Sensing Censorship and Censuring Censors with Censorship-Evident Publishing Systems

Swaminathan Ramesh
University of Calgary
swaminathan.ramesh@ucalgary.ca

Ryan Henry
University of Calgary
ryan.henry@ucalgary.ca

Abstract

Censorship has always existed, serving both to prevent harms and to inflict them by chilling speech, suppressing organizing, and withholding inconvenient facts and ideas; most technical work aims to prevent all forms of censorship—the "good", the "bad", and everything in between. We study the complementary, rarely explored goal of making any censorship attempt *transparent*. We formalize *censorship-evident publishing systems* (CEPS), protocols that force both overt and covert takedowns to yield transferable evidence. We also provide a CEPS instantiation with Streisand, a proof-of-concept deployment that combines a blockchain-backed timestamp oracle, private information retrieval (PIR)-based anonymous queries to prevent extraction attempts from being conspicuous, and probabilistic Merkle-witness retrieval to produce compact censorship proofs. We present performance evaluations on a 1.3 GiB Enron-derived dataset with regex-based PII redaction to model realistic censorship, and demonstrate that a background daemon can detect heavy censorship after a small number of post-censorship queries, making Streisand an effective auditing mechanism rather than interactive file retrieval. We also discuss design trade-offs (proof size vs. computation, PIR sufficiency vs. necessity), scalability limits, and how CEPS complements existing transparency practices, with Streisand as a starting point for CEPS deployments.

Keywords

Censorship resistance, accountability, private information retrieval

1 Introduction

Social media has emerged as a dynamic and interconnected digital landscape that has revolutionized how netizens communicate and share information. This vast domain encompasses various platforms, from industry behemoths like Facebook through microblogging services like Bluesky and 𝕏 (formerly known as Twitter), video- and audio-sharing apps like TikTok and Clubhouse. Additionally, it encompasses discussion hubs and news-sharing sites like Reddit and Lemmy. Despite the diversity of these platforms, a common thread that binds them is their focus on user-submitted content and community building, with users submitting content of various forms—be it text, images, videos, or arbitrary files—which is then made available to other users for consumption and engagement.[1]

The ideal underpinning of social media revolves around an ethos of extreme inclusivity, where all viewpoints are welcome, fostering a diverse exchange of ideas and perspectives. However, service providers often struggle to navigate a delicate balance between allowing unrestricted content on the one hand and adhering to legal obligations and general "good netizenship" on the other. For example, within various legal jurisdictions, service providers are compelled to prevent the proliferation of child sexual abuse material (CSAM) and non-consensual pornography ("revenge porn") on their platforms [15, 22, 45]. Meanwhile, many service providers voluntarily adopt terms of service that prohibit, e.g., hate speech, cyberbullying, or the deliberate dissemination of disinformation [37, 52]. The fulfillment of these obligations may necessitate social media platforms to exercise content oversight and, sometimes, censor specific user submissions.

This paper focuses on the tension between a commitment to the competing priorities of free and open communications with socio-legal responsibility. Indeed, predicated on the notion that *transparency* is necessary (though we make no claims of sufficiency!) for striking a meaningful balance between the two priorities, we investigate a class of primitives for cryptographically enforced transparency for all content censorship. To this end, we propose and formalize *censorship-evident publishing systems* (CEPS) and present an efficient CEPS construction called Streisand. The Streisand moniker is a nod to the "purpose" of CEPS, namely to disincentivize censorship using the so-called *Streisand effect* [48], a phenomenon wherein the act of censorship or suppression ultimately draws increased public interest and scrutiny, backfiring on the party seeking to suppress the information. Both the CEPS notion and the Streisand construction draw inspiration from Martiny, Miers and Wustrow's *Proofs of Censorship* [30].

Where CEPS plugs in: CEPS sits alongside a platform's existing moderation stack. Users upload content; if the publisher accepts it, the publisher issues and logs a ticket committing to host it. Users continue to consume content via the ordinary front-end. In the background, CEPS clients send anonymous probe and extract queries; if censorship is detected, a transferable proof is produced. CEPS does not force publication; it prevents silent reneging on prior commitments.

Why simple hashes alone are insufficient: One might hope to "fix" a ground truth by hashing the original content and relying on that hash thereafter. This fails in our setting: outside observers often cannot know which hash is correct when faced with conflicting claims, and embedding full content in tickets makes ticket sizes and ledger costs scale with content. CEPS instead focuses on evidence that can be efficiently and unambiguously demonstrated to third parties without assuming globally known ground truth—indeed, without even provers necessarily having access to ground truth.

[1] Other so-called "Web 2.0" properties, such as Wikipedia and its controversial counterpart Wikileaks, share social media's focus on user-generated content. For the purposes of this paper, we lump such websites in with "social media" despite this being a decidedly non-standard usage of the term.

1.1 Related work in censorship resistance

Khattak, Elahi, Simon, Swanson, Murdoch and Goldberg [25, §2] define *censorship resistance systems* as follows:

> *A* censorship resistance systems [sic] *(CRS) is meant for information exchange between a user and a publisher that are connected by a communication channel in the face of a censor who actively attempts to prevent this communication. A censor's goal is to disrupt information exchange over a target system, by attacking the information itself (through corruption, insertion of false information, deletion or modification), or by impairing access to, or publication of, information...*

Khattak et al.'s definition references entities taking on three distinct roles: publishers, users, and censors. In the context of this paper, the *publishers* are social media platforms that host content on behalf of—and subsequently make that content available to—their user base. In contrast, *users* are the entities that contribute content to and consume content from the publisher's platform. Finally, *censors* are entities that seek to prevent specific content from being accessible via the platform. Who plays the role of censor can vary with context; for example, a censor might be the publisher itself willfully removing content that violates its terms of service or a coercive entity (e.g., the courts) that compels the publisher to limit access to specific content.

Modern-day Internet censors operate through various means ranging from technological to coercive [16]. In response, the research community has developed a broad range of countermeasures on which to base censorship resistance systems.

Resisting technological censorship: Technological censorship typically occurs via some form of network disruption, such as DNS poisoning [2], BGP hijacking [17], IP filtering [32], or more sophisticated means based on deep-packet inspection [11]. Censorship resistance systems to counter such technological censorship employ technological solutions like encryption [12, 13], proxies or anonymity networks (like Tor [42] or I2P [39]), or decoy (aka. refraction) routing [21, 51] protocols that seek to limit censors' abilities to distinguish objectionable communication flows from innocuous ones, preventing targeted disruption.

Resisting coercive censorship: Coercive censorship, by contrast, relies on a censoring entity issuing legal or extralegal orders for publishers to take down objectionable content. Censorship resistance systems to counter such coercive censorship include censorship-resistant publishing systems based on redundancy [47] or counter-coercion [5] to make censorship so prohibitively costly as to render publishers impotent to comply with coercive takedown requests, alongside "darknet" routing protocols (such as Tor onion services [41] and I2P [36]) that shield publishers' identities and network locations from would-be coercers.

All, or nothing at all: All aforementioned censorship resistance approaches try to impose an all-or-nothing choice on censors: *Everything* remains accessible—even in cases where the censor and publisher (and perhaps most users) agree that a given takedown request is justified—unless the decision is made to also suppress *large swaths of innocuous content* as "collateral damage", with any economic and political consequences this entails. Despite its appeal to free-speech absolutists, the all-or-nothing approach to censorship resistance all but precludes buy-in from pragmatic publishers or lawmakers, who may be sympathetic to calls for free and open communications but also see a need to balance it with legal and ethical obligations. Therefore, we argue that exploring additional, less rigid forms of censorship resistance systems that can dissuade or limit the harms caused by coerced censorship or overzealous moderation is a prudent research direction—even if proposed solutions are destined to fall short of preventing all censorship harms.

Proofs of retrievability: Another line of related work on verifiable hosting, albeit not directly related to censorship resistance, is found in the literature on *proofs of retrievability (PoRs)* [6, 24]. PoRs are concerned with storage-as-a-service (STaaS) scenarios where a user outsources personal storage to a rational, profit-seeking entity that may—if it believes it can get away with it—simply delete the outsourced content to save on storage costs while continuing to accept payment from the user. To this end, PoR schemes enable the user to audit the content host by issuing challenges and verifying the responses. PoRs guarantee that if responses to these challenges pass verification with a *noticeable* probability, then the hosted data is efficiently retrievable with an *overwhelming* probability. However, the challenges in a typical PoR scheme do not attempt to hide which specific data is being audited, nor do they provide any mechanism for generating transferable proofs (both of which turn out to be a central requirement in the CEPS case). In contrast, CEPS tackles a related question, viewed through the lens of censorship resistance: How can users interacting with a publisher—who may subsequently be coerced into deleting or blocking access to specific content—verify that user submissions are not covertly censored, or, failing that, convincingly demonstrate to others that covert censorship has occurred? This notion of allowing all users to audit arbitrary submissions and generate transferable censorship proofs for arbitrary publisher-hosted content makes the CEPS notion fundamentally different from, and harder to achieve than, that of PoRs.

Proofs of Censorship: A direct predecessor to and inspiration for CEPS is the *Proofs of Censorship (PoCs)* work by Martiny, Miers, and Wustrow [30]. PoCs attempt to discern and produce transferable censorship proofs using a workflow centred around authenticated single-server private information retrieval (PIR) queries. The notion of CEPS presented herein both generalizes and extends the security guarantees of PoCs, while enabling deployment as a *censorship auditing mechanism* through the introduction of probabilistic *probe* and *extraction* queries that replace the simpler PIR-based query mechanism in Martiny et al.'s scheme. The Streisand system likewise inherits several design choices from PoCs.

Authenticated PIR and certificate transparency: There are also overlapping goals between CEPS and PIR-based certificate transparency (CT) mechanisms, including SimplePIR [20] and VeriSimplePIR [9], which share some common machinery with Streisand. Indeed, both CEPS and CT audit data from an untrusted source; however, the threat models diverge sharply. In CT, the adversary is a rogue certificate authority (CA) that is mis-issuing certificates for some server, and a benevolent CT log helps the clients detect such

mis-issuance; in CEPS, the CT-like "log", the server, and the adversary are all the *same* entity. CT seeks to detect equivocation over tall/skinny databases of hashes; CEPS must enable extraction of short-/fat *data* records, and/or produce *transferable* proofs. CT uses PIR to hide browsing habits (privacy); CEPS uses PIR (plus an anonymous channel) to keep extraction attempts indistinguishable from probes (security/transferability). CT returns full Merkle proofs on every query; CEPS only cares about rare Merkle *disproofs*, so Streisand uses probabilistic uncle-fetching to avoid a $\Theta(\log s)$ communication blow-up for s-word content. Client-preprocessing PIR used in CT is infeasible at CEPS data scales; alternative PIR/verification choices remain possible because CEPS does not mandate Merkle trees. CEPS is likewise related to public-key transparency systems (e.g., WhatsApp key transparency [28], CONIKS [31], and certificate logs) that rely on Merkle-tree consistency to audit distributed state. Again, the CEPS setting differs in threat model (publisher-as-adversary), goal (transferable proofs of censorship vs. mis-issuance), and data shape (short/fat content vs. tall/skinny key logs), but shares some auditing intuition.

Our contributions: We formalize the notion of censorship-evident publishing systems (CEPS) with constituent protocols and security guarantees necessary, and provide Streisand as a proof-of-concept instantiation to demonstrate feasibility of realizing CEPS guarantees in practice. We also discuss directions to pursue to achieve efficient CEPS instantiations in Section 5.

2 CEPS and its threat model

CEPS schemes seek to provide a technological way for publishers to abide by lawful takedown requests and enforce terms of service while maintaining a measure of transparency and accountability. Thus, *we envision a publisher who actively desires freedom of expression—including freedom from censorship—but must contend with legal and ethical guardrails that may demand censorship in some instances.* We refer to such a publisher as a *reluctant censor*. We assume that such a reluctant censor has *voluntarily* adopted the CEPS mechanism and willingly incurs any "reasonable" costs associated with doing so; nonetheless, it will comply with lawful censorship orders and may proactively and voluntarily remove content that egregiously violates its published terms of service. Of particular note, the publisher may be compelled by a *subpoena attacker* to engage in *covert censorship*—wherein it is compelled not only to censor but to do so "under the radar" in a bid to avoid a Streisand event. By employing CEPS, then, our reluctant censor seeks to render all censorship *overt*—the substance of the specific submission is censored *but the act of censorship itself remains decidedly conspicuous* in a cryptographically verifiable sense.

Indeed, one can reasonably liken CEPS to a cryptographic analog of *warrant canaries*—effectively legal signage indicating that no user data was disclosed in response to a subpoena—such as those added to the transparency reports of Apple (d. 2014), Pinterest (d. 2015), and Reddit (d. 2016), among others, in the wake of the Snowden revelations [38, 43]. While some commentators question the legal basis for warrant canaries [14, 29], their intention remains as laudable as their conceptual elegance remains compelling. Platforms already publish transparency reports documenting takedown requests and enforcement actions (e.g., Twitter's Transparency Center [53], Google's transparency reports [18], and similar efforts by major platforms), complemented by other credibility signals such as warrant canaries, "no-log" VPN claims, bulletproof hosting providers, and end-to-end encryption defaults. CEPS aims to be a cryptographic successor to such signals: instead of trusting self-reported numbers, users can obtain transferable evidence of covert censorship. Although publishers are not forced to deploy CEPS, history shows that platforms and services do invest in these credibility signals; disabling CEPS would itself signal that censorship is afoot, so rational publishers are incentivized to retain CEPS. Our threat model treats CEPS adoption as a voluntary but market-driven choice; the only way to defeat its goal is to keep CEPS running while covertly blocking extraction, which is exactly what CEPS aims to make cryptographically provable.

Threat model: We model all entities (i.e., publishers, users, and censors alike) as probabilistic polynomial-time (PPT) algorithms. There are then two main categories of attackers that the CEPS scheme must defend against: *covert censors* who seek to prevent some users from accessing specific content without enabling them to produce "evidence" that censorship is happening; and *false accusers* who seek to frame the publisher by implicating it in (i.e., falsifying "evidence" of) covert censorship that never happened. Network-layer adversaries are orthogonal; CEPS assumes an anonymous channel (e.g., Tor or mixnets) to hide which user issues which query while the PIR/indistinguishability properties hide which queries are extractions.

Both these attackers are fully malicious entities, but a *covert* censor tries not to expose their malicious behaviour. CEPS schemes specifically target scenarios where the publisher operates as a reluctant covert censor (presumably under coercion); this means that, for example, network-based censorship attacks are out of scope. Indeed, as a willing and active participant of the CEPS scheme, the publisher-*cum*-censor in our model can shut down its services arbitrarily and at will, and no clever use of cryptography can hope to prevent it from doing so, though such behaviour would come across as incriminating. We reiterate that CEPS schemes are primarily concerned with *covert censorship* by the publisher, and its security properties seek to force the publisher to perform all censorship *overtly*—or, better yet, not at all.

Ledger and timestamping: CEPS uses a tamper-evident, append-only ledger that imposes a partial order on events and provides unpredictable epoch nonces; we refer to this as the *timestamp oracle* (T). CEPS relies on T for adjudicating *happened-before* [27] claims about CEPS-relevant events. In practice, this is a commodity component: our deployment uses Hyperledger Fabric as the oracle/ledger, but any permissioned ledger with append-only semantics and fresh randomness per epoch suffices. The ledger orders events, prevents back-/forward-dating, and serves as a broadcast channel for tickets and overt takedown notices. A minimal interface (detailed in Appendix A) provides epoch numbers/nonces and append operations for tickets, proofs, and takedowns; the method names are self-explanatory. The throughput and latency requirements here are modest; in our deployment, crypto and uploads dominate costs, not the ledger.

While there exist many ways to realize a CEPS timestamp oracle, one obvious and elegant way to instantiate the timestamp

oracle is with a *permissioned blockchain* [1] or *proof-of-authority blockchain* [50] operated by a committee of trustees (possibly comprising a group of independent publishers, oversight bodies, activist organizations, NGOs, etc.). A blockchain is a tamper-evident ledger with built-in randomness, and adding additional timestamp oracle APIs (elaborated in Appendix A) to it would transform a blockchain into a plug-and-play CEPS timestamp oracle.

2.1 Bird's-eye view

A given CEPS deployment involves a publisher who *voluntarily* employs the CEPS scheme alongside many users, at least some of whom actively use the CEPS mechanisms to hold the publisher to account,[2] as well as the aforementioned timestamp oracle T.

When a user uploads new content to the publisher, the publisher can inspect and scrutinize said content to decide whether or not to host it. Such a decision may be fully automated and occur in real-time, or it may involve "holding" submissions until they can be subjected to human moderation—CEPS does not seek to force publication of materials, but rather to prevent publishers from silently reneging on standing commitments to host already-published content. If the publisher ultimately decides to publish the submission, the contributing user receives a tacit *commitment-to-host* from the publisher in the form of a *ticket*, t; that is, it receives a cryptographic blob that non-repudiably (i) authenticates the contents of the submission and associates it with a unique *CEPS handle*, (ii) attests to the provider's commitment-to-host the given content and make it available via the given CEPS handle, and (iii) binds the commitment-to-host with an unambiguous "time" at which the commitment takes effect. Rather than sending the ticket only to the submitting user, the publisher broadcasts it to all users via the append-only ledger by invoking $T.\mathsf{appendTicket}(\mathsf{t})$. From here, users can access and engage with the content using either the publisher's regular (presumably non-cryptographic and user-friendly) front-end or a CEPS-specific cryptographic interface (described later).

If the publisher subsequently chooses—or is compelled—to renege on its commitment-to-host, it may do so *overtly* by broadcasting a public *overt takedown announcement* via the append-only ledger by invoking $T.\mathsf{overtTakedown}(\mathsf{h}, \ldots)$ at or before the time it makes the offending submission inaccessible. Such an announcement references the submission's CEPS handle and allows the publisher to self-declare its rationale for censoring it. Any claims of censorship pertaining to that content will pass verification only if both (i) the commitment-to-host the content *happened-before* all evidence constituting the proof and (ii) all evidence constituting the proof *happened-before* any overt takedown announcement for the content.

The evidence itself arises due to user-generated queries. In particular, whenever a CEPS-aware user interacts with the publisher, its client software automatically generates cryptographic queries designed to obtain evidence of a breach of some standing commitment-to-host. Such queries may be either (i) non-targeted *probe queries*, which search for evidence of covert censorship of *arbitrary* submissions (i.e., submissions not necessarily related to what the user is currently consuming or suspects censorship of) or (ii) targeted *extraction queries*, which try to elicit evidence that a *specific* submission is presently subject to covert censorship (e.g., after apparent censorship was observed via the non-CEPS interface). Probe queries serve two purposes: first, they serve as "cover traffic" for targeted extraction queries and, second, they serve as a probabilistic means to uncover as-yet-unnoticed instances of covert censorship.

The CEPS scheme guarantees that—with a probability overwhelming in some system-wide tuning parameter—a user possessing sufficiently many query-response pairs referencing the same CEPS handle can either (i) extract the original contents of that submission (i.e., the content remains available and uncensored via CEPS) or (ii) generate a transferable *censorship proof* that attests to the nonavailability of the submission (i.e., the content is provably censored). In the latter case, the append-only ledger provides a publicly queryable source of ground truth regarding the covert-versus-overtness of the censorship; otherwise-valid censorship proofs will be rejected if any of their evidence was collected during or after the epoch where an overt takedown announcement was issued for that handle. Users can broadcast their censorship proofs via the append-only ledger, thereby publicizing covert censorship and perhaps triggering a dreaded Streisand event.

We illustrate CEPS use-case via the timeline diagram in Figure 1. At epoch $1, 3, 5$, Alice, Bob and Charlie respectively upload files $\mathsf{f}_0, \mathsf{f}_1, \mathsf{f}_2$ via an anonymous channel to `example.com`, indicated by arrows $A_{\text{up}}, B_{\text{up}}, C_{\text{up}}$. `example.com` stores the uploads under handles $\mathsf{h}_0, \mathsf{h}_1, \mathsf{h}_2$, and for each one it generates a ticket, which serves as a commitment-to-host. Each ticket is transmitted back through the same anonymous channel, and reaches the correct uploader. In the background, each CEPS-aware user, including Alice, Bob and Charlie, runs a daemon that continuously sends anonymous probe queries to `example.com`; these probes are routine and persistent. Bob's daemon uses these probe queries as cover traffic to hide its targeted extraction queries for h_1 to check f_1 is still being served faithfully. Alongside the aforementioned activity, normal users interact with `example.com` in the "ordinary" way, browsing via the internet. However, Charlie is a malicious user: his intention is to falsely implicate `example.com` of censorship. To this end, he had been trying to generate extract queries (along with his probe queries) for contents at h_2 prior to uploading f_2, hoping to demonstrate its inconsistency with the ticket he obtained in epoch 5. However, this plan was foiled since the "binding" timestamp on the ticket was after the "binding" timestamps on supposedly incriminating responses from `example.com`. At some epoch 6, `example.com` decides to remove f_0 for violating terms of service, and notifies the timestamp oracle T of the same; T publishes this notification in its ledger for epoch 6. Separately, sometime between epoch 4 and epoch 6, `example.com` is compelled by some coercive entity (e.g. a national security letter) to covertly censor the file stored at h_1. Recall that CEPS-aware users send continuous probe traffic via background daemons, and Bob has been utilizing this to hide his extraction queries. During epoch 5, Bob detects evidence that file f_1 at handle h_1 was censored. f_1 was later reinstated in epoch 6, however, this does not invalidate Bob's proof because of the "binding" timestamps to his probe responses. That evidence is then published through T as proof that f_1 was censored at least

[2] We envision deployments in which users of a CEPS-enabled publisher can be "CEPS agnostic", contributing and engaging with published content while being oblivious to the fact that the publisher has deployed a CEPS scheme.

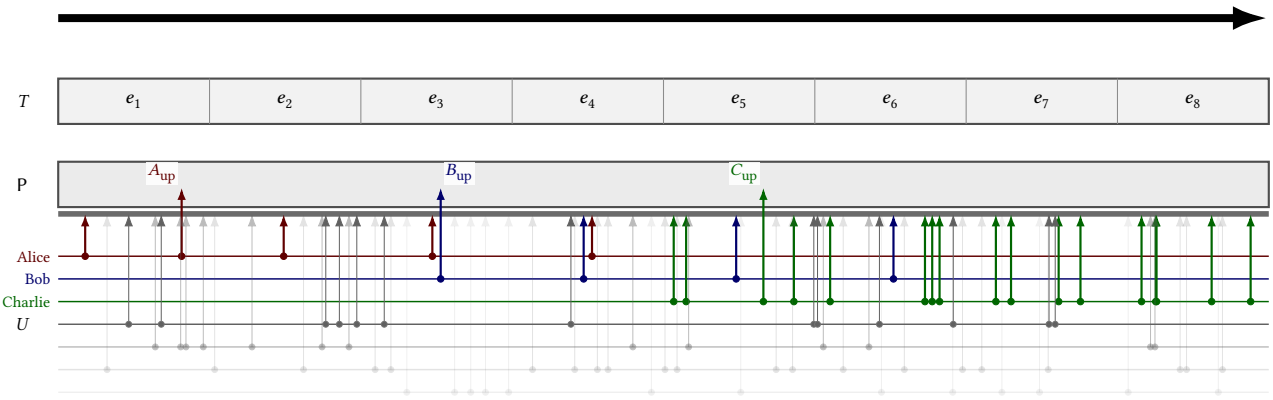


Figure 1: CEPS timeline and user interactions

during epoch 5, which can now be verified by other users of the system.

3 Defining censorship-evident publishing

We are now ready to formally define CEPS schemes and their constituent algorithms and protocols.

3.1 Notational conventions

In the sequel, we denote randomized algorithm A augmented with oracle access to T and taking as input x to produce output a as $a \leftarrow A^T(x)$ (or $a \leftarrow A^T(r; x)$ when prescribing internal randomness r). For 2-party interactive protocols, we extend this notation as $(a; b) \leftarrow \langle A^{T,R}(x), B^{T,S}(y)\rangle$, where A and B are interactive algorithms respectively having access to oracles T, R and T, S and outputting private outputs a and b. Here, both algorithms have access to a common oracle T in addition to their own, role-specific oracles R and S.

3.2 Constituent algorithms and protocols

Formally, a *censorship-evident publishing system* (CEPS) is a triple of PPT algorithms (Setup, Extractor, CheckProof) alongside a pair of 2-party interactive protocols (Post–Receipt, Request–Response) run by a publisher P and users u from some universal set of users U, where all parties have blackbox access to a common timestamp oracle T. CEPS assumes the existence of an *anonymous channel* over which users $u \in \mathsf{U}$ run the Request–Response protocol with P,[3] thereby preventing P from using network identifiers to distinguish which users submit which challenges. We stress that CEPS imposes no anonymity requirement for the Post–Receipt protocol (or non-CEPS workflows), allowing users to interact with CEPS-enabled services over a regular, non-anonymous network connection. To clarify this distinction, we affix a hat $\hat{u} \in \mathsf{U}$ to denote users in anonymity-required contexts, and use $u \in \mathsf{U}$ without a hat in contexts where anonymity is not required.

[3]The required anonymous channel can be realized with Tor [42], a mix network [7, 34], a sender-anonymous messaging system [8, 46], or other related primitives.

3.2.1 The Setup algorithm. $(\mathsf{pp}, \mathsf{sk}) \leftarrow \mathsf{Setup}^T(1^\lambda)$ is a PPT algorithm run once by P to initialize the scheme. It takes as input a security parameter $\lambda \in \mathbb{N}$, and generates (long-lived) public parameters pp and private key material sk for P. Parameters in pp—including public key pk for authenticating P's messages—are broadcast via T marking the start of epoch 0. These then serve as implicit input to subsequent CEPS algorithms and protocols.

3.2.2 The Post–Receipt 2-party interactive protocol. $(\mathsf{t_h}; \cdot) \leftarrow \langle u.\mathsf{Post}^T(\mathsf{f}), \mathsf{P.Receipt}^T(\mathsf{sk})\rangle$ is initiated by a user $u \in \mathsf{U}$ to upload new content f for hosting by P. If P agrees to host f, it produces in collaboration with u a commitment-to-host f in the form of a *ticket* $\mathsf{t_h}$ verifiably bound to (i) the submitted content f, (ii) a canonical CEPS handle h for accessing f, (iii) the upcoming epoch number $i' \leftarrow T.\mathsf{getEpoch}() + 1$ when the commitment-to-host takes effect, and (iv) the public identity key of P (as reflected in pp). P broadcasts the ticket to all of U by adding it to the timestamp oracle's ledger using $T.\mathsf{appendTicket}(\mathsf{h}, \mathsf{t})$.

3.2.3 The Request–Response protocols. The Request–Response protocols are a pair of 2-party interactive protocols initiated by an *anonymous* user $\hat{u} \in \mathsf{U}$ to query the status of a submission ostensibly hosted by P. There are two variants:

$$\big((\mathsf{b}, \varphi'_\mathsf{h}); \cdot\big) \leftarrow \langle \hat{u}.\mathsf{ProbeQ}^T(\varphi_\mathsf{h}), \mathsf{P.Response}^T(\mathsf{sk})\rangle\,,$$

which interrogates the status of a *(pseudo)randomly-selected* submission, and

$$\big((\mathsf{b}, \varphi'_\mathsf{h}); \cdot\big) \leftarrow \langle \hat{u}.\mathsf{ExtractQ}^T(\mathsf{h}, \varphi_\mathsf{h}), \mathsf{P.Response}^T(\mathsf{sk})\rangle\,,$$

which queries a specific user-provided handle. In both variants, $\hat{u}$ inputs a *client state* variable φ_h, which is the set of all authenticated observations (e.g., Request–Response tuples) about h known to $\hat{u}$. Both variants can culminate in one of three outcomes:

Outcome 0 ($\hat{u}$ outputs $\mathsf{b} = 0$): The state has been updated with a new observation o (i.e., $\varphi'_\mathsf{h} = \varphi_\mathsf{h} \cup \{o\}$) but the resulting set φ'_h does *not* contain observations sufficient to derive a censorship proof for h nor to recover the content f associated with it.

Outcome 1 ($\hat{u}$ outputs $\mathsf{b} = 1$): The state has been updated with a new observation o (i.e., $\varphi'_\mathsf{h} = \varphi_\mathsf{h} \cup \{o\}$) and the resulting set φ'_h contains observations sufficient to derive either a censorship proof π'_h for h or the content f associated with it.

Outcome 2 ($\hat{u}$ outputs $b = \perp$): This outcome arises when P returns a *malformed* response (i.e., a response that fails authentication checks and can thus be repudiated by P). In this case, $\varphi'_h = \varphi_h$ as no valid response was received.

We stress that P runs *the exact same* interactive algorithm, namely $\mathsf{P.Response}^T(sk)$, in both protocol variants. Indeed, jumping ahead, *it will be imperative that* P *cannot differentiate between interactions with $\hat{u}$ running $\hat{u}$.*$\mathsf{ProbeQ}^T(\varphi_h)$ *versus those in which $\hat{u}$ runs $\hat{u}$.*$\mathsf{ExtractQ}^T(h, \varphi_h)$; the presumed-anonymity of $\hat{u}$ is relevant to—indeed, inspired by the need for—this indistinguishability.

3.2.4 The Extractor oracle machine. The *extractor* $(\{f, \pi_h\}, \varphi'_h) \leftarrow \mathsf{Extractor}^{T, \hat{u}.\mathsf{ExtractQ}, \mathsf{P.Response}}(h, \varphi_h)$ is invoked by an *anonymous* user $\hat{u} \in \mathsf{U}$ to retrieve the content f associated with, or a censorship proof for, a given handle h. We model it as an oracle machine with access to the timestamp oracle T, as well as to the user $\hat{u}$ and publisher P, respectively, via their $\hat{u}$.ExtractQ and $\mathsf{P.Response}^T(sk)$ interfaces. It takes as input a handle h and client state φ_h comprising all authenticated observations (e.g., Request–Response tuples) about h known to $\hat{u}$, and it outputs either

(1) the content f associated with h, or
(2) a censorship proof π_h for h.

To accomplish this, the Extractor algorithm repeatedly invokes $\langle \hat{u}.\mathsf{ExtractQ}^T(h, \varphi_h), \mathsf{P.Response}^T(sk) \rangle$ using its oracle access, until it receives an output with $b = 1$ indicating that the updated φ_h contains a censorship proof π_h or the desired content f

3.2.5 The CheckProof algorithm. $\{0, 1\} \leftarrow \mathsf{CheckProof}(h, \pi_h)$ is a PPT algorithm invoked by a user $u \in \mathsf{U}$ to verify a censorship proof π_h for handle h. It outputs 1 to indicate that π_h is a valid censorship proof for h, and it outputs 0 otherwise.

3.2.6 PoCs as a CEPS scheme. Appendix C formally specifies and analyzes the original PoC construction of Martiny et al. [30] as a special (degenerate) case of the CEPS notion. What CEPS adds relative to PoCs is a formal extraction notion enabling succinct, transferable proofs even for large content and expands the adversary model (selective-failure blindness, extraction indistinguishability). Concretely, Streisand achieves asymptotic and practical speedups over PoCs (which effectively operate in a worst case where the "word size" equals the file size), while supporting equivocation and falsification proofs with word-level PIR and probabilistic Merkle-witness retrieval.

3.3 Defining censorship & proofs thereof

Notice that every commitment-to-host is codified with a ticket t_h that binds the submitted content f with a unique CEPS handle h for accessing that content. Consequently, in a CEPS deployment, there is a one-to-one correspondence between tickets and handles such that each corresponding ticket-handle pair (t_h, h) is computationally bound to some content f. We can thus define *censorship* in the CEPS context with reference to the pair (t_h, h) and, indirectly, to the content f itself.

Intuitively, we want our definition of censorship to capture the idea that it occurs whenever f is not available in its original, unaltered form through h, despite the commitment-to-host encoded in t_h. However, relying on some assumed "ground truth" about the file is problematic, since outside observers may not know which version of f is correct when faced with conflicting claims. To avoid this, we instead focus on what can be efficiently and unambiguously demonstrated; that is, we identify the kinds of evidence that serve as incontrovertible and efficiently checkable evidence that f is (or was) inaccessible in its original form, and then define "censorship" as any situation that can be demonstrated to a third party in this way.

Table 1: PoCs vs. CEPS/Streisand (informal comparison)

Aspect	PoCs (Martiny et al.)	CEPS/Streisand
Threat model	Single-server PIR; proof of substitution only	Expanded: equivocation + falsification; selective-failure blindness; extraction indistinguishability
Data granularity	Word size ≈ file size (worst case)	Word-level PIR; probabilistic Merkle-witness retrieval
Proof types	Substitution	Equivocation, falsification, substitution
Performance	Simulation/microbench; linear in file size; large proofs	Containerized PoC; asymptotic + concrete speedups; compact proofs; single-threaded baseline reported
Anonymity / channels	Not explicit	Anonymous channel + PIR to hide extraction attempts
Transferability	Yes (limited to substitution)	Yes (all proof types)

Concretely, the "evidence" comprising a censorship proof will primarily consist of authenticated responses to probe and extract queries made by some $\hat{u} \in \mathsf{U}$. Specifically, embedded within each response is a *claim* about the content associated with handle h; that is, an implicit tuple (h, F, y, i, σ) in which $F(\cdot)$ is some univariate function, y is the purported result of evaluating $F(f)$ for the content f bound to (t_h, h), i is the epoch number during which the claim $F(f) = y$ was made, and σ is a cryptographic authenticator (e.g., digital signature) that attests to P having made the claim $F(f) = y$ about handle h during epoch number i. Such a set is said to be *demonstrably contemporaneous with the commitment-to-host* h if and only if it is non-empty and each epoch number reflected in it resides within the half-open interval $[t_{\text{start}}, t_{\text{end}})$, where $t_{\text{start}} \coloneqq T.\mathsf{findEarliestEpoch}(h)$ and $t_{\text{end}} \coloneqq T.\mathsf{findLastEpoch}(h)$. Note that, as explained in Appendix A, we have $t_{\text{end}} = \infty$ if there has been no overt takedown announcement about h.

Definition 3.1. A *covert censorship proof* (often abbreviated as simply a *censorship proof*) for the handle h is a pair $\pi = (\Pi, w)$ with Π a set demonstrably contemporaneous with the commitment-to-host h and w a *witness* string[4] demonstrating that the claims

[4] In actual constructions, the witness string w is likely to be only "computationally convincing"; for example, in Streisand they are Merkle proofs. Consequently, the resulting censorship proof will be a computationally sound *argument* rather than an unconditionally sound proof; nevertheless, we ignore this distinction and simply refer

in Π cannot all possibly describe the same content (i.e., $\forall \mathsf{f} \in \{0,1\}^*$, $\exists(\mathsf{h}, F, y, i, \sigma) \in \Pi$ such that $F(\mathsf{f}) \neq y$).

Note that, since $\mathsf{t_h}$ is guaranteed to be available via T, the set Π can always include $\mathsf{t_h}$, enabling CheckProof to identify discrepancies between claims made by P during Request–Response interactions and the "ground-truth" content f that P originally committed to host. Specifically, $\mathsf{t_h}$ introduces a special equation $F(\mathsf{f}) = y$ for which y is computationally bound to the "correct" content f (e.g., a collision-resistant hash digest of f).

Sufficient claim sets: It is useful to consider the *sufficiency* of a set of claims $\Pi = \{(\mathsf{h}_j, F_j, y_j, i_j, \sigma_j)\}_{j\in\{1,\ldots,n\}}$ for recovering f. In particular, we say that a set Π which is demonstrably contemporaneous with the commitment-to-host h is *sufficient* if and only if there exists an algorithm $\mathcal{F}$ such that, for all possible submission contents f', the algorithm $\mathcal{F}$ uniquely recovers f' from the system of (not-necessarily-linear) equations $F_j(\mathsf{f}') = y_j$ for $j \in \{1,\ldots,n\}$

in time polynomial in n; i.e., *if* all claims are true, *then* there is an efficient algorithm to solve the system and recover the original contents f. For any sufficient set Π including $\mathsf{t_h}$, it is guaranteed to be possible to extract either (i) f if Π is *well constrained*, or (ii) a censorship proof π if Π is *over-constrained*.

3.4 Proof taxonomy

Currently, we have defined a censorship proof for h as a set Π that is *demonstrably contemporaneous* with the commitment-to-host for h, along with a witness string w demonstrating that the claims in Π are mutually inconsistent with the originally submitted content. There are three ways in which this inconsistency can manifest, which we discuss here.

Proofs of equivocation: A censorship proof may demonstrate some sort of equivocation from P about the content associated with h. In such equivocation-based proofs, Π will be a set of *mutually contradictory* Request – Response tuples, such as two distinct answers to the same question or a collection of answers that induce an over-constrained subsystem.

Proofs of falsification: As an alternative to demonstrating equivocation, a censorship proof may provide a witness string w that demonstrates that a set of Request–Response tuples is inconsistent with the ticket $\mathsf{t_h}$ for h. In such falsification-based proofs, Π will be an internally consistent set of plausible-looking Request–Response tuples and w will be some additional information that demonstrates that Π is inconsistent with $\mathsf{t_h}$.

Proofs of substitution: This is a special case of proof of falsification wherein the censorship proof consists of a complete (purported) copy of the contents alongside the ticket $\mathsf{t_h}$. In such substitution-based proofs, Π will be a set of Request–Response tuples that "successfully" extract to some value that is *inconsistent with* $\mathsf{t_h}$. We note that proofs of substitution have the largest proof sizes and extractor running times among the three proof types; moreover, they are the only kind of censorship proof possible in Martiny et al.'s PoCs.

to censorship *proofs* regardless of whether they offer computational or unconditional soundness.

3.5 CEPS security properties

To effectively disincentivize covert censorship, a CEPS scheme must provide some security guarantees, which we describe here.

3.5.1 Non-frameability. CEPS is *non-frameable* if users cannot forge or misrepresent tickets or query responses to probe and extraction queries in a way that falsely implicates P in covert censorship. This notion parallels unforgeability in digital signature schemes and message authentication codes (MACs); indeed, in our Streisand construction, this is partially provided by the so-called *existential unforgeability under chosen message attacks* (EUF-CMA) property of standard digital signatures.

The non-frameability attacker is a *false accuser* seeking to frame the publisher for covert censorship that never occurred. We model this as a PPT oracle machine $\mathcal{A} = \mathcal{A}^{T,\mathsf{P.Receipt},\mathsf{P.Response},T.\mathsf{Takedown}}$ with access to the timestamp oracle T and to the publisher P via its P.Receipt and P.Response interfaces from the Post–Receipt and Request–Response protocols. Notably, $\mathcal{A}$ can invoke T.Takedown to instruct P to overtly censor arbitrary content. Then, CEPS provides *non-frameability* if no such $\mathcal{A}$ can win Game 1 with probability non-negligible in the security parameter.

Note that winning Game 1 requires $\mathcal{A}$ to produce either (i) one or more forged claims—purportedly from P—about h to include in Π, (ii) a bogus witness w falsely demonstrating that the claims in Π are inconsistent with $\mathsf{t_h}$, or (iii) both forged claims in Π *and* a bogus witness w. The first case is mitigated using conventional authenticity mechanisms like digital signatures; the second case can be mitigated using cryptographic commitment schemes.

Definition 3.2 (Non-frameability). A CEPS scheme is *non-frameable* if, for every PPT algorithm $\mathcal{A}$ having oracle access to T (including T.Takedown) and to P via its P.Receipt and P.Response interfaces, there exists a negligible function $\varepsilon\colon \mathbb{N} \to \mathbb{R}^+$ such that $\mathcal{A}(1^\lambda)$ wins Game 1 with probability at most $\varepsilon(\lambda)$.

Game 1 (The CEPS frame-up game)**.**

> *Common input:* Security parameter, $1^\lambda \in 1^{\mathbb{N}}$.
> *Initialization:* P invokes $(\mathsf{pp}, \mathsf{sk}) \leftarrow \mathsf{Setup}^T(1^\lambda)$, sends pp to $\mathcal{A}$.
> *Probing:* $\mathcal{A}$ performs arbitrary oracle-assisted computation.[a]
> *Winning:* $\mathcal{A}$ wins if it outputs a *valid* handle-proof pair (h, π), i.e., a pair (h, π) for which $\mathsf{CheckProof}(\mathsf{h}, \pi) = 1$, and it loses otherwise.
>
> [a] $\mathcal{A}$ can use its oracles to post content to P, have content overtly censored by P, and/or submit probe and extraction requests to P.

3.5.2 Extractability. CEPS exhibits *extractability* if Extractor halts after an expected polynomial number of steps. Jumping ahead, this ensures that a user holding a valid CEPS handle h, absent any overt takedown announcement via T, can "efficiently" learn either (i) the content f bound to $\mathsf{t_h}$ or (ii) a censorship proof π for h. The formal CEPS model ultimately separates these concerns so that formal *extractability* focuses its attention on efficiently retrieving the original content f; subsequently, *selective-failure blindness* (discussed in Section 3.5.3) constrains P to ensure the extractor can also produce censorship proofs when f is not available.

Our notion of extractability draws inspiration from that of knowledge extraction in the context of interactive proof-of-knowledge (PoK) systems, formalized by Bellare and Goldreich [3]. However, whereas the knowledge extractors in PoK systems are rhetorical devices that exist solely within thought experiments, the Extractor in CEPS is a concrete algorithm that all users *must* be able to realistically run. This makes CEPS extraction more analogous to extraction in the context of PoR systems [24]; nevertheless, fundamental differences between the CEPS and PoR threat models and architectures necessitate a somewhat more intricate and nuanced definition for CEPS extractors.

Publisher availability: A key consideration in formalizing CEPS extractability is the availability of P via its P.Response interface. A publisher seeking to evade accountability for covert censorship may refuse to respond—or provide inauthentic responses—to stall extraction for as long as desired. Since CEPS treats inauthentic responses and non-responses equivalently (in that neither of them leads to transferable proofs), the treatment below makes no distinction between them.

We emphasize that, while no cryptographic mechanism can hope to force P to respond to CEPS queries, outright refusal by P to engage in the Request–Response protocol is effectively a *de facto* admission of guilt—albeit one that is neither transferable nor illuminates which specific handle(s) the publisher seeks to suppress. The anonymous channel over which users submit probe/extract queries, of course, forces P to disregard probe queries from *all* users, effectively broadcasting P's misbehaviour to all online CEPS users. Therefore, we consider only cases where P does *not* fully embrace this extreme and self-incriminating measure; that is, *we assume that* P *always responds to a non-trivial fraction of—if not most—probe and extract queries,* an idea formalized in Definition 3.3. We stress that this assumption is consistent with our modelling P as a *covert censor* who voluntarily adopts the CEPS scheme and wishes to avoid implicating itself in covert censorship.

Definition 3.3 ($(\mathsf{h}, p_{\mathsf{h}})$*-availability).* Fix a CEPS handle h such that $T.\mathsf{findLastEpoch}(\mathsf{h}) = \infty$, and a probability $p_{\mathsf{h}} \in (0, 1]$. Then P is $(\mathsf{h}, p_{\mathsf{h}})$*-available* if, in any given run of $\langle \hat{u}.\mathsf{ExtractQ}^T(\mathsf{h}, \varphi), \mathsf{P.Response}^T(\mathsf{sk})\rangle$, we have that $\hat{u}$ outputs an authenticated claim about h with probability equal to p_{h}. Here the probability is taken over the internal coin tosses made by $\hat{u}.\mathsf{ExtractQ}^T(\mathsf{h}, \varphi)$ and by $\mathsf{P.Response}^T(\mathsf{sk})$.

With Definition 3.3 in place, we can now state a formal definition for extractability.

Definition 3.4 (CEPS extractability). A CEPS scheme provides *extractability* if, for any handle h and $(\mathsf{h}, p_{\mathsf{h}})$-available PPT publisher P having oracle access to T, there exists a positive integer d such that $\mathsf{Extractor}^{T, \hat{u}.\mathsf{ExtractQ}, \mathsf{P.Response}}(\mathsf{h}, \varphi)$ halts after an expected $O(\lambda^d / p_{\mathsf{h}})$ steps.[5]

Note that this imposes an upper bound on the running time of a CEPS extractor as a function of (the availability of) the handle being extracted. Notably, Definition 3.4 ensures that a CEPS extractor is an expected PPT algorithm whenever p_{h} is a *noticeable function* (i.e., vanishes slower than the reciprocal of some polynomial). The next property, *selective-failure blindness*, ensures that if p_{h} is noticeable for even one handle h, then p_{h} is equally noticeable for all handles.

3.5.3 Selective-failure blindness. Intuitively, CEPS extraction exhibits *selective-failure blindness* if the publisher P cannot condition its $(\mathsf{h}, p_{\mathsf{h}})$-availability on the specific choice of h; that is, if P is powerless to make extraction efficient for some handles but inefficient for others. Since extraction always halts with either the file or a censorship proof as output, this property prevents P from leveraging selective non-responsiveness to stymie censorship proof extraction for specific covertly censored submissions, while allowing extraction for non-censored content.

The selective-failure game (Appendix B, Game 2) takes place between a selective-failure attacker—a *covert censor* (i.e., P under coercion)—seeking to prevent users from extracting either the associated content or a censorship proof for some specific handle h. We model this as a PPT algorithm $\mathcal{A} = \mathcal{A}^T$ with access to the timestamp oracle T that accepts Request–Response queries from an honest challenger $\hat{u}$, where $\hat{u}$ runs two Request–Response instances $\hat{u}_0, \hat{u}_1$, one for each $\mathcal{A}$-specified CEPS handle $(\mathsf{f}_0, \mathsf{h}_0)$ and $(\mathsf{f}_1, \mathsf{h}_1)$. We say that CEPS extraction provides *selective-failure blindness* if no such $\mathcal{A}$ can distinguish between the two $\hat{u}$-instances querying for these CEPS handles with a non-negligible advantage (i.e., with a probability non-negligibly higher than that of a random process). Winning the selective-failure blindness game is trivial if $\mathcal{A}$ can respond differently depending on which handle is under extraction; consequently, for this game to be hard, $\hat{u}$.ExtractQ queries must not reveal which handle was input to them. If so, then irrespective of f_1's status (i.e., whether it is covertly censored or available) an inability to win the game implies an inability to stop $\hat{u}$ from extracting with handle h_1 without equally disrupting $\hat{u}$'s ability to extract with handle h_0, implying that $p_{\mathsf{h}_0} \approx p_{\mathsf{h}_1}$.

Definition 3.5 (Selective-failure blindness). A CEPS scheme provides *selective-failure blindness* if, for every PPT algorithm $\mathcal{A}$ having oracle access to T, there exists a negligible function $\varepsilon\colon \mathbb{N} \to \mathbb{R}^+$ such that $\mathcal{A}(1^\lambda)$ wins the *selective-failure blindness* game with probability at most $1/2 + \varepsilon(\lambda)$.

3.5.4 Extraction indistinguishability and unobservability. The final two security properties limit the publisher's ability to detect and respond strategically to extraction attempts. The first property, *extraction indistinguishability*, insists P be unable to differentiate incoming probe queries (i.e., queries created by ProbeQ) from targeted extraction queries (i.e., queries created by ExtractQ); the second property, *extraction unobservability*, goes a step further by insisting that P cannot even tell when extraction is happening at all (as opposed to when all queries from all users are just probes). These two notions are related but distinct: one is about distinguishing between probe and extraction queries, while the other is about inferring the presence of extraction queries based on the timing and volume of incoming queries.

Intuitively, extraction indistinguishability is a cryptographic guarantee that tries to stop P from ignoring (or issuing inauthentic responses to) extraction queries while continuing to respond correctly to probes. Such an attack would enable P to prevent

[5] In practice, the bitlength $|\mathsf{f}|$ of each submission is assumed to be polynomial in λ, and then we typically expect the extractor to halt within $O(|\mathsf{f}|^d / p_{\mathsf{h}})$ steps.

users from extracting without any obvious indication to the larger population of probing users. Extraction unobservability is a non-cryptographic guarantee that takes this a step further to prevent P from feigning technical difficulties to avoid responding to queries when extraction is underway. Both properties ultimately rely on the anonymous communication channel to rule out trivial attacks that link queries to a common source based on network identifiers. Appendix B formalizes both properties.

4 Streisand

We now describe Streisand, our proof-of-concept instantiation of the CEPS notion formalized in the prequel. In Streisand, submissions are stored as rows in a 'vector-matrix' PIR [19] database $\mathbf{D}$, with one submission per row. The CEPS handle h is simply the 0-based row index of the associated content within $\mathbf{D}$. Each row is represented as a vector of w-bit words from $\mathbb{Z}_{2^\mathsf{w}}$ (with 0-padding applied to the final word, if needed). Because submissions will generally vary in length, the public parameters set a maximum word-length N. To create a coherent PIR database of fixed-sized rows, P pads shorter rows by cyclically repeating their word sequences until they reach length N.

Users construct probe and extraction queries to audit the availability of submitted content via a modified PIR workflow that either (i) recovers uncensored files word-by-word or (ii) produces a set of authenticated claims (and witnesses) that form a transferable censorship proof. To enable efficient word-level processing, P materializes N "synthetic databases", each corresponding to one column of the full database (including words repeated in padded rows). Denote these synthetic databases $\mathbf{D}_0, \ldots, \mathbf{D}_{\mathsf{N}-1}$. The querying client reveals the word index j in the clear while hiding the handle h (and any recovered value) inside its PIR query, enabling P to respond to the query using the jth synthetic database $\mathbf{D}_j$.

Word-based claims: Let $\mathsf{f} \in (\mathbb{Z}_{2^\mathsf{w}})^\mathsf{N}$ be some committed content (possibly padded to length). For each word index $j \in [0, \mathsf{N}-1]$, define $F_j \colon (\mathbb{Z}_{2^\mathsf{w}})^\mathsf{N} \to \mathbb{Z}_{2^\mathsf{w}}$ as $F_j(\mathsf{f}) \coloneqq \langle \vec{e}_j, \mathsf{f} \rangle$, where $\vec{e}_j$ is the jth standard basis vector over $\mathbb{Z}_{2^\mathsf{w}}$; that is, F_j extracts the jth w-bit word from f. The collection of functions $\{F_j\}_{j=0}^{\mathsf{N}-1}$ represents a family of *claims* about f that users may solicit from P: namely, that "the jth word of the content f associated with handle h is w_j". Here j is public while h, f, and w_j remain hidden.

Due to linearity, these word-based claims are particularly easy to support in vector-matrix PIR; indeed, that

$$F_j\big(\mathsf{PIR.Respond}(\mathsf{h})\big) \equiv \mathsf{PIR.Respond}\big(F_j(\vec{\mathsf{f}_\mathsf{h}})\big)$$

follows immediately from the definition of the F_j and the cyclic-shift invariance of product-traces [49]. Furthermore, because each synthetic database is a factor N smaller than $\mathbf{D}$, the right-hand side of this expression can be answered with a factor N less work than the left-hand side. Parallelization is especially straightforward by distributing the $\mathbf{D}_j$ across different physical servers.)

Witness-based claims: Streisand enables efficient verification of $F_j(\mathsf{f})$ claims using Merkle-tree commitments. For each submission f, P constructs a Merkle tree over the N words of f and embeds the resulting Merkle root in the ticket. Suppose censorship occurs (whether by substitution or equivocation) on word w_j of f, while a user u holds a Merkle proof for the authentic word. In this case, the Merkle proof already suffices as a censorship proof for w_j relative to the ticket. More generally, suppose censorship occurs on word w_j, while u holds a Merkle proof for the authentic word w'_j. If j and j' differ only in their $(k+1)$-least-significant bits, then u only needs to fetch the $2^k - 1$ additional leaves in the shared subtree to "complete the Merkle proof" and, thus, a censorship proof for w_j. This means, for example, that a submitting user who simply "remembers" a full Merkle proof for some arbitrary word can construct censorship proofs with a cost related to the distance from that word to the nearest censored word. In other words, an extracting user who possesses some (demonstrably correct) interior Merkle nodes can generally produce censorship proofs faster than a user who does not. This observation motivates a second family of claims: in addition to words w_j, Streisand users may request arbitrary nodes along their Merkle paths. Concretely, P uses the set of submission-specific Merkle trees to conceptually construct $\mathsf{O}(\mathsf{N} \lg \mathsf{N})$ additional synthetic databases, as follows: (i) for each w_j, define S_j as the sequence of $\mathsf{O}(\lg \mathsf{N})$ interior nodes in its Merkle path; call the kth-last node in S_j the "k-uncle" of w_j, and (ii) for each j, k, define $\mathbf{D}_{j,k}$ to be the database consisting of k-uncles corresponding to $\mathbf{D}_j$. Notice that $\mathbf{D}_{j,k} = \mathbf{D}_{j',k}$ whenever j and j' agree on all but their $(k+1)$-least-significant bits. Accordingly, Streisand allows u to request the k-uncle of w_j; we denote such a claim by $w_{j,k} = F_{j,k}(\mathsf{f})$. In this case, u reveals both j and k in plaintext, but h, f, and $w_{j,k}$ remain hidden.

Querying for claims: The word size w is a crucial tuning parameter in Streisand, affecting the number and sizes of synthetic databases. Intuitively, smaller values of w improve per-query probe and extraction efficiency, at the expense of increasing the number of queries required to fully extract content. To fully leverage small w, Streisand employs a customized PIR protocol based on the additive variant of ElGamal [44] over an elliptic curve group.

This is a decidedly unusual choice: decoding additive ElGamal queries requires solving an instance of the discrete logarithm (DL) problem, which is only feasible (but still costly) for very small values of w. However, in exchange for this inflated decoding cost, we obtain two concrete benefits:

(1) query responses are small, consisting of a pair of elliptic curve group elements; and
(2) server-side processing reduces to a single multiexponentiation with w-bit exponents, which is relatively fast and easy to evaluate with standard techniques.

Then, to mitigate DL decoding costs, we "absorb" the discrete logarithm into the Merkle tree hash function; that is, we use the hash function $\mathsf{H}'(x) \coloneqq \mathsf{H}(g^x)$, where H is a standard cryptographic hash function (e.g., SHA-3) and g is the elliptic curve group generator used in the additive ElGamal. This way, clients need only partially decode responses (i.e., up to—but not including—the costly DL step) in order to validate them against the Merkle tree.

Therefore, we have responses for

(1) word-based claims $F_j(\mathsf{f})$ of the form $(g^{w_j} h^r, g^r)$, where w_j is w bits; and
(2) witness-based claims $F_{j,k}(\mathsf{f})$ of the form $(g^{w_{j,k}} h^r, g^r)$ where $w_{j,k}$ is a cryptographically large hash digest.

Hence, for suitable choices of the word size w, it is feasible to fully decode responses for F_j claims—allowing the extractor to recover all of f, albeit at a steep cost—whereas u never fully decodes $F_{j,k}$ claims. Importantly, no value need be fully decoded in order to validate responses or construct censorship proofs; full decoding is required *only* when extracting *uncensored* content.

Interfacing with the timestamp oracle: In addition to the Merkle root, each ticket embeds an epoch number indicating when the commitment-to-host takes effect. To generate a probe or extraction query, user u

(1) samples a cryptographically long nonce $R \in \{0,1\}^*$ and fetches the current epoch nonce N_i from T; and then it
(2) applies a KDF to (R, N_i) to derive a pair of pseudorandom seeds (a_1, a_2) where (i) a_1 seeds a PRNG that samples (j,k) pairs,, and (ii) a_2 seeds a randomness for the PIR query.

Notice that this design ensures that, given the user-selected nonce R, epoch number i (not epoch nonce N_i), and handle h, it is possible to reconstruct the entire CEPS query, Qry. Moreover, seeding the KDF with the epoch nonce N_i (not the epoch number i) acts as an assertion that this query was created at or after the start of the ith epoch; that is, this design prevents u from later claiming that Qry (and any response it invokes) was produced in an epoch later than the current epoch.

Upon receiving (j, k, Qry) from u, P retrieves the epoch nonce $N_{i'}$ from T, evaluates Qry against $\mathbf{D}_j$ and $\mathbf{D}_{j,k}$ to get Response_j and $\text{Response}_{j,k}$, and returns $(\sigma, \text{Response}_j, \text{Response}_{j,k}, i')$, where σ is a digital signature on $(j, k, \text{Qry}, \text{Response}_j, \text{Response}_{j,k}, N_{i'})$. Notice that including the epoch nonce $N_{i'}$ in the signature (instead of i') prevents P from falsely claiming that a response issued in i' was actually created in a later epoch, after some future overt takedown request.

Since Streisand's primary goal is to dissuade covert censorship, we do not expect proofs of censorship to arise often. However, when censorship occurs, retrieving elements of the Merkle proof with every query increases detection speed *and* reduces the overall extraction cost by enabling users to passively assemble proofs of censorship piecewise. Since higher level Merkle tree nodes get returned more often than ones below them, we balanced retrieval probabilities via the two ways a user can infer them: (i) directly computing the node and (ii) requesting them from the server, and came up with heuristic per-level probabilities for Merkle nodes. The "passive proof assembly" nature of CEPS means that censorship detection is not latency dependent but rather runs as a background process, and depends on the client's and server's throughput to a significantly higher degree than the latency of any underlying anonymous channel.

Censorship proofs: Before providing formal descriptions of the constituent Streisand algorithms and protocols, we note what the three types of censorship proof look like in this design. First note that u can learn interior Merkle tree nodes in three distinct ways, namely (i) it can *retrieve* them directly from P by requesting an $F_{j,k}$ claim, in which case it holds a signature attesting to the validity of that interior node, (ii) it can *compute* them (perhaps recursively) from their immediate children, who may themselves be either requested (with signature) or computed from values ultimately requested (with signatures); or (iii) it can *remember* them from the original file, in which case their validity may be demonstrated via a Merkle proof relative to the ticket (which is signed). Thus, a *proof of equivocation* is just any pair of Request–Response tuples yielding contradictory answers for either a common F_j claim or $F_{j,k}$ claim. A *proof of falsification* is any larger set of Request–Response pairs that yield a contradiction in some interior Merkle node (e.g., a single node shown to be retrieved, computed, or remembered in two distinct ways, yielding two distinct values). Finally, a *proof of substitution* is a special case of fabrication involving a complete set of leaf nodes that yield a different Merkle root than the ticket.

4.1 Streisand protocols

We now formally define the algorithms comprising Streisand.

Setup: The setup algorithm generates public parameters and associated private key materials for P, including public parameters for additive ElGamal-based PIR and a key pair for digital signatures, and then publishes the public parameters via the timestamp oracle to initiate the system.

Specifically, on input a security parameter 1^λ, it fixes (i) a DDH--hard group $\mathbb{G}$ with generator g of λ-bit prime order q, and (ii) a signing-verification key pair (sk, pk) for digital signatures, alongside system-wise constants like the word size w, maximum word-length N of a submission, a key-derivation function (KDF) to use, and which hash cryptographic function H to employ in the Merkle tree. It also initializes the (initially empty) database $\mathbf{D}$ and its (also empty) synthetic counterparts. It publishes the public parameters $\text{pp} \coloneqq \big((\mathbb{G}, g, q), \text{w}, \text{N}, \text{H}; \text{pk}\big)$ to T, commencing epoch 0.

Post-Receipt: The Post–Receipt protocol is initiated by u to submit new content f to P. In it, u uses $(\text{w}, \text{N}, \text{H}, g)$ from the public parameters to express f as a length-N vector of w-bit words and construct the Merkle tree for f using this vector; it notes the Merkle root root_f, and (optionally) records Merkle paths to some arbitrary subset of nodes for use in later censorship proofs.[6] u then uploads f to P in cleartext.

If P agrees to host f, it reproduces $\vec{\text{f}}$ and the Merkle commitment to it, appends $\vec{\text{f}}$ to the end of $\mathbf{D}$ (and correspondingly updates all synthetic databases), and then constructs a ticket $\text{t}_\text{h} \coloneqq (\text{root}_\text{f}, \text{h}, i, \sigma)$ where h is the row number of $\vec{\text{f}}$ in $\mathbf{D}$, i is the epoch number when the commitment-to-host takes effect, and σ is a digital signature over the tuple $(\text{root}_\text{f}, \text{h}, N_i)$. It returns (σ, h, i) to u and queues up $(\text{root}_\text{f}, \text{h}, \sigma)$ to be reflected in epoch i of T. The commitment-to-host f takes effect as soon as the epoch advances to i.

Request-Response: The Request–Response protocol is initiated by u to query the status of hosted content. For probe queries, u selects a handle arbitrarily (e.g., uniformly at random, perhaps from a u-chosen subset of submissions), and then it invokes the same logic used for extraction queries below.

For extraction queries using handle h, the user $\hat{u}$ connects to P over an anonymous channel and does the following:

[6] Note that if f is shorter than $\text{w} \cdot \text{N}$ bits, then u uses cyclic padding described earlier to extend it into a length-N vector.

(1) Choose a random value $R \in \{0,1\}^\lambda$ and fetch the current epoch number i and nonce N_i from T; using the KDF, compute the pseudorandom seeds $(a_1, a_2) \leftarrow \mathrm{KDF}(R, N_i)$.
(2) using a_1, sample a uniform (pseudo)random word index j and uncle height k; using a_2, sample a secret key $x \in \mathbb{Z}_q^*$ for additive ElGamal, and a length-N vector of random values $r_0, \ldots, r_{\mathsf{N}-1}$ to use in the next step;
(3) setting $G = g^x$, compute the PIR query

$$\mathsf{Qry} = \big((g^{r_0}, G^{r_0}), \ldots, (g^{r_{j-1}}, G^{r_{j-1}}), (g^{r_j}, g \cdot G^{r_j}), (g^{r_{j+1}}, G^{r_{j+1}}), \ldots, (g^{r_{N-1}}, G^{r_{N-1}})\big)$$

(4) then, send (j, k, Qry) to P over the anonymous channel.

Upon receiving the request, P does the following:

(1) evaluate Qry against $\mathbf{D}_j$ and $\mathbf{D}_{j,k}$ to yield a pair of PIR responses (each a pair of elements in $\mathbb{G}$), $\mathsf{Response}_j$ and $\mathsf{Response}_{j,k}$;
(2) compute a digital signature σ on the tuple $(j, k, \mathsf{Qry}, \mathsf{Response}_j, \mathsf{Response}_{j,k}, N_{i'})$ where i' is the current epoch number.
(3) return $(\mathsf{Response}_j, \mathsf{Response}_{j,k}, i', \sigma)$ to $\hat{u}$ over the anonymous channel.

Upon receiving its response $(\mathsf{Response}_j, \mathsf{Response}_{j,k}, i', \sigma)$ from P, $\hat{u}$ uses i' to fetch $N_{i'}$ from T, and then it checks that σ is a valid signature on $(j, k, \mathsf{Qry}, \mathsf{Response}_j, \mathsf{Response}_{j,k}, N_{i'})$; if not, it aborts. [7] Otherwise, it outputs $(\mathsf{h}, i, R, \mathsf{Response}_j, \mathsf{Response}_{j,k}, i', \sigma)$.

Extract: The Extract algorithm takes as input a handle h and any prior knowledge φ_h about h (e.g., interior Merkle nodes remembered, alongside their Merkle proofs from Post–Receipt, or claims about h learned via prior probe or extraction queries), and it attempts to recover the associated content or a censorship proof for the same. It does this by repeatedly issuing requests for h using the Request–Response protocol. Specifically, the extractor checks each subset of values in φ_h for a censorship proof and returns one if it exists. If more than one set constitutes a censorship proof, the extractor outputs an arbitrary one among those having the smallest total size. While the anonymous channel combined with the hiding properties of additive ElGamal suffice to imply extraction indistinguishability, they do little to support extraction unobservability.

Achieving extraction indistinguishability: Streisand users can achieve extraction unobservability by blending their extraction queries among the background probe traffic, typically by setting extraction query rate to be "very low" compared to probe query rate. Since users do not have a way to effectively determine the background probe rate, they can utilize the timestamp oracle to do this: whenever a CEPS user is online and actively querying, they broadcast this status to the timestamp ledger, and refresh their status every epoch they are active. This way, any user joining the system can estimate the background probe rate by looking at the number of actively querying users in the current epoch and set their extraction rate accordingly.

[7] Recall that CEPS regards inauthentic responses as being equivalent to non-responses.

4.2 Security analysis

Next, we informally analyze the security guarantees of Streisand.

Non-frameability: Streisand ensures that malicious parties cannot falsely frame honest publishers. Both tickets and responses are authenticated using EUF-secure signatures. Thus, framing requires forging either a ticket signature, a response signature, or a Merkle proof relative to some (signed) node of the Merkle tree. The first two possibilities are directly prevented by the EUF security of the digital signatures; the latter follows from the *second-preimage resistance* property of cryptographic hash functions.

Extractability: Extractability requires that publishers cannot discern which file—if any—is being extracted. This comprises two main sub-properties, namely:

(1) **Selective-failure blindness**, which is guaranteed by the *query privacy* of the underlying PIR protocol, itself an immediate consequence of the IND-CPA security of additive ElGamal encryption. Breaking query privacy of additive ElGamal-based PIR to distinguish between probe and extraction queries reduces to the breaking DDH in $\mathbb{G}$.
(2) **Extraction indistinguishability**, which is likewise guaranteed by query privacy of the underlying PIR protocol. In particular, extraction queries are indistinguishable from probe queries because both utilize the same Request–Response interface. More precisely, the only difference between probe and extraction queries is how they select which handle to query for—which is ultimately hidden due to selective-failure blindness. Combined with anonymity, this ensures that extraction cannot be distinguished from background auditing.

4.3 Experimental evaluation

In this section, we evaluate our proof-of-concept CEPS instantiation, Streisand. Our aim in developing and evaluating Streisand is to demonstrate the feasibility of CEPS and explore surface tradeoffs of the various CEPS parameters, not necessarily to prescribe a deployment-ready CEPS construction. We use the Enron email dataset [26], padding/truncating messages to 2712 B (517,400 emails; 1.307 GiB) and treating regex-redacted PII as censorship. Records are 339 words of 64 bit each.

Parameter choices: We use the Ristretto255 group [10] with 64-bit words to enable practical decoding of additive ElGamal PIR responses.

System setup and workload model: We ran all of our experiments on a single server with an Intel Core i9-13900KS and 128 GiB of RAM. Clients, the publisher, and the timestamp oracle (instantiated with the open-source permissioned blockchain, Hyperledger Fabric [23]) run in separate Docker containers. We emulate network conditions with `tc` at a modest 10 Mbit/s throughput (Ookla global index [33]) (70 ms, 0.1% packet loss). The core is in C++ using Rust's `curve25519-dalek` [40] for Ristretto255; Hyperledger Fabric provides the timestamp oracle. [8] Probe and extract queries are issued by a background daemon; users do not wait on responses, so sustained

[8] We opted for curve25519-dalek over libsodium to avoid repeated point compression/decompression.

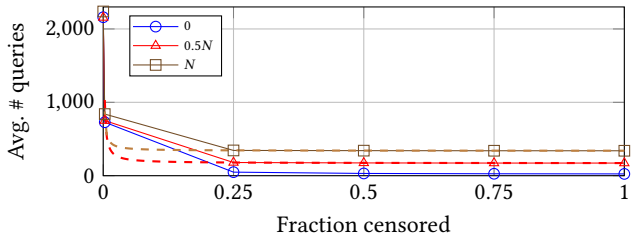


Figure 2: Avg. no. queries vs fraction of file censored (No. words = 339)

throughput (detection/extraction rate) is the metric of interest. Both client and server workloads are embarrassingly parallel, though our prototype runs single-threaded per role; thus, the performance numbers represent a very pessimistic estimate.

Measured costs (Enron dataset): For this experiment, we configured the server to provide responses from the original (unredacted) Enron email dataset for the first 30 queries, after which it was switched to the redacted database. The one-time-per-record ticket generation cost, corresponding to Post–Receipt, took about 6.5 ms. Each query took approximately 43 s (query generation, response computation, and network transmission) both before and after the switch. Different records in the dataset underwent varying degrees of redaction, from 0% to 87%. For the record having an 87% redaction rate, producing a proof of censorship required only an average of 6.2 post-redaction queries, on average. We provide example JSON-formatted proofs of censorship for equivocation (on the Enron dataset) and falsification in Appendix E.

Client-side costs are dominated by linear-size query generation, where each element costs one Ristretto scalar multiplication plus one extra Ristretto addition. This results in queries of size 31.58 MiB for the Enron dataset generated at an average of approximately 8.7 s, with upload taking approximately 30.4 s on the configured link.

On the server side, computing responses—specifically, computing additive ElGamal PIR responses over Merkle witnesses—dominates the computation time. We use a simple case of Pippenger's multi-scalar multiplication algorithm [35] for computing additive ElGamal PIR responses. We observe from Figure 4, unsurprisingly, that Streisand's PIR response computation scales linearly with the number of rows. Responses are 128 B (i.e., four Ristretto elements for words plus witnesses). Server computation is dominated by Merkle-witness terms and averages approximately 4 s per query under the configured link and hardware.

Censorship detection performance: The primary experiment of interest is measuring the average number of queries it takes for Streisand to detect censorship. Figure 2 shows the relation between number of queries and fraction of file censored, averaged over 100 trials. Each line corresponds to the number of queries performed by the client before the file was censored. As a reminder, the number of words per file is 339 for our experiments. A detailed analysis of theoretical expected values and comparison with experimental values is in Appendix F.

Scalability and limitations: Measurements presented here are single-threaded on a 1.3 GiB dataset and are crypto- and upload-heavy (31.58 MiB per query; PIR over Merkle nodes dominates). Parallelizing probe/extract daemons, swapping in faster (possibly larger-proof) PIR or non-PIR mechanisms, and exploring sharding and ORAM-like layouts are plausible paths to higher throughput.

Throughput and trade-offs: Streisand's PIR choice minimizes proof size but makes upload costs and server computation heavy; PIR is sufficient for extraction indistinguishability but may not be necessary. Faster (e.g., lattice-based) PIR or non-PIR designs could trade larger proofs for higher throughput. Parallelizing probe/extract daemons would linearly raise throughput.

5 Conclusion and future work

We formalized CEPS and built Streisand as a proof-of-concept deployment (containerized clients, publisher, and Hyperledger Fabric oracle) on a realistic Enron-derived dataset with regex-based PII redaction and controlled network conditions. The experiments demonstrate feasibility but are crypto- and upload-heavy: our single-threaded background daemon takes approximately 43 sec per query, though only an average of 6.2 post-censorship queries (approximately 4.4 min) were needed to prove censorship of an 87% redacted record.

Future directions include: (i) scaling: exploiting parallelism, exploring faster (e.g., lattice-based) PIR or other indistinguishability mechanisms, and sharding/ORAM-like designs; (ii) richer deployments: multi-server settings—with updated definitions to match the setting—and end-to-end runs with varied ledgers/anonymous channels; and (iii) incentives and transparency: deepening adoption drivers and integrating with existing transparency-reporting practices. Streisand is a starting point; we expect subsequent designs to tighten efficiency while preserving transferability and indistinguishability guarantees.

Acknowledgments

This work has been made possible in part by financial support from the Government of Canada via the *National Cybersecurity Consortium (NCC)* through Spearhead Grant 2024-1346 and the *Natural Sciences and Engineering Research Council of Canada (NSERC)* through Discovery Grant RGPIN-2019-04821.

References

[1] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. 2021. Permissioned Blockchains: Properties, Techniques and Applications. In *Proceedings of SIGMOD/PODS 2021*. ACM, Xi'an, China, 2813–2820. https://doi.org/10.1145/3448016.3457539

[2] Anonymous. 2012. The Collateral Damage of Internet Censorship by DNS Injection. *ACM SIGCOMM Computer Communication Review* 42, 3 (July 2012), 21–27. https://doi.org/10.1145/2317307.2317311

[3] Mihir Bellare and Oded Goldreich. 1993. On Defining Proofs of Knowledge. In *Advances in Cryptology: Proceedings of CRYPTO 92 (LNCS, Vol. 740)*, Ernest F. Brickell (Ed.). Springer, Santa Barbara, CA, USA, 390–420. https://doi.org/10.1007/3-540-48071-4_28

[4] Gunnar Blom, Lars Holst, and Dennis Sandell. 1994. *Problems and Snapshots from the World of Probability*. Springer, New York, NY. https://doi.org/10.1007/978-1-4612-4304-5

[5] Cecylia Bocovich, John A. Doucette, and Ian Goldberg. 2017. Lavinia: An Audit-Payment Protocol for Censorship-Resistant Storage. In *Proceedings of FC 2017*

(LNCS, Vol. 10322). Springer, Sliema, Malta, 601–620. https://doi.org/10.1007/978-3-319-70972-7_34
[6] Kevin D. Bowers, Ari Juels, and Alina Oprea. 2009. Proofs of Retrievability: Theory and Implementation. In *Proceedings of CCSW@CCS 2009*. ACM, Chicago, IL, USA, 43–54. https://doi.org/10.1145/1655008.1655015
[7] David L. Chaum. 1981. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM (CACM)* 24, 2 (February 1981), 84–90. https://doi.org/10.1145/358549.358563
[8] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. 2015. Riposte: An Anonymous Messaging System Handling Millions of Users. In *Proceedings of IEEE S&P 2015*. IEEE, San Jose, CA, USA, 321–338. https://doi.org/10.1109/sp.2015.27
[9] Leo de Castro and Keewoo Lee. 2024. VeriSimplePIR: Verifiability in SimplePIR at No Online Cost for Honest Servers. In *Proceedings of USENIX Security 2024*. USENIX Association, Philadelphia, PA, USA, 5931–5948. https://www.usenix.org/conference/usenixsecurity24/presentation/de-castro
[10] Henry de Valence, Jack Grigg, Mike Hamburg, Isis Lovecruft, George Tankersley, and Filippo Valsorda. 2023. The Ristretto255 and Decaf448 Groups. IETF RFC 9496. https://www.rfc-editor.org/info/rfc9496
[11] Sarang Dharmapurikar, Praveen Krishnamurthy, Thomas S. Sproull, and John W. Lockwood. 2004. Deep Packet Inspection Using Parallel Bloom Filters. *IEEE Micro* 24, 1 (January–February 2004), 52–61. https://doi.org/10.1109/mm.2004.1268997
[12] T. Dierks and E. Rescorla. 2008. The Transport Layer Security (TLS) Protocol Version 1.2. IETF RFC 5246. https://www.rfc-editor.org/info/rfc5246
[13] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. 2013. Protocol Misidentification Made Easy with Format-Transforming Encryption. In *Proceedings of CCS 2013*. ACM, Berlin, Germany, 61–72. https://doi.org/10.1145/2508859.2516657
[14] Electronic Frontier Foundation. 2014. Warrant Canary Frequently Asked Questions. Archived EFF Deeplinks post. https://www.eff.org/deeplinks/2014/04/warrant-canary-faq
[15] European Parliament. 2022. Combating Child Sexual Abuse Online. https://www.europarl.europa.eu/RegData/etudes/BRIE/2022/738224/EPRS_BRI(2022)738224_EN.pdf.
[16] Freedom House. 2023. Freedom in the World 2023. https://freedomhouse.org/sites/default/files/2023-03/FIW_World_2023_DigtalPDF.pdf.
[17] Daniil Gavrichenkov and Artyom Gavrichenkov. 2015. Breaking HTTPS with BGP Hijacking. Black Hat USA 2015 working paper. https://www.blackhat.com/docs/us-15/materials/us-15-Gavrichenkov-Breaking-HTTPS-With-BGP-Hijacking-wp.pdf
[18] Google LLC. 2024. Google Transparency Report. https://transparencyreport.google.com/.
[19] Syed Mahbub Hafiz and Ryan Henry. 2017. Querying for Queries: Indexes of Queries for Efficient and Expressive IT-PIR. In *Proceedings of CCS 2017*. ACM, Dallas, TX, USA, 1361–1373. https://doi.org/10.1145/3133956.3134008
[20] Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. 2023. One Server for the Price of Two: Simple and Fast Single-Server Private Information Retrieval. In *Proceedings of USENIX Security 2023*. USENIX Association, Anaheim, CA, USA, 3889–3905. https://www.usenix.org/conference/usenixsecurity23/presentation/henzinger
[21] Amir Houmansadr, Giang T. K. Nguyen, Matthew Caesar, and Nikita Borisov. 2011. Cirripede: Circumvention Infrastructure Using Router Redirection with Plausible Deniability. In *Proceedings of CCS 2011*. ACM, Chicago, IL, USA, 187–200. https://doi.org/10.1145/2046707.2046730
[22] House of Commons of Canada. 2022. Bill C-291. https://www.parl.ca/DocumentViewer/en/44-1/bill/C-291/first-reading.
[23] Hyperledger Project. 2024. Hyperledger Fabric Documentation. https://hyperledger-fabric.readthedocs.io/.
[24] Ari Juels and Jr. Burton S. Kaliski. 2007. PORs: Proofs of Retrievability for Large Files. In *Proceedings of CCS 2007*. ACM, Alexandria, VA, USA, 584–597. https://doi.org/10.1145/1315245.1315317
[25] Sheharbano Khattak, Tariq Elahi, Laurent Simon, Colleen M. Swanson, Steven J. Murdoch, and Ian Goldberg. 2016. SoK: Making Sense of Censorship Resistance Systems. *Proceedings on Privacy Enhancing Technologies (PoPETS)* 2016, 4 (2016), 37–61. https://doi.org/10.1515/popets-2016-0028
[26] Bryan Klimt and Yiming Yang. 2004. The Enron Corpus: A New Dataset for Email Classification Research. In *Proceedings of ECML 2004 (LNCS, Vol. 3201)*. Springer, Pisa, Italy, 217–226. https://doi.org/10.1007/978-3-540-30115-8_22
[27] Leslie Lamport. 2019. Time, Clocks, and the Ordering of Events in a Distributed System. In *Concurrency: The Works of Leslie Lamport*. ACM, New York, NY, USA, 179–196. https://doi.org/10.1145/3335772.3335934
[28] Kevin Lewi. 2023. WhatsApp Key Transparency. https://www.usenix.org/conference/pepr23/presentation/lewi Keynote presentation.
[29] Moxie Marlinspike. 2014. Provide a "Warrant Canary" for the TextSecure Server (Issue 34). GitHub issue, archived. https://web.archive.org/web/20141027143819/https://github.com/WhisperSystems/whispersystems.org/issues/34
[30] Ian Martiny, Ian Miers, and Eric Wustrow. 2018. Proof of Censorship: Enabling Centralized Censorship-Resistant Content Providers. In *Proceedings of FC 2018 (LNCS, Vol. 10958)*, Sarah Meiklejohn and Kazue Sako (Eds.). Springer, Nieuwpoort, Curaçao, 99–115. https://doi.org/10.1007/978-3-662-58387-6_6
[31] Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Edward W. Felten, and Michael J. Freedman. 2015. CONIKS: Bringing Key Transparency to End Users. In *Proceedings of USENIX Security 2015*. USENIX Association, Washington, DC., USA, 383–398. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/melara
[32] Steven J. Murdoch and Ross Anderson. 2008. Tools and Technology of Internet Filtering. OpenNet Initiative background paper. https://murdoch.is/papers/opennet08tools.pdf
[33] Ookla, LLC. 2024. Speedtest Global Index. https://www.speedtest.net/global-index.
[34] Ania M. Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. 2017. The Loopix Anonymity System. In *Proceedings of USENIX Security 2017*. USENIX Association, Vancouver, BC, Canada, 1199–1216. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/piotrowska
[35] Nicholas Pippenger. 1976. On the Evaluation of Powers and Related Problems. In *Proceedings of FOCS 1976*. IEEE, Houston, TX, USA, 258–263. https://doi.org/10.1109/sfcs.1976.21
[36] The Invisible Internet Project. 2024. I2P Site (Eepsite); Frequently Asked Questions. https://geti2p.net/en/faq#I2P%20Site.
[37] Reddit Inc. 2024. Reddit Content Policy. https://www.redditinc.com/policies/content-policy.
[38] The Guardian. 2013. NSA Files: Decoded. https://www.theguardian.com/world/interactive/2013/nov/01/snowden-nsa-files-surveillance-revelations-decoded.
[39] The Invisible Internet Project. 2024. The Invisible Internet Project (I2P). https://geti2p.net/en/.
[40] The RustCrypto Developers. 2024. The curve25519-dalek Cryptography Library. Rust crate documentation. https://docs.rs/curve25519-dalek/latest/curve25519_dalek/
[41] Inc. The Tor Project. 2024. Onion Services. https://tb-manual.torproject.org/onion-services/.
[42] The Tor Project, Inc. 2024. The Tor Project. https://www.torproject.org/.
[43] The Washington Post. 2016. Edward Snowden Revelations—Article Hub. https://www.washingtonpost.com/edward-snowden-revelations/ad3b27c1-37d7-4d41-b0ae-c9f5c2e572b2_note.html.
[44] Osman Ugus, Alban Hessler, and Dirk Westhoff. 2007. Performance of Additive Homomorphic EC-ElGamal Encryption for TinyPEDS. In *Proceedings of FGSN 2007*. IEEE, Aachen, Germany, 55–58.
[45] United States Congress. 2024. 18 U.S. Code § 2258A: Reporting Requirements of Providers. https://www.law.cornell.edu/uscode/text/18/2258A.
[46] Adithya Vadapalli, Kyle Storrier, and Ryan Henry. 2022. Sabre: Sender-Anonymous Messaging with Fast Audits. In *Proceedings of IEEE S&P 2022*. IEEE, San Francisco, CA, USA, 1953–1970. https://doi.org/10.1109/SP46214.2022.9833601
[47] Marc Waldman, Aviel D. Rubin, and Lorrie Faith Cranor. 2000. Publius: A Robust, Tamper-Evident, Censorship-Resistant, and Source-Anonymous Web Publishing System. In *Proceedings of USENIX Security 2000*. USENIX Association, Denver, CO, USA, 59–72.
[48] Wikipedia contributors. 2024. Streisand Effect — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Streisand_effect.
[49] Wolfram Research, Inc. 2024. Matrix Trace—from MathWorld. A Wolfram Web Resource, created by Eric W. Weisstein. https://mathworld.wolfram.com/MatrixTrace.html
[50] Gavin Wood. 2017. Proof of Authority Private Chains (PoA). Archived Ethereum guide resource. https://github.com/ethereum/guide/blob/master/poa.md
[51] Eric Wustrow, Scott Wolchok, Ian Goldberg, and J. Alex Halderman. 2011. Telex: Anticensorship in the Network Infrastructure. In *Proceedings of USENIX Security 2011*. USENIX Association, San Francisco, CA, USA, 327–342. https://www.usenix.org/legacy/events/sec11/tech/full_papers/Wustrow.pdf
[52] X Corp. 2024. The X Rules: Hateful Conduct Policy. https://help.x.com/en/rules-and-policies/hateful-conduct-policy.
[53] X Corp. 2024. X Transparency Report—Removal Requests. https://transparency.x.com/en/reports/removal-requests.

A Timestamp oracle interfaces

In this appendix, we provide additional detail on the interfaces that are necessary for a CEPS timestamp oracle and ledger discussed in Section 2.

(1) $i \leftarrow \mathsf{getEpoch}()$, which returns the current epoch number $i \in \mathbb{N}$;

(2) $N_i \leftarrow \mathsf{nonceFromEpoch}(i)$, which returns the epoch nonce $N_i \in \{0,1\}^\lambda$ associated with epoch number i, or a distinguished failure symbol $\perp$ if there is not yet an epoch number i;

(3) $\mathsf{appendTicket}(\mathsf{h}, \mathsf{t})$, which enqueues a handle–ticket pair (h, t) to be recorded in the ledger entry for the upcoming epoch (as created by the next advanceEpoch() call) and returns nothing;

(4) $\mathsf{appendPoC}(\mathsf{h}, \pi)$, which enqueues a handle-proof pair (h, π) to be recorded in the ledger entry for the upcoming epoch (as created by the next advanceEpoch() call) and returns nothing;

(5) $(i+1, N_{i+1}) \leftarrow \mathsf{overtTakedown}(\mathsf{h}, m)$, which enqueues an overt censorship announcement (with rationale m) for handle h to be recorded in the ledger entry for the upcoming epoch, and then returns the output of advanceEpoch();

(6) $(i+1, N_{i+1}) \leftarrow \mathsf{advanceEpoch}()$, which creates a new epoch by (i) incrementing the most recent epoch number i, (ii) sampling a uniform random nonce $N_{i+1} \in_R \{0,1\}^\lambda$, (iii) recording all currently enqueued events as having occurred during the new epoch $i+1$, and then (iv) returning the new epoch number–nonce pair $(i+1, N_{i+1})$, whereafter getEpoch() returns $i+1$ and $\mathsf{nonceFromEpoch}(i+1)$ returns N_{i+1} (meanwhile, $\mathsf{nonceFromEpoch}(j)$ still returns N_j for all $0 \le j \le i$); and

(7) $\{(i_0, e_0), \ldots, (i_n, e_n)\} \leftarrow \mathsf{getEvents}(\mathsf{h})$, which returns an ordered list of epoch number–event pairs (i_j, e_j) comprising all events e_j in the ledger associated with handle h alongside the epoch numbers i_j in which those events are recorded.

In a CEPS implementation, only the publisher should hae the privileges necessary to invoke overtTakedown(), advanceEpoch();[9] all other methods are arbitrarily invokable by all users, publishers, and censors. In addition to the above interface, we define a triple of helper functions that wrap getEvents() to retrieve (metadata about) specific events of interest:

(8) $\{i_0, \infty\} \leftarrow \mathsf{findEarliestEpoch}(\mathsf{h})$ queries the oracle for the list of events $\{(i_0, e_0), \ldots, (i_n, e_n)\} \leftarrow \mathsf{getEvents}(\mathsf{h})$ and then it outputs the epoch number i_0 from the earliest recorded event e_0 about h; if the event list is empty, it outputs ∞ to indicate that no such event has been recorded yet.

(9) $\{\mathsf{t}, \perp\} \leftarrow \mathsf{fetchTicket}(\mathsf{h})$ queries the oracle for the list of events $\{(i_0, e_0), \ldots, (i_n, e_n)\} \leftarrow \mathsf{getEvents}(\mathsf{h})$ and then it checks, for j ranging from 0 up to n, if e_j is a handle-ticket pair. If it finds a handle-ticket pair $e_j = (\mathsf{h}, \mathsf{t}_\mathsf{h})$, then it immediately returns t_h; if no such pair is found, it eventually outputs a distinguished failure symbol $\perp$ to indicate that no such ticket is available.

(10) $\{i_j, \infty\} \leftarrow \mathsf{findLastEpoch}(\mathsf{h})$ queries the oracle for the list of events $\{(i_0, e_0), \ldots, (i_n, e_n)\} \leftarrow \mathsf{getEvents}(\mathsf{h})$ and then it checks, for j ranging from 0 up to n, if e_j is an overt takedown announcement. If it finds an overt takedown announcement $e_j = (\mathsf{h}, m)$ published in epoch number i_j, then it outputs i_j; otherwise, it outputs ∞ to indicate that no such overt takedown announcement exists yet. (In case of multiple takedown announcements, this helper always returns the epoch number of the chronologically *first* such announcement.)

[9] The advanceEpoch() method may also run on a schedule, e.g., with the epoch automatically advancing if the publisher has not invoked advanceEpoch in the last, say, 10 or 20 minutes.

B Game-based definitions of security properties

In this appendix, we provide formal game-based security definitions for extraction indistinguishability and extraction unobservability.

B.1 Selective failure blindness

Game 2 (The CEPS selective-failure game)**.**

Common input:
- Security parameter, $1^\lambda \in 1^\mathbb{N}$.

Initialization:

$\mathcal{A}$: $(\mathsf{pp}, \mathsf{sk}, \mathsf{f}_0, \mathsf{f}_1) \leftarrow \mathcal{A}(1^\lambda)$, where pp and sk are public parameters and private values, respectively,[a] and where $\mathsf{f}_0, \mathsf{f}_1$ are arbitrary submissions satisfying $|\mathsf{f}_0| = |\mathsf{f}_1|$. $\mathcal{A}$ sends $(\mathsf{pp}, \mathsf{f}_0, \mathsf{f}_1)$ to u.

u : Upon receiving $(\mathsf{pp}, \mathsf{f}_0, \mathsf{f}_1)$ from $\mathcal{A}$, u initiates two runs of the Post–Receipt protocol to publish both submissions, namely

$$(\mathsf{t}_{\mathsf{h}_0}; \cdot) \leftarrow \langle u.\mathsf{Post}^T(\mathsf{f}_0), \mathcal{A}.\mathsf{Receipt}^T(\mathsf{sk})\rangle, \text{ and}$$
$$(\mathsf{t}_{\mathsf{h}_1}; \cdot) \leftarrow \langle u.\mathsf{Post}^T(\mathsf{f}_1), \mathcal{A}.\mathsf{Receipt}^T(\mathsf{sk})\rangle .$$

u also initializes a pair of sets $\varphi_0 \coloneqq \{\}$ and $\varphi_1 \coloneqq \{\}$.

Probing: u tosses an unbiased coin $b \in \{0,1\}$ and spins up two concurrent instances of the Extractor algorithm

$$\{\mathsf{f}_b, \pi_b\} \leftarrow \mathsf{Extractor}^{T, \mathcal{A}.\mathsf{Response}}(\mathsf{h}_b, \mathsf{t}_{\mathsf{h}_b}, \varphi_0)$$
$$\{\mathsf{f}_{1-b}, \pi_{1-b}\} \leftarrow \mathsf{Extractor}^{T, \mathcal{A}.\mathsf{Response}}(\mathsf{h}_{1-b}, \mathsf{t}_{\mathsf{h}_{1-b}}, \varphi_1),$$

such that the two instances alternate their queries to $\mathcal{A}$.Response in turns. *Notice that the order of the invocations of the two* Extractor *instances is dictated by the outcome of u's random coin toss.*

Winning: u reveals the output (either extracted content or a PoC) of whichever Extractor instance halts first; $\mathcal{A}$ wins if u reveals f_0 and loses if u reveals any other value.

[a] Normally, $\mathcal{A}$ would invoke $(\mathsf{pp}, \mathsf{sk}) \leftarrow \mathsf{Setup}$; however, we leave this purposely vague to allow $\mathcal{A}$ to generate pp dishonestly if doing so will help it win the game.

B.2 Extraction indistinguishability

Game 3 (Extraction indistinguishability game)**.**

Common input:
- Security parameter, $1^\lambda \in 1^\mathbb{N}$;
- Inter-probe query arrival time distribution, R_0;
- Inter-extraction query arrival time distribution, R_1.

Initialization:

$\mathcal{A}$: $(\mathsf{pp}, \mathsf{sk}, \mathsf{f}_0) \leftarrow \mathcal{A}(1^\lambda)$, where pp and sk are public parameters and private values, respectively, and where f_0 is an arbitrary file. $\mathcal{A}$ sends $(\mathsf{pp}, \mathsf{f}_0)$ to u.

u : Upon receiving $(\mathsf{pp}, \mathsf{f}_0)$ from $\mathcal{A}$, u initiates a run of the Post−Receipt protocol to publish f_0, namely

$$(\mathsf{t}_{\mathsf{h}_0}; \cdot) \leftarrow \langle u.\mathsf{Post}^T(\mathsf{f}_0), \mathcal{A}.\mathsf{Receipt}^T(\mathsf{sk})\rangle .$$

u also samples $S \leftarrow X_{R_0,R_1}$, initializes sets $\varphi_0 \coloneqq \{\}$ and $\varphi_1 \coloneqq \{\}$, and then sets the starting timestamp to $\mathsf{ts}_0 = 0$.

Probing: For $(\mathsf{ts}_i, b_i) \in S$ from $i = 1, 2, \ldots,$

1. u sends timestamp ts_i to $\mathcal{A}$, indicating when the next (i.e., ith) query will arrive (the query type b_i stays private);
2. $\mathcal{A}$ does the following arbitrarily and at will: Invokes T.overtTakedown() to overtly censor files and instructs u to initiate runs of the Post−Receipt protocol to publish new files;
3. If $b_i = 0$, then u initiates a run of
   $\big((b, \varphi_0); \cdot\big) \leftarrow \langle u.\mathsf{ProbeQ}^T(\varphi_0), \mathcal{A}.\mathsf{Response}^T(\mathsf{sk})\rangle$
   Otherwise, if $b_i = 1$, then u does the following
   - If `done_flag` $= 1$, then u informs $\mathcal{A}$ that the game is over.
   - Otherwise, u initiates a run of
     $\big((b, \varphi_1); \cdot\big) \leftarrow \langle u.\mathsf{ExtractQ}^T(\mathsf{h}_0, \varphi_1), \mathcal{A}.\mathsf{Response}^T(\mathsf{sk})\rangle$
     and then it sets `done_flag` $=$ isSufficient(φ_1)

Winning: $\mathcal{A}$ outputs $j \in [0, ..., i-1]$; it wins if $(ts_j, 1) \in S$ corresponds with an extract query.

B.3 Extraction unobservability

Game 4 (Extraction unobservability game)**.** *Common input:* Security parameter, $\lambda \in \mathbb{N}$; probe query rate distribution R_0, extraction query rate distribution R_1.

Initialization: $(\mathsf{pp}, \mathsf{f}_0) \leftarrow \mathcal{A}(1^\lambda)$, where pp are public parameters and f_0 is an arbitrary file, and then it sends $(\mathsf{pp}, \mathsf{f}_0)$ to u.

Upon receiving $(\mathsf{pp}, \mathsf{f}_0)$ from $\mathcal{A}$, u initiates a run of the Post−Receipt protocol to publish f_0, namely

$$(\mathsf{t}_{\mathsf{h}_0}; \cdot) \leftarrow \langle u.\mathsf{Post}^T(\mathsf{f}_0), \mathcal{A}.\mathsf{Receipt}^T(\mathsf{sk})\rangle .$$

u also samples $S \leftarrow X_{R_0,R_1}$ and initializes $\varphi \coloneqq \{\}$ and $\mathsf{Qry} \coloneqq \{\}$. u also tosses a fair coin with output $\mathsf{c} \in \{0, 1\}$.

Probing: Based on the output of the coin toss c, u does the following:

- $\mathsf{c} == 0$: u invokes $\mathsf{ProbeQ}^T(\varphi)$, and sends probe queries to $\mathcal{A}$ for all timestamps in S.
- $\mathsf{c} == 1$: u invokes $\mathsf{ProbeQ}^T(\varphi)$ and $\mathsf{ExtractQ}^T(\mathsf{h}_0, [\mathsf{t}_{\mathsf{h}_0}], \varphi)$ based on query type corresponding to every timestamp in S, and sends the resulting queries to $\mathcal{A}$.

Winning: At any point, $\mathcal{A}$ outputs its guess for u's coin toss c', and wins if $\mathsf{c}' == \mathsf{c}$.

C Martiny-Miers-Wustrow's *Proofs of censorship* as CEPS

In this appendix, we translate Martiny et al.'s construction [30] using our formalism in Section 3. Martiny et al.'s construction utilizes single server cPIR, an existentially unforgeable signature scheme, and a collision-resistant hash function.

C.1 Protocols

Setup: The server generates public parameters for the single-server cPIR and public-private key pair for the signature scheme, and broadcasts the public parameters and public key.

Post-Receipt: The client uploads a file to the server. The server chooses a handle at the database to store the file in. Then, the server hashes the file, and computes a signature over this hash, the file handle and timestamp when the file was uploaded. The server generates a ticket using this signature, hash of the file, file handle and timestamp, and sends it to the client. Upon receiving the ticket, the client verifies the signature, and checks the timestamp to make sure it is in the "recent past". If these checks fail, the client attempts to upload the file till it receives a valid ticket.

Request-Response: At some later point in time, the client generates a random seed and uses it to create a PIR query for the file handle in the ticket. The server computes the PIR response, and computes a signature over the tuple of query, response and timestamp when the request was received, and sends the signature and the tuple over which it was computed.

Extract: Upon receiving a response from the server, the client decodes the PIR response, and compares the decoded output to the hash contained in the ticket. If the decoded output does not match the hash in the ticket, the client publishes the random seed, ticket, and server's signed response as a "censorship proof".

CheckProof: A verifier takes as input the published censorship proof from the previous step, and does the following:

(1) check timestamps to make sure the ticket or server was not generated before the file was uploaded, and the server response was not generated before the ticket;
(2) verify signature on the ticket to ensure that it was indeed signed by the server and not forged by a malicious entity;
(3) regenerate PIR query and check signature on the server's response; and
(4) extract file from server's PIR response and verify against the ticket.

If any of these steps fail, the verifier is rejects the censorship proof as invalid.

C.2 Security properties

Here, we provide informal arguments that this construction satisfies a subset of security properties for CEPS.

Non-frameability: Non-frameability follows from the existential unforgeability of the signature scheme used in the construction.

Selective failure blindness: Selective failure blindness follows from the query privacy of the PIR scheme. Since query privacy guarantees that the server cannot distinguish between queries to specific file handles, the server cannot selectively reject queries.

We note that while this scheme allows a user to retrieve files by dint of using PIR to retrieve a file in its entirety, it provides neither

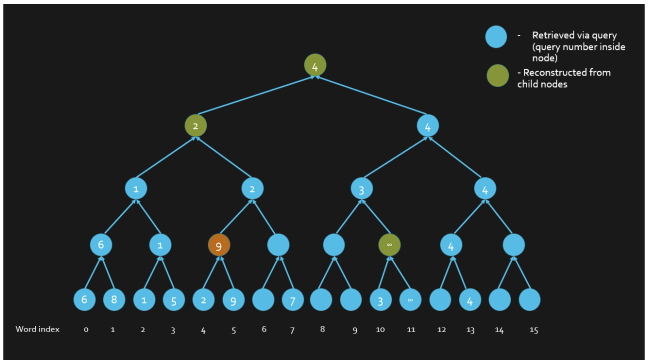


Figure 3: Probabilistic retrieval of intermediate Merkle nodes

extraction indistinguishability nor unobservability as formalized in Section 3.5.4. The response algorithm of Martiny et al.'s proof-of-censorship construction requires the server to perform PIR over the entire database for every client query, which makes routine probes and auditing an expensive affair for the server. In fact, according to their experimental evaluation in [30, §5.1], response computation at the server takes about 2 hours per query. This incentivizes clients to only query the server when it suspects censorship has happened for a file, which in turn makes this scheme not satisfy extraction indistinguishability.

D Determining close-to-optimal witness retrieval probabilities

The general heuristic we used to determine a candidate probability for retrieving intermediate nodes in the Merkle tree is to balance the probability of retrieving an intermediate node. For illustration, consider leaf nodes 0-3 from Figure 3 which demonstrates sample probabilistic Merkle node retrieval, which fall under the same height-2-or-more sub-tree, and the parent node of nodes 2, 3. The expected number of queries needed to recover one of nodes 2 or 3 is $\frac{n}{2}$ (inverse of probability of choosing one of a specific pair of nodes), and the expected number of queries to choose its sibling is $n-1$, so the expected number of queries for getting the pair of nodes 2, 3, and consequently recovering its parent node via recovering the children, is $\frac{3n}{2}-1$. To quantify direct retrieval probability of a parent level node, say p_1 is the probability of requesting the parent of nodes 2, 3 as part of either node 0's or 1's Merkle proof. The expected number of queries to recover node 0 (or 1) is n, so the total probability of recovering nodes 2, 3's parent node as part of node 0's or 1's Merkle proof is $\frac{p_1}{n}$. Equating the two probabilities, we get $p_1 = \frac{1}{3-\frac{2}{n}}$. In order to utilize CEPS as an effective auditing scheme, the number of words (equal to number of leaf nodes in the tree) need to be large, so $n \gg 2$ is a reasonable assumption to make, giving $p \approx \frac{1}{3}$. We then wrote a script in `Python` to then perform this analysis recursively for arbitrary tree heights and heuristically determine probabilities for intermediate nodes in lieu of using a closed form solution.

We also needed to account for disparity between costs for computing word responses versus Merkle witness responses. Streisand uses we use 64-bit words and 256-bit witnesses, which makes witness level response computation costs 7–8 times as expensive as word level. Consequently, we scale these per-level heuristic probabilities by a factor 1/8 to account for that, which did not result in much difference in terms of censorship detection power, but made average per-response computation approximately 40% of what it was without this scaling.

E JSON-formatted proofs of censorship

We present here two sample proofs of censorship written to the append-only ledger. They are hex-encoded representations of the various values from client-server interaction that resulted in the client detecting censorship, and any third-party verifier can do the same verification process as the client to verify the validity of these proofs.

Sample proof of equivocation from the Enron dataset:


```
{
  "equivocation_proofs": [
    {
      "first": {
        "client_secret_exponent": "7c90f4c247f206726d97e7a46915cbb5038b8
        ↪  bc46fcd6b84138dd233f677a905",
        "leaf_index": 62,
        "merkle_level": 1,
        "record_index": 436178,
        "response": [
          "bc065fa9dbb842f1a619f0035b15f6b719b37b0bcb1da7fc63cc2a4dd9924f
          ↪  3a",
          "8a8eb311d77d350bd41ba102a2f3718b15635649c4668207779f849757ff99
          ↪  0d",
          "448a91c35df868da9a561e0eb4f0b3d4e8ce22d64d578633643b9c4e360487
          ↪  2c",
        "f04e19cc8cd62ae1574e382db46f83743559ab5438dc44e93a86c2b2c180bf4f"
        ],
        "signature": "3046022100d0f4df0305ded160fdbefd8ae8e1b6265fdb4b4e
        ↪  a4ae912b5119b8edbe481e000221008ca4a31d4288adbf5af77bffaae87e
        ↪  7ff1ee03542258507a8522d4998945a456",
        "signature_valid": true
      },
      "second": {
        "client_secret_exponent": "2c6f4d9b077620ade39a4cadd3ef625b4ab2c
        ↪  e90bda6081e00954e1453890e09",
        "leaf_index": 62,
        "merkle_level": 2,
        "record_index": 436178,
        "response": [
          "6a37f72c21dff78e53b48c17cde46c288aab6ae543314cf8f0ea7c71a1c73e
          ↪  51",
          "86fe31330305899cc694bd7e04ef525284973be6cef377799fed576485ee41
          ↪  2c",
          "b878faa617e97a899839158843fa12898abbdd9130083554b24c8c506d09c3
          ↪  49",
        "9accadeb03635c0cf7236f8ee6f51e7e9f186cf6626ec9e1066579967c72b731"
        ],
        "signature": "304502205820770a1767b1e799baff1da2bdd3f3d5f6d76e1f
        ↪  cdcf85eb7f06c3b1bf4b0d022100f02722194fdae0bee0a8d1e1dae94cf2
        ↪  f4f46ea4666d85d7031aea3c1d519b0d",
        "signature_valid": true
      }
    }
  ],
  "falsification_proofs": []
}
```


Sample proof of falsification:


```
{
  "equivocation_proofs": [],
  "falsification_proofs": [
    {
      "responses": [
        {
          "client_secret_exponent": "b1a7892d62810378f27c541609c24cee571
          ↪  f1146cc46475bbd15e56af8badf03",
          "leaf_index": 51,
          "merkle_level": 4,
          "record_index": 0,
          "response": [
```

```
        "10c768fe528f3f9ba5ad002f8ede3ef0a31f159dedc473cd96d6b82a843c
        ↪ 362c",
        "563a9f9d93f8e65cc263ae7d9b86d9b0de4e09643fcacb57b0d56576bc80
        ↪ 0d57",
        "6a5f8f1ef637b8010a05ffac61b23e955fadfac31cde1a6ec3fc99f590a2
        ↪ c04e",
        "ce00d6ce6323d2ef09138049c1e23572375073f0d40390bc276a473ffcef
        ↪ 3e6a"
       ],
      "signature": "3045022100e15bdbbfae66deda1b80a8fed47a44dc1d6c7ff
      ↪ af1250b17bd8ab2bacf93b9290220686dd006d7c0773c25f0327eb9730
      ↪ 9805f4a0c4060078c4f116f46d1dcb07c86",
      "signature_valid": true
     },
     {
      "client_secret_exponent": "01529f9e5636406c5626fe1d6c4c416de7b
      ↪ c264719115c913f76cca2befcaa0b",
      "leaf_index": 61,
      "merkle_level": 1,
      "record_index": 0,
      "response": [
       "1a3c40dfc8ef75a5f5c7dd6e3c5be061052d42ebaa1be70d760f98838607
       ↪ df7b",
       "7c0dc4684dfe8c3bb24534fac29d75ee39344231ca558541a6e32d3e8667
       ↪ 8d12",
       "900a3f452374a202a9f1820aa82e78f907bb9218bfd4fac95d794941750e
       ↪ dc2e",
       "f47646e51d6ddafa5b5be069dbcb07fc62947bc064b52397cefaaea90de4
       ↪ b707"
      ],
      "signature": "3045022011f8ce2ef832919dd00503b3eac95fc8cc2f508f5
      ↪ 38726f03874b7eb22d0ebe5022100ac061270d73076be7bf6022b9d0bc
      ↪ 95d80394cf5a30445911118a047015b7b23",
      "signature_valid": true
     },
     {
      "client_secret_exponent": "33b0f2967dfd23ea1424865cb5153b6f08b
      ↪ e9d8669151e1f481c31d08233bc0e",
      "leaf_index": 61,
      "merkle_level": 0,
      "record_index": 0,
      "response": [
       "60d80ebdd45483c03cf0617a4d24c169aba5000ceffb17e8ba946b696ccd
       ↪ af70",
       "9c86a74f22acb47febe10e05b7f5e9a5891abc0ad92ed0277211899b60bd
       ↪ 7c59",
       "a465b4f1eb9f48c686495e1b3cd65649c0c8a00d78e4cea699230aeee384
       ↪ ff17",
       "74fe3b8182ac780f54b9209c6f779e96837b66af77374c252cd256a5edf6
       ↪ 2647"
      ],
      "signature": "304502210095de294a7e7786fc391e1b0d987b40678a4a34c
      ↪ 92b6f44c9f0845a3e8c59b31c022032c9ab7b2cf2a8c0e074ecf9c2ac0
      ↪ 77e46a6f3882a7c27492414880558074 8f8",
      "signature_valid": true
     }
    ]
   }
  ]
}
```


F Analytic expression for total number of queries to detect censorship

The number of queries required to extract the uncensored file from our experiments is close to the Coupon Collector's expectation of 1975 queries, and as more of the file is censored, there are significantly fewer queries. The analytic approximation for the expected total number of queries $t(s, r, N)$ on a censored database, with client probes prior to censorship, is

$$t(s, r, N) = s + \frac{1}{r(1 - e^{\frac{-s}{N}})}$$

where $r > 0$ is the fraction of censored content, $s > 0$ is the number of queries prior to censorship, N is the record size in words. A full derivation is provided later in this section. We note that Figure 2 agrees with this analytic expression. When $(r = 0.5, s = N/2)$ and $(r = 0.75, s = N)$, the expression predicts an expectation of 174.583 and 341.109 total queries respectively; our experiments gave us an average of 173.92 ± 4.93899 and 339.96 ± 1.43471 for these cases. Further, the number of post-censorship queries is inversely related to number of pre-censorship probe queries; for instance, when $s = N/2$, the client only needed between 2 and 10 additional queries to detect censorship when $r > 0.25$, consistent with our analytic expression.

F.1 Derivation for the analytic expression

Let the number of words in a record be N, $r > 0$ be the ratio of file censored, and $s > 0$ be the number of client queries before censorship occurred. Let expected number of words censored = rN. Since clients query for words uniformly at random, let w be the expected number of unique words seen by client prior to censorship.

$$\Pr[\text{Word } w_i \text{ was not seen in } S \text{ queries}] = \left(1 - \frac{1}{N}\right)^s \approx e^{\frac{-s}{N}}$$

$$\implies \Pr[\text{Word } w_i \text{ was seen in } S \text{ queries}] = 1 - e^{\frac{-s}{N}}$$

$$\implies \text{Expected no. words seen} = w = rN\left(1 - e^{\frac{-s}{N}}\right)$$

After censorship,

$$\Pr[\text{Seeing a censored word}] = c = \frac{w}{N} = r\left(1 - e^{\frac{-s}{N}}\right)$$

Therefore, expected number of total queries t is

$$t(s, r, N) = s + \frac{1}{c} = s + \frac{1}{r\left(1 - e^{\frac{-s}{N}}\right)}$$

That is, expected number of post-censorship queries to detect censorship is $\frac{1}{r\left(1-e^{\frac{-s}{N}}\right)}$.

Note that this occupancy-style argument changes to the Coupon Collector's [4] style counting argument when $r = 0$. When $s = 0$, this occupancy style argument is no longer valid, because the number of words seen before censorship is 0, and probability that a word seen pre-censorship will be seen again is exactly 0.

What happens when $s = 0$: There are still two ways to detect censorship: via falsification or via equivocation, i.e., reconstructing Merkle path and finding a node that is inconsistent with a server response. Detecting equivocation is similar to the birthday paradox for censored words. The expectation of queries over only censored words is then $E[T_c] \approx \text{sqrt}(\frac{\pi r N}{2})$. Since r is the fraction of file censored, expected total queries to detect equivocation is then $E[T_{eq}] \approx \frac{1}{r}E[T_c] = \text{sqrt}(\frac{\pi N}{2r})$. For falsification, we fall back to our heuristic argument of minimizing queries for retrieving intermediate nodes via directly querying compared to reconstructing them, and the expected number of queries to detect overall censorship via either equivocation or falsification is the minimum of the two expected query counts.

G Plots for PIR computation costs

We present a plot showing time taken for PIR query and response generation for Streisand.

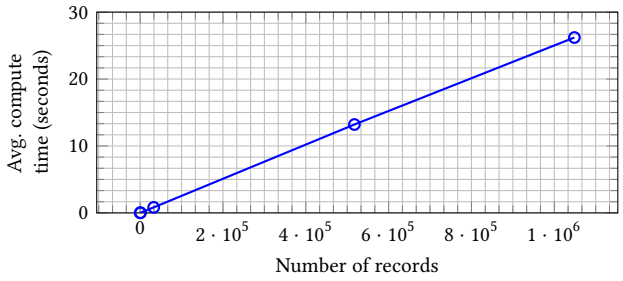


Figure 4: Total client + server compute time vs number of database records