.

[22] Pluggable transports.

[23] Regulation (ec) no 2006/2004 of the european parliament and of the council. http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32004R2006&from=EN.

[24] Regulation (ec) no 2006/679 of the european parliament and of the council. http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679&from=en.

[25] Rfc 7323 - tcp extensions for high performance. https://tools.ietf.org/html/rfc7323.

[26] Rfc 918 - post office protocol. https://tools.ietf.org/html/rfc918.

[27] Snowflake.

[28] tshark-the wireshark network analyzer 2.0.0. https://www.wireshark.org/docs/man-pages/tshark.html.

[29] ACQUISTI, A. Nudging privacy: The behavioral economics of personal information. *IEEE Security Privacy* (2009).

[30] ANGEL, S., AND SETTY, S. T. Unobservable communication over fully untrusted infrastructure.

[31] ARTZ, D. Digital steganography: hiding data within data. *IEEE Internet Computing* (2001).

[32] BAUER, M. New covert channels in http: Adding unwitting web browsers to anonymity sets. In *Proceedings of the 2003 ACM Workshop on Privacy in the Electronic Society* (New York, NY, USA, 2003), WPES '03, ACM, pp. 72–78.

[33] BEAUDOUIN-LAFON, M. Designing interaction, not interfaces. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '04.

[34] BERNSTEIN, D. J. Curve25519: New diffie-hellman speed records. In *Public Key Cryptography - PKC 2006* (Berlin, Heidelberg, 2006), M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, Eds., Springer Berlin Heidelberg, pp. 207–228.

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

[36] BOYKO, V. *On the Security Properties of OAEP as an All-or-Nothing Transform.*

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

[38] CAO, Y., LI, S., AND WIJMANS, E. (Cross-)Browser Fingerprinting via OS and Hardware Level Features. In *NDSS 2017.*

[39] 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.*

[40] CORRIGAN-GIBBS, H., BONEH, D., AND MAZIÈRES, D. Riposte: An anonymous messaging system handling millions of users. In *S&P 2015.*

[41] CORRIGAN-GIBBS, H., AND FORD, B. Conscript Your Friends into Larger Anonymity Sets with JavaScript. In *WPES 2013.*

[42] DAEMEN, J., AND RIJMEN, V. *The design of Rijndael: AES-the advanced encryption standard.* 2013.

[43] DANEZIS, G., AND SERJANTOV, A. Statistical disclosure or intersection attacks on anonymity systems. In *Information Hiding*.

[44] DANEZIS, G., AND SERJANTOV, A. Statistical disclosure or intersection attacks on anonymity systems. In *Information Hiding*, J. Fridrich, Ed.

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

[46] FIFIELD, D., HARDISON, N., ELLITHORPE, J., STARK, E., BONEH, D., DINGLEDINE, R., AND PORRAS, P. Evading censorship with browser-based proxies. In *Privacy Enhancing Technologies* (2012), Springer Berlin Heidelberg.

[47] FIFIELD, D., LAN, C., HYNES, R., WEGMANN, P., AND PAXSON, V. Blocking-resistant communication through domain fronting. *Proceedings on Privacy Enhancing Technologies* (2015).

[48] GEDDES, J., SCHUCHARD, M., AND HOPPER, N. Cover your acks: Pitfalls of covert channel censorship circumvention. In *PCCS 2013*.

[49] 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).

[50] HOUMANSADR, A., ZHOU, W., CAESAR, M., AND BORISOV, N. Sweet: Serving the web by exploiting email tunnels. *IEEE/ACM Trans. Netw.*.

[51] JOACHIM J. EGGERS, ROBERT BAEUML, B. G. Communications approach to image steganography.

[52] 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* (2013).

[53] LABAJ, M., AND BIELIKOVÁ, M. Tabbed Browsing Behavior as a Source for User Modeling. In *User Modeling, Adaptation, and Personalization* (2013).

[54] LANDAU, S. Making sense from snowden: What's significant in the nsa surveillance revelations. *IEEE Security Privacy* (2013).

[55] 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).

[56] MACKAY, D. Fountain codes. *IEE Proceedings - Communications*.

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

[58] MEISER, S., AND MOHAMMADI, E. Privacy buckets: Upper and lower bounds for k-fold tight approximate differential privacy. Cryptology ePrint Archive, Report 2017/1034.

[59] PATRICK, A. S., AND KENNY, S. From privacy legislation to interface design: Implementing information privacy in human-computer interactions. In *Privacy Enhancing Technologies* (2003), R. Dingledine, Ed.

[60] PIOTROWSKA, A. M., HAYES, J., ELAHI, T., MEISER, S., AND DANEZIS, G. The loopix anonymity system. In *26th USENIX Security Symposium, USENIX Security* (2017), pp. 16–18.

[61] RIVEST, R. L. All-or-nothing encryption and the package transform. In *Fast Software Encryption* (Berlin, Heidelberg, 1997), E. Biham, Ed., Springer Berlin Heidelberg.

[62] ROTH, V., GÜLDENRING, B., RIEFFEL, E., DIETRICH, S., AND RIES, L. A Secure Submission System for Online Whistleblowing Platforms. In *FC 2013*.

[63] SHOKROLLAHI, A. Raptor codes. *IEEE transactions on information theory*.

[64] STEFANOV, E., VAN DIJK, M., SHI, E., FLETCHER, C. W., REN, L., YU, X., AND DEVADAS, S. Path ORAM: an extremely simple oblivious RAM protocol. In *CCS 2013* (2013), pp. 299–310.

[65] SUNDARARAJAN, J. K., SHAH, D., AND MÉDARD, M. Arq for network coding. In *ISIT 2008*.

[66] 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).

[67] WEBER, R. *E-Commerce und Recht, 2. Auflage.* 2010.

[68] WOLINSKY, D. I., CORRIGAN-GIBBS, H., FORD, B., AND JOHNSON, A. Dissent in numbers: Making strong anonymity scale. In *OSDI* (2012), pp. 179–182.

A Constructions

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

A.1 Fountain Code

Fountain codes [56, 65] 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* [63] in the literature but most of them are protected by intellectual property rights.

A.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 un-keyed, invertible, randomized transformation, with the property that it is hard to invert unless all of the output is known."[36].

We modified the *all-or-nothing scheme* proposed by Rivest [61] which encrypts all data with a symmetric key cryptography algorithm (in our implementation, we use `AES-128` [42]) 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 $\mathscr{K} \overset{R}{\leftarrow} \{0,1\}^{128}$ 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.

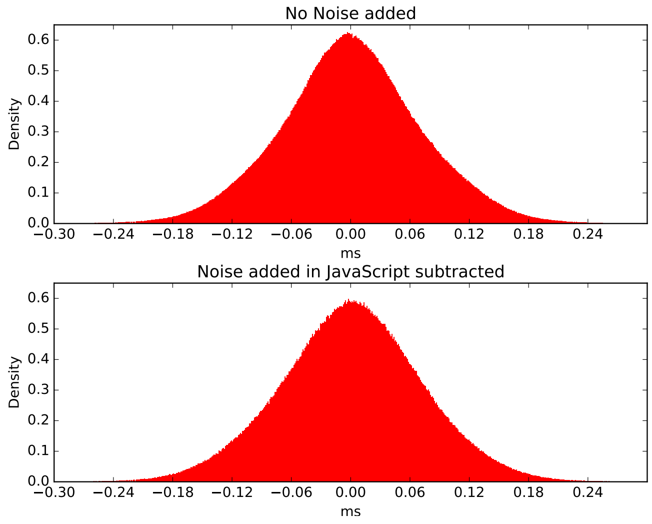

Figure 9: Statistical Independence using uniform noise: Distance: 1.8%

- Let $m'_{\text{key}} = \mathscr{K} \oplus h_1 \oplus h_2 \oplus \ldots \oplus h_n$ where $h_i = \mathscr{H}_i[1,\ldots,128]; \mathscr{H}_i = \texttt{SHA-256}(m_i)\ \forall i \in 1,\ldots,n$
- 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

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

B Independence of additional noise

Recall that 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 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 9 shows the resulting distribution. The statistical distance of these two distributions is 1.8% which is an acceptable value.

C 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 [7] remains limited by territory because sovereign states put their own legislation into effect [5, 14, 2]. 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.

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 [3] in the Convention on Cybercrime [7] (CCC) – the international treaty on crimes committed via the Internet and other computer networks. This international treaty criminalizes, among others, illegal access (Art. 2 CCC), data interference (Art. 4 CCC), and misuse of devices (Art. 6 CCC).

C.1 Passive participants

Illegal access. Illegal access (Art. 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 [12, 10, 11, 23] by client). However, a broad interpretation of Art. 2 CCC is not undisputed (refer [7], §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 packet. 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 Art. 2 CCC, the provider of the entry server should not be illegally accessing the browser.

Data interference. Data interference (Art. 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 should in principle be considered legitimate protection of privacy (refer [15, 17, 21, 18], [13, Recitals(1) and (35)]), [16, Art. 13], and, therefore, be considered as being undertaken "with right" [7, §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 packets from the ACN to the passive 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 concluded above, the JavaScript code utililized 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 Art. 4 should not apply to the provision of the webpage with the iframe by the provider of the entry server.

Misuse of devices. Misuse of devices (Art. 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) [3, §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 Art. 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? Art. 6 CCC restricts the scope to cases where the code is objectively designed primarily for the purpose of committing an offense, usually excluding dual-use devices [3, §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 [13, Art. 2(a)], [24, Art. 4(1)]. If a person is identifiable or identified, data protection law applies. However, if the personal data are pseudonymized or anonymized, then data protection law might not apply anymore because the (formerly identifiable or identified) person cannot longer be identified.

Recital (83), Art. 6(4)(e), 32(1)(a) and 34(3)(a) of the new General Data Protection Regulation[11] 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 ACN in the packets are not information because the data is unintelligible. Not only does the passive participant not have sufficient data to reassemble the packet 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, with regard to the passive 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. Moreover, the JavaScript code, with regard to the active participant, can be qualified as dual use device because the encrypted and unintelligible data are decrypted and reassembled to intelligible information.

Legal conclusion. We discussed the applicability of Art. 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 passive participant does not qualify as misuse of device. In regard to the reassembly of the packets to a meaningful whole, if the information is illegal, CoverUp might qualify as dual use device and fall under Art. 6 CCC. We conclude that at least with regard to the risk of indictment pursuant to Art. 6 CCC it seems advisable that the provider of the entry server does not provide the JavaScript code for download.

C.2 Entry servers

A participant is dependent on Internet service providers (ISP). The question arises whether an (ISP) should be liable for illegal Internet activities of its subscribers. In the following we discuss legislation and case law on the ISP's liability in two different jurisdictions: the EU and the USA.For this discussion it is important to differentiate among the various types of ISPs, for instance access providers, hosting providers, and content providers [67].

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

European union. In the European Union, liability of ISPs has been regulated in the E-Commerce Directive [9]. Generally, providers shall not have any obligation to monitor the information which they transmit or store, or to seek actively facts or circumstances indicating illegal activity [9, Art. 15 (1)]. According to the directive, access providers acting as "mere conduits" shall not be liable for the information transmitted, on the condition that they do not initiate, select the receiver of, or select or modify the information contained in the transmission [9, Art. 12 (1)].[12] Caching providers (efficiency transmitters) shall not be liable for the automatic, intermediate and temporary storage of information, on the condition that they do not modify the information; comply with access regulations and industry standards for updating the information; do not interfere with the lawful use of technology; and act expeditiously to remove information if removed from the initial source [9, Art. 13 (1)]. Hosting providers shall not be liable for the information stored on their servers, on the condition that they are unaware of illegal activity or information or acts expeditiously to remove or disable access to the illegal information [9, Art. 14 (1)].

With regard to the obligations of a hosting provider, the European Court of Justice decided in SABAM v Netlog[13] that, among other directives, the E-Commerce Directive precluded a national court from issuing an injunction against a hosting service provider which requires it to install a system for filtering (a) information which is stored on its servers by its service users, (b) which applies indiscriminately to all of those users; (c) as a preventative measure; (d) exclusively at its expense; and (e) for an unlimited period; which is capable of identifying IP-infringing content.

USA. Similarly, in the United States there are limitations on liability relating to material online [1]. There are statutory limitations for transitory communications (i.e. access provider, "mere conduit") [1, Section 512(a)], system caching (i.e. storage for limited time) [1, Section 512(b)], information residing on systems or networks at the direction of users (i.e. hosting) [1, Section 512(c)], and information location tools (i.e. search engines or hyperlinking) [1, Section 512(d)].

With regard to the obligations of a hosting provider [1, Section 512(c)], the United States Court of Appeals for the Second Circuit, by referencing UMG Recordings, Inc. v. Shelter Capital Partners LLC, 667 F.3d 1022 (9th Cir. 2011), argued that "*[t]he Court of Appeals affirmed [...] that the website operator was entitled to safe harbor protection. With respect to the actual knowledge provision, the panel declined to 'adopt [...] a broad conception of the knowledge requirement,' id. at 1038, holding instead that the safe harbor '[r]equir[es] specific knowledge of particular infringing activity,' id. at 1037. The Court of Appeals reach[ed] the same conclusion' [..] noting that [w]e do not place the burden of determining whether [materials] are actually illegal on a service provider.' Id. At 1038 (alterations in original) (quoting Perfect 10, Inc. v. CCBill LLC, 488 F.3d 1102, 1114 (9th Cir. 2007))*". Hence, the 2nd Circuit Court concluded, among others, that 17 U.S.C. §512(c)(1)(A) requires knowledge or awareness of facts or circumstances that indicate specific and identifiable instances of infringement.

Legal conclusion. The entry server is probably not an access provider, maybe a caching provider and presumably a hosting provider. In the latter case two points seem relevant: (i) by whom the information is stored on the entry server and (ii) the entry server's knowledge of any (illegal) activity.

First, depending on how the entry server's webpage is set up, the JavaScript code may be stored by the entry server itself or by a third party. Only in the latter case does the provider's liability privilege apply, because if the JavaScript code is stored on the entry server by the entry server itself, then it is neither an access, nor a caching nor a hosting provider, but probably a content provider (assuming that the JavaScript code is qualified as content). The ISP liability privilege does not apply to content providers. Second, if the JavaScript code is stored by the entry server itself on the entry server, then the entry server obviously has knowledge of the content. The ISP liability privilege should not apply. If the JavaScript code is uploaded by a third party (as done in CoverUp) to the entry server, and the entry server therefore has no knowledge about the content, then under EU and US legislation and case law the entry server should not be held liable for the JavaScript code.

D Estimator-assumptions

Definition 1 (Total variation over finite domain)**.** *Let X, Y be two discrete distributions over a finite domain with a joint domain Ω. Then, the* total variation *d of X and Y is $d(X,Y) := \frac{1}{2}\sum_{a\in\Omega}(|p_X(a) - p_Y(a)|)$.*

Lemma 1. *Let X_l, X_p be the Loading, respectively the Periodic, measurement distribution of the passive user and Y_l, Y_p the Loading respectively the Periodic measurement distribution of the active user, all with a joint Domain Ω. Let further be δ_l be the total variation between X_l and let Y_l and δ_p be the total variation between X_p and Y_p. Then, for all Turing machines A, if all the measurement samples are independent (**AI**), Loading and Periodic measurements are independent (**AII**), and the measured distributions represent the accurate underlying distributions (**AIII**),*

$$\begin{aligned}&|\Pr[b=1 : b \leftarrow A(w_l,w_p), w_l \leftarrow X_l^n, w_p \leftarrow X_p^m]\\ &-\Pr[b=1 : b \leftarrow A(w_l,w_p), w_l \leftarrow Y_l^n, w_p \leftarrow Y_p^m]| \leq n\delta_l + m\delta_p\end{aligned}$$

Proof. Let $w \overset{n}{\leftarrow} X$ denote n independent draws from a distribution X. Let $\Pr[w \leftarrow X] = \Pr[b=1 : b \leftarrow A(w), w \leftarrow X]$ and $\Pr[w_l \leftarrow X_l \bowtie w_p \leftarrow X_p] = |\Pr[b=1 : b \leftarrow A(w_l,w_p), w_l \leftarrow$

[12] With regard to the German liability for interference ("Störerhaftung") according to Sommer unseres Lebens (I ZR 121/08), see also decision by the ECJ in Mc Fadden (C- 484/14).

[13] ECJ C-360/10.

$X_l, w_p \leftarrow X_p]$. We conclude:

$$\begin{aligned}
&|\Pr[b=1 : b \leftarrow A(w_l, w_p), w_l \leftarrow X_l^n, w_p \leftarrow X_p^m] \\
&- \Pr[b=1 : b \leftarrow A(w_l, w_p), w_l \leftarrow Y_l^n, w_p \leftarrow Y_p^m]| \\
= &|\Pr[w_l \overset{n}{\leftarrow} X_l \bowtie w_p \overset{m}{\leftarrow} X_p] - \Pr[w_l \overset{n}{\leftarrow} Y_l \bowtie w_p \overset{m}{\leftarrow} Y_p]| \\
\overset{\text{AI}}{\leq} &|\Pr[w_l \overset{n}{\leftarrow} X_l \vee w_p \overset{m}{\leftarrow} X_p] - \Pr[w_l \overset{n}{\leftarrow} Y_l \vee w_p \overset{m}{\leftarrow} Y_p]| \\
\overset{\text{AII}}{\leq} &n \cdot |\Pr[w_l \overset{1}{\leftarrow} X_l] - \Pr[w_l \overset{1}{\leftarrow} Y_l]| \\
&+ m \cdot |\Pr[w_p \overset{1}{\leftarrow} X_p] - \Pr[w_p \overset{1}{\leftarrow} Y_p]| \overset{\text{AIII}}{\leq} n \cdot \delta_l + m \cdot \delta_p
\end{aligned}$$