On Preserving Privacy in Content-Oriented Networks

Somaya Arianfar, Teemu Koponen, Barath Raghavan, and Scott Shenker
Aalto University, Nicira Networks, ICSI, and UC Berkeley

ABSTRACT

The recent literature has hailed the benefits of content-oriented network architectures. However, such designs pose a threat to privacy by revealing a user's content requests. In this paper, we study how to ameliorate privacy in such designs. We present an approach that does not require any special infrastructure or shared secrets between the publishers and consumers of content. In lieu of any informational asymmetry, the approach leverages computational asymmetry by forcing the adversary to perform sizable computations to reconstruct each request. This approach does not provide ideal privacy, but makes it hard for an adversary to effectively monitor the content requests of a large number of users.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

General Terms

Design, Security

Keywords

Content-oriented Networking, Privacy

1. INTRODUCTION

Content-oriented networking improves the availability of data (by integrating caching and replication into the network) and simplifies ensuring the integrity and provenance of content (by moving from securing the communication channel to securing the content). Indeed, security has been a primary driver in many clean-slate content-oriented architectural proposals [19, 20, 29].

These benefits come with a cost. Because the basic functions of content-oriented networks operate at the content level, not the bit level, nodes in the infrastructure know the names of the content users request. An adversary that can compromise the security of this infrastructure (here we have in mind a nation-state or other formidable institution that can exert control over the infrastructure) can learn the names of the content a user requests. In contrast, in today's infrastructure, the exact nature of the data requested is only visible to the server providing that data, and the adversary would have to compromise all servers or break end-to-end cryptography, not just the network infrastructure, to learn the content being requested by a user. Thus, content-oriented networks may represent a fundamental shift in the degree of communication privacy.

Many countries already censor Internet usage, and giving them a network architecture that provides far greater access to user requests would further tighten their grip on information flow. In particular, had content-oriented networks been in use, blocking information during the uprisings in the Middle East would have been much an easier task for governments.

Our aim in this paper is tip the balance of privacy in content-oriented networks back toward the network user. We are not striving for ideal privacy (because we don't think it possible), just a significant improvement over what current content-oriented designs would give.

2. MODEL AND SCENARIO

In this section we define a realistic scenario for preserving privacy in content-oriented networks, including the capabilities of adversaries, the rules of the game, and why prior work doesn't immediately meet the need.

2.1 Basic Setting

We consider a setting in which a government (or any adversary) is trying to prevent the dissemination of *flagged* content; in particular they want to block downloads of such content and/or detect which users are asking for this content.[1] There are three parties involved: the government, the publisher of the flagged content, and the users of the flagged content. We assume that the set of users is large, and all information flow between publishers and users is in the open with no detailed coordination between publishers and individual users.

In addition, we assume that the government is interested in blocking wide dissemination of content (rather than merely trying to block its delivery to a few individuals). Moreover, we assume that the government wants to stop the content delivery in near real-time; it cannot afford to take days or weeks before detecting that flagged content has been delivered. Thus, to summarize, this is a problem of mass dissemination and real-time mass censorship.

[1]While we mainly concern ourselves with a censoring-government style of adversary, the adversary may be any party that aims to filter and monitor content for any reason.

ICN'11, August 19, 2011, Toronto, Ontario, Canada.

We stress this point because the nature of the problem renders ineffective many traditional approaches to privacy.

2.2 Attack Model

We assume all parties attach to a public network in which all content requests (fetches) and content deliveries, can be observed by the adversary. We focus on two attacks.

In a *name-watchlist attack*, the adversary has a list T of content names that it wishes to filter or eliminate. It then interposes on links in the network performing real-time filtering; if a content fetch matches against T the adversary may squelch the request and/or record the user that requested that data. In addition, the adversary may attempt to delete the data with names in this target list T. The watchlist attack can be thwarted by query and data anonymity—if it is difficult for an adversary to determine whether a fetch or a piece of stored content matches against T, then it is difficult for the adversary to effectively interfere with the dissemination of this content.

In a *content-analysis attack*, the adversary does not use a precompiled watchlist, but instead inspects the data to see if it should have been flagged (it contains the wrong keywords, etc.). This attack can be thwarted by providing plausible deniability for users (which means that they can plausibly claim that the data they received is good).

2.3 Scenario

Our goal is to prevent the two attacks above in a scenario that closely matches what we believe realistic deployments of content-oriented networks might look like. In particular, we make the following assumptions:

Neutral, large storage infrastructure. We assume that the infrastructure used to store data (we do not mean caches in the network, but the servers where original data is stored) is not controlled by the users, publishers, or the government. That is, we expect that a wide range of administrative and political domains will provide storage infrastructure with no central control. We also assume that the storage infrastructure is very large. This means that the government cannot erase content quickly (it may be able to take action against a small set of objects, but tracking down data on a disparate set of machines will be slow), and that publishers cannot enforce certain storage rules (as used in the censorship-resistant storage systems). Finally, we expect that the servers will allow a flexible, scheme-agnostic approach to naming at high level. That is, publishers are basically free to name their content as they see fit.

No secrets. We assume that users and publishers do not share any secret information that can be used to bootstrap privacy-enhanced communication. That is, adversaries will know everything users know. Besides preventing shared key cryptography, this also prevents the publishers from using secret servers from which users can download data; the government will know about these servers and can shut them down.

No key distribution. While public keys for major publishers may be widely known, we assume that user-related information (their identity or public keys) will not be widely known at the time of publication and cannot be easily distributed, thus making it difficult for content publishers to perform user-targeted content (broadcast) encryption.

No infrastructure support. We assume that the infrastructure will perform the basic functions of content-oriented networking but provides no special privacy-oriented services or mechanisms as would be needed for host-based anonymity infrastructures such as Tor.

Willing and able publishers. We assume that every flagged document has a publisher who is willing to help users preserve their anonymity. This publisher need not be the originator of the document, but is someone who is willing to publish material derived from that document (as described later). Moreover, these publishers can maintain a reasonably high rate of publication.

Limited adversarial resources. Because these attacks are in near real-time, we assume that an adversary can only bring a limited amount of resources to bear on any individual request. That is, the amount of computation devoted to analyzing the content is limited (in a content-analysis attack) and the size of the watchlist is limited.

Limited protection for individuals. Our goal is not to protect an individual from a direct assault by the adversary. If the adversary devotes a large amount of resources against an individual user, we assume it will be quite difficult to prevent them from determining if flagged content was downloaded. So, we choose an easier goal: protecting users when the resources devoted per user is limited. This won't protect a small group of dissidents, but mass movements would find protection from these measures.

2.4 Known Approaches

While the study of network privacy has a long history, to the best of our knowledge no existing approach is adequate for our needs. We'll delay the detailed analysis of the prior work until Section 5 but now briefly summarize the primary limitations of the existing systems relative to our scenario.

Systems such as Tor [13] and Freenet [11] provide anonymity for users. However, they require a sizable infrastructure (which we assume the users don't have). In addition, they do not prevent watchlist attacks: they prevent the adversary from knowing who asked for a given object, not which object was requested. Approaches such as broadcast encryption [15] and public-key steganography [31] effectively preserve the privacy of content. They do not apply to our situation because they require information to be shared between publishers and users.

We cannot use approaches such as Infranet [14] or private information retrieval [10] because they require infrastructure (or a cooperating storage). Censorship-resistant storage systems such as Freehaven [12], Tangler [32], and Dagster [28] are perhaps the most promising for our setting, though they too use specialized storage infrastructure.

For content-analysis attacks, we can't use existing approaches to provide users plausible deniability, such as repudiable information retrieval [3] or off-the-record messaging [6], since they require cooperation from particular storage nodes (that could be shut down if known by the government, and the government knows everything users know). Approaches such as deniable encryption [8] require that content publishers share secret keying information with end users.

3. DESIGN

Since both the user and the adversary share the same information in our setting, our goal is to create computational asymmetry that allows users to retrieve content efficiently but that makes it computationally expensive for the adversary to:

- Identify that the name being requested refers to flagged content. This makes name-watchlist attacks hard to mount on a large scale.
- Identify that the content retrieved should have been flagged. This makes large scale content-analysis attacks difficult.

Different aspects of our mechanism achieve these separate goals,

but they share the same overall approach which is to hide the names and content the adversary wishes to blacklist or discover by mixing the content's constituent data blocks with the blocks of normal content. A user can then fetch the content by judiciously selecting mixed data blocks to reconstruct the desired content, while the adversary is forced to perform significant computation to determine the true name of and content in those mixed blocks.

We disclaim the novelty of the basic mechanism we use to mix blocks; it bears a strong resemblance to the storage mechanisms used in [28] and [32]. The novelty of our approach is in its application to the domain of content-oriented architectures without the cooperation of the storage infrastructure. In our setting, publishers insert mixed blocks and users submit a series of requests for this mixed data, and rather than intertwine the fate of cover (normal) and target (flagged) documents (which is the goal of [28] and [32]) we instead create computational asymmetry between the user and the adversary as well as the content producer and the adversary.

3.1 Setup

Consider a "target" file t that the adversary wishes to censor to be composed of *blocks* $t_1, t_2, \ldots, t_n$ and a "cover" file c with blocks $c_1, c_2, \ldots, c_m$. The content names for all files, cover or target, are known to all parties. We assume that although the file names are not limited by a specific structure, the name of each block is direct or indirect result of hashing the file/block name/content (see Section 3.2). We also assume all blocks are of equal length and that files are padded to a multiple of the block length. These blocks are mixed together by content producers during the content publishing step; a *chunk* is the result of mixing two or more data blocks. That is, a block is the piece of the original file, and a chunk is a mixture of two or more blocks.

3.2 Chunk Creation

Before two data blocks can be mixed together we must first make the data pseudorandom (or else our construction below won't sufficiently hide document content). Conventional approaches to randomizing data involve encryption, but since content publishers and users don't share keying information, the randomizing transformation must be keyless. Fortunately, two approaches can meet our needs: the all-or-nothing transform [26] and fixed key large-width block cipher constructions [25]. We simply apply our randomizing function $r(\cdot)$ to each data block before performing any operations. These randomizing functions can be reversed by the user (and adversary) once the data has been received.[2]

Content publishers select specific "cover" content to mix with the "target" piece of content being published. The identity of these "cover files" are known to both users and the adversary. For all k-tuples composed of cover and target blocks (in any mixture, in any order), the content publisher computes the exclusive-or of the tuple and publishes the resulting chunk. For example, for $k = 2$ and given the blocks $r(t_1), r(t_2)$, and $r(c_1), r(c_2)$, the publisher would compute and publish $r(t_1) \oplus r(t_2)$, $r(t_1) \oplus r(c_1)$, $r(t_1) \oplus r(c_2)$, $r(t_2) \oplus r(c_1)$, etc. where $\oplus$ denotes exclusive-or. By fetching some purposely chosen set of these blocks (*e.g.* $r(t_i) \oplus r(c_1)$, and $r(c_1)$) the user can reconstruct the original block t_i and therefore the file t. The question is how to do this without the adversary being able to do the same easily.

[2]An all-or-nothing transform can also be applied to the whole file before being applied to the individual blocks; in this way, only after decoding all blocks of a file can an adversary perform content analysis on the data.

The name of the randomized blocks and the composite chunks are computed in a well-known way; we will assume it is of the following form (but little in our approach depends on this assumption). The name $n(t, i)$ for block t_i is $n(t, i) = H(H(t), i)$ where H is a well-known cryptographic hash function. The same applies to cover blocks c_i, taking the name $n(c, i) = H(H(c), i)$. This naming convention applies to all cover and target files; in the presentation of our notation we only referred explicitly to a single cover and target file, but our process applies to the entire set of target files and cover text cover files.

The names of composite chunks are computed by taking the hash of the names of the constituent blocks: for example, the name of the 2-tuple (t_2, c_7) is $H(n(t, 2), n(c, 7))$ which is given by $H(H(H(t), 2), H(H(c), 7))$.

All chunks that are created and published are known to all parties. That is, both users and adversaries are aware of the names of the target files, the names of the cover text cover files, and the size of the tuples used to create chunks (publishers create tuples up to a certain size). The user optimistically assumes that the content for all possible chunks (up to the given size k) is available, and upon failure to locate a chunk, notes that the chunk is unavailable and selects other chunks that can be used recreate the same block data.

This process of chunk creation can be thought of as creating a directed bipartite graph: nodes which represent chunks have edges from their constituent data blocks. However, since it is hard to invert a chunk name or the chunk data itself, it is difficult for the adversary to determine which blocks were used to compose a given chunk. We discuss this further in Section 4.

3.3 Content Retrieval

To receive a file a subscriber needs to know the content hash, its length in blocks, and its cover blocks. Secure back-channels should exist in the system that allows the user to receive meta information listing the names and algorithm to generate the names for each block. The user then starts receiving a file by requesting blocks and chunks belonging to that file. The user requests chunks by requesting them explicitly by name. Given a piece of content t that the user wishes to retrieve and the associated cover blocks with which the content's blocks are mixed, the subscriber requests chunks that will enable reassembly of the content t via belief propagation [22] or Gaussian elimination.

To provide a degree of plausible deniability, the user can select a set of chunks that enable re-creation of more than a single piece of content. In particular, the user can select chunks where the entire set recreates one or more cover files, while the composition of a particular subset recreates a target file. While there are many possible strategies a user might employ in selecting chunks, a simple approach is as follows: the user initializes a simulated reassembly session (symbolically—no real data is needed) for the data of interest, t, and randomly selects chunks that contain each of the blocks of t and of the cover data c until the simulation is capable of "decoding" both t and some data c; at this point, the user requests all the chunks it selected in a random order.

Note that the noninvertability of the names of the chunks makes it hard to execute a watchlist attack, while the need to explore all combinations of reconstruction (*i.e.* all permutations of the requested chunks) makes it hard to mount a content-analysis attack. We analyze these properties in greater detail next.

4. ANALYSIS

In this section we analyze two key aspects of our design. First,

how hard is it for the adversary to perform a privacy-threatening attack at scale? That is, what is the cost to the adversary to decode chunks observed in the network in an attempt to match the chunks to its watchlist or do a content-analysis attack? Second, even if an adversary successfully mounts an online name-watchlist attack or does post-hoc analysis after a content-analysis attack, can the adversary non-repudiably link the deciphered content to a user? We offer preliminary answers to these two questions, and discuss the implications.

4.1 Computational Asymmetry

Here we examine the computational cost for an adversary to take an unknown chunk or chunk name and determine its constituent blocks. For the adversary it is enough to be able to decompose a chunk name to its constituent block names, we call this process decoding. As described in the last section, the content publisher creates chunks by mixing blocks of k tuples of the n target and m cover blocks. Assuming that $2 \leq k \ll n \leq m$, then for each k there exists $O((n+m)^k)$ chunks, and thus the time to enumerate these chunk names in real time is $O((n+m)^k)$. Since neither a chunk's name or content can be examined to determine its contents, to decode a chunk request an adversary must compute all possible $O((n+m)^k)$ chunk names and compare them against the given chunk name until a match is found.

This decoding process from the adversary's perspective is akin to solving the subset-sum problem, in which given a set of numbers the goal is to determine whether some subset sums to a target value. Here instead of numbers, the set is composed of block names, some subset of which must combine to form a chunk name. Since we limit k, the cost does not grow to be exponential as in the general case of subset-sum, but the cost does grow rapidly as k increases or as the set of block names $(n+m)$ grows. While there exists a pseudo-polynomial dynamic programming solution for subset-sum, it cannot be directly applied here since it makes assumptions on the bound of the *values* themselves.

4.1.1 Subscriber vs. Adversary

First, we assume there exists one flagged file of interest to the adversary and several users; and all the combinations of flagged and cover blocks are published. The cost to the user in requesting and later determining the constituent blocks of a chunk is $O(1)$, since the user explicitly selects chunks based upon the blocks it desires. For the adversary, the cost depends on its resources (time and storage). The adversary can either pre-compute and store all possible chunk names and their constituent block-names, or, if it has limited resources, the cost for it to decode each chunk is $O((n+m)^k)$ at best, since for each comparison the adversary needs to calculate all possible chunk names.

If the adversary is able to pre-compute and store all possible chunk names beforehand, its cost is reduced to $O(\log{(n+m)^k})$. However, if the adversary is interested in several flagged files, it is at a significant disadvantage: it may not even know which set of target and cover blocks to consider in its decoding attack. That is, the user knows that it is seeking out a flagged file that is mixed with one or more specific cover files. While the adversary also knows this mapping between flagged files and cover files, it does not know which flagged file the user is currently requesting and therefore cannot easily limit the space of chunk names it must generate in the decoding process. Thus to be able to perform mass censorship and/or content analysis for a stream of i chunks for unknown flagged files per user that j users request per unit time, the adversary must perform $\Omega(i \cdot j \cdot \log{(n+m)^k})$ work per unit time, while each chunk request costs a user $O(1)$ work.

4.1.2 Publisher vs. Adversary

In the scenario above, while the work required of the user is constant, content publishers must produce all the chunks in advance, and thus must perform $O((n+m)^k)$ work to generate, name, and publish the chunks for a given file. While this is more work than users must perform, especially in the general case in which the adversary does not pre-compute names, the amount of work the publisher must do is likely less than the adversary since the content publisher knows exactly which flagged and cover files to consider during chunk generation. More importantly, there is no time constraint for the content publisher, since chunk generation does not need to be done in real time. However, this approach is inefficient in its use of storage.

As mentioned earlier, the cost for the adversary is reduced if it has the capability of pre-computing possible chunk names based on their constituent blocks. To give a sense for the computation resources required on the part of the adversary and content publisher, we calculate a few examples of the cost in computation. Suppose a file is composed of 500 blocks, each 1 Kb, and a publisher wants to publish these blocks in combination with 500 blocks of cover data. With k = 3 this results in 10^9 chunks of data in total to publish whereas the user needs to retrieve only enough chunks to reassemble the 500 blocks it wants, likely not much more than 10^3 chunks. Assuming a fast hash function implementation on a single CPU pre-computing all the possible chunk names is likely to take the adversary less than 10 seconds.[3] 10 seconds of pre-computation time is not a limiting factor even if the cover blocks are set to change periodically. Increasing $(n+m)$ by decreasing the size of the data blocks or increasing the number of cover files as well as increasing k results in a likely decrease in how fast the adversary can pre-compute a table of chunk names. In this case, although the costs per user remain the same, the publisher's cost increases as rapidly as the adversary's cost.

A more sophisticated way of preserving-privacy is when publisher is not forced to publish all possible combinations of flagged and cover blocks and instead publishes only a proportion of possible chunks it announces. This approach is much less resource-intensive for the publisher as it can announce a huge set of cover blocks to keep the adversary busy but not publish/match all the combinations itself.

Suppose in previous example the number of cover blocks that publisher announces were equal to 9500 blocks. For just this one flagged file adversary's cost increases to generating 10^{12} possible chunk names. Generating 10^{12} chunk names takes more than 2.5 hours for a single CPU, enough time to change the cover set by the publisher(see Section 4.2). More importantly—since the adversary can exploit parallelism—the adversary, given 512-bit hash values, would have to perform real-time matching against a 64TB table to match the blocks for just this one file. It is important to note that the cost for the publisher has not changed as it still needs to generate 10^9 chunks of data. The user will still need to receive about 10^3 chunks and thus the actual chunk reception cost remains the same but user has to ask for more chunks at a time because one chunk request does not necessarily result in a chunk response. Figure 1 depicts the relationship between computation time of chunk names, the number of data blocks, and k.

4.2 Timing and Scalability

To thwart an attack in which the adversary pre-computes a table of all possible chunks, content servers and users can consider

[3]Suppose 10^8 hash operations can be performed per second.

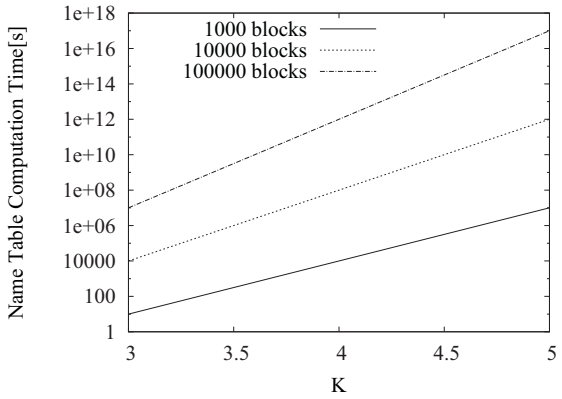

Figure 1: Having more blocks or increasing k makes the adversary's cost grow exponentially (linear in log scale) while the user's costs increase linearly.

adding a *timing dimension* to the generated chunks and requests to create scalability issues for the adversary.

Content publishers can rotate and republish the mapping between target and cover files regularly, thereby invalidating the precomputed tables. The watchlist attack could be mounted in either a passive or an active mode. In passive mode, the adversary is unable to access the user's traffic in real time, and instead can only analyze archived traffic to find a match. In active attacks, the adversary has timely access to the users traffic and can monitor it in real time.

Under a passive attack, content servers can benefit from their stateless operation vs. the adversary and the storage cost of its attack. In a name-watchlist attack, the adversary aims to prove the user has asked for the flagged file. This means that the adversary needs to keep and check each generated request against all possible chunk combinations across time. Given rotational republication, the computational and storage cost for the adversary increases to $O(t \cdot (n+m)^k)$ where t is the number of times the set of cover and target files have been rotated and changed in the system. This value does not increase costs for the content server as it only needs to index the most recent chunk combinations and update the "meta-information" for current users.

Active attacks are harder to prevent, as t does not increase the real-time matching cost for the adversary. Another aspect of the adversary's operation can be used to increase its operational complexity. To prove a watchlisted file has been retrieved by the user, the adversary needs to decode many chunks belonging to that file that been retrieved by the user. Thus, the adversary must keep a chunk retrieval log. Assuming there is only a single suspicious user at the time, the user can aim to produce extra, spurious requests. The user should be able to rotate its request pattern and even add noise to the sequence of requests that result in retrieving a file. That is, instead of requesting all the necessary chunks for an specific file in a sequence of $t_1, t_2, \ldots, t_i$ periods, the user should be free to distribute and rotate its requests over a much larger time span such as $t_1, t_2, \ldots, t_i, \ldots, t_{i+p}$. In this case, in addition to all matching operations, the adversary would require at least $O((i+p) \cdot (n+m)^k)$ storage space to be able to prove a single chunk of a watchlisted file has been retrieved by the user.

4.3 Plausible Deniability

Despite the costs borne as described above, adversaries may still decide that it's worth it to prosecute a single user or single piece of content. End-to-end protocols securing the communication channel typically perform authenticated key exchange and use random symmetric keys to encrypt and authenticate data. The encrypting party generates the key, and then it is explicitly delivered to any party that needs to decrypt the content. Since the key is exchanged before any data is transmitted, a content-analysis attack is difficult for an adversary to mount against secure communication in today's networks.

However, in a content-oriented network, an adversary may decide to perform an offline attack in which it spends the time to decode a chunk or set of chunks, or perform an online watchlist attack that targets a specific piece of content. To protect the user in that case, we aim for plausible deniability. To this end, the user can select chunks in order to enable the reassembly of many different files, including normal files.

The two goals of our design—computational asymmetry for chunk decoding and plausibly deniable requests—provide defense in depth. Since it is possible that a determined adversary will prosecute a single user or target a piece of content, it is important that even if the adversary puts the computational effort in, it isn't possible to prove a user got "target" content.

5. RELATED WORK

User, storage, and query privacy research each has a long history, encompassing numerous distinct strands. There are many possible axes along which to categorize prior work but here we consider only prior work with a primary goal of *a)* privacy, *b)* censorship resistance, or *c)* plausible deniability for users. However, we note it is typical for these systems to have more than a single goal; many of them protect users against a broad set of threats with a panoply of mechanisms. We note the literature is replete with approaches for steganographic communication, though steganographic encodings are generally domain-specific, and none to our knowledge directly apply to content-oriented networks.

Privacy. Chaum's mix-nets [9] forms the foundation for several systems providing user anonymity in public networks. Tor [13, 17] is a widely used instantiation of the technique, though many similar approaches provide specific benefits over it, including *e.g.*, Tarzan [16] (peer-to-peer overlay nodes), Freedom [33] (cover traffic), Anonymizer [2] (centralized performance). While all these approaches provide privacy of the end user's identity, they are not designed for an environment in which content itself is central to the architecture and is the target of censors.

Private Information Retrieval (PIR) and Oblivious Transfer (OT) ensure privacy of the queries themselves. In OT schemes [5, 23, 24] a client can request a piece of data from a server while ensuring that neither the server learns what was requested nor the client learns anything about data that wasn't requested. PIR schemes [7, 10, 21, 27] relax this requirement by only requiring that the server not learn what the client requested. These mechanisms require specialized server-side support and do not fit with our scenario.

Censorship Resistance. Eternity service [1], Freehaven [12], and Freenet [11] have sought to enable the persistent and anonymous publication of content via a global storage service that, through replication and oblivious storage mechanisms, make it difficult to censor. Tangler [32] and Dagster [28] take a complementary approach, aiming to "entangle" the on-disk representation of benign content with that of potentially-objectionable content, making it harder to selectively censor content [4]. Mnemosyne [18] aimed to provide an oblivious storage service via which users could store their content without revealing the content or even its presence.

Plausible Deniability. To provide deniability in storage there are

popular systems (*e.g.*, TrueCrypt [30]) using an approach known as deniable encryption [8], which allows a single, opaque ciphertext to decrypt to multiple plaintexts (presumably one of which is benign). To provide lower-cost query anonymity, researchers developed a variant of PIR called Repudiable Information Retrieval, in which a server cannot learn definitively the content requested [3]. Off-the-record messaging provides deniability for instant messaging: users can communicate securely and authentically, but their communication is repudiable after the fact [6].

6. CONCLUSION

Compared to traditional networks, content-oriented networks have the potential to expose users to a range of new privacy vulnerabilities without an obvious recourse. In this paper we introduced a technique by which users can protect their privacy despite a lack of specialized network mechanisms or shared secrets by leveraging computational asymmetry. Our analysis suggests this approach may have promise in an environment where substantial storage infrastructure is available, as we expect will be the case.

We only scratched the surface of the privacy implications of content-oriented networking, and we expect the area to require more efforts to raise the privacy bar to the level of today's Internet. Specifically, further study is needed in reader anonymity—the sort of privacy that systems like Tor provide. While there are clumsy ways of achieving such functionality in content-oriented networks even today, there is an urgent need for approaches that achieve the same degree of reader anonymity, and crucially, pseudonymity, *without* resorting to a host-oriented overlays.

7. REFERENCES

[1] R. Anderson. The Eternity Service. In *Proc. of Pragocrypt*, 1996.

[2] Anonymizer. `http://www.anonymizer.com/`.

[3] D. Asonov and J.-C. Freytag. Repudiative Information Retrieval. In *Proc. of Workshop on Privacy in the Electronic Society*, 2002.

[4] J. Aspnes, J. Feigenbaum, A. Yampolskiy, and S. Zhong. Towards a Theory of Data Entanglement. *Theoretical Computer Science*, 389(1-2):26–43, 2007.

[5] M. Bellare and S. Micali. Non-interactive Oblivious Transfer and Applications. In *Proc. of CRYPTO*, 1989.

[6] N. Borisov, I. Goldberg, and E. Brewer. Off-the-record communication, or, why not to use PGP. In *Proc. of Workshop on Privacy in the Electronic Society*, pages 77–84, 2004.

[7] C. Cachin, S. Micali, and M. Stadler. Computationally Private Information Retrieval with Polylogarithmic Communication. *Lecture Notes in Computer Science*, 1592, 1999.

[8] R. Canetti, C. Dwork, M. Naor, and R. Ostrovsky. Deniable Encryption. In *Proc. of CRYPTO*, 1997.

[9] D. L. Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[10] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private Information Retrieval. In *IEEE Symposium on FOCS*, 1995.

[11] I. Clarke, T. W. Hong, S. G. Miller, O. Sandberg, and B. Wiley. Protecting Free Expression Online with Freenet. *IEEE Internet Computing*, 6(1):40–49, 2002.

[12] R. Dingledine, M. J. Freedman, and D. Molnar. The Free Haven Project: Distributed Anonymous Storage Service. *LNCS*, 2009, 2001.

[13] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *Proc. of USENIX Security Symposium*, 2004.

[14] N. Feamster, M. Balazinska, G. Harfst, H. Balakrishnan, and D. Karger. Infranet: Circumventing Web Censorship and Surveillance. In *Proc. of USENIX Security Symposium*, 2002.

[15] A. Fiat and M. Naor. Broadcast Encryption. In *Proc. CRYPTO*, 1993.

[16] M. J. Freedman and R. Morris. Tarzan: a Peer-to-Peer Anonymizing Network Layer. In *Proc. of ACM CCS*, 2002.

[17] D. Goldschlag, M. Reed, and P. Syverson. Onion Routing. *Commun. ACM*, 42(2):39–41, 1999.

[18] S. Hand and T. Roscoe. Mnemosyne: Peer-to-Peer Steganographic Storage. In *Proc. of IPTPS*, 2002.

[19] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking Named Content. In *Proc. CoNEXT*, Dec. 2009.

[20] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A Data-Oriented (and beyond) Network Architecture. In *Proc. of SIGCOMM*, 2007.

[21] E. Kushilevitz and R. Ostrovsky. Replication Is Not Needed: Single Database, Computationally-private Information Retrieval. In *Proc. 38th IEEE FOCS*, 1997.

[22] M. Luby. LT codes. In *Proceedings of IEEE FOCS*, 2002.

[23] M. Naor and B. Pinkas. Oblivious Transfer with Adaptive Queries. In *Proc. of CRYPTO*, 1999.

[24] M. Naor and B. Pinkas. Efficient Oblivious Transfer Protocols. In *Proc. of SODA*, 2001.

[25] M. Naor and O. Reingold. On the Construction of Pseudorandom Permutations: Luby-Rackoff Revisited. *Journal of Cryptology*, 12(1):29–66, 1999.

[26] R. Rivest. All-or-nothing Encryption and the Package Transform. In *Proceedings of Fast Software Encryption*, 1997.

[27] S. W. Smith and D. Safford. Practical Private Information Retrieval with Secure Coprocessors. In *Technical report, IBM T.J. Watson Research Center*, 2000.

[28] A. Stubblefield and D. Wallach. Dagster: Censorship-Resistant Publishing Without Replication. *Rice University, Dept. of Computer Science, Tech. Rep. TR01-380*, 2001.

[29] D. Trossen, M. Särelä, and K. Sollins. Arguments for an Information-centric Internetworking Architecture. *SIGCOMM CCR*, 40, Apr. 2010.

[30] TrueCrypt. `http://www.truecrypt.org/`.

[31] L. Von Ahn and N. Hopper. Public-key Steganography. In *Proc. of EUROCRYPT*, 2004.

[32] M. Waldman and D. Mazieres. Tangler: a censorship-resistant publishing system based on document entanglements. In *Proc. of ACM CCS*, pages 126–135, 2001.

[33] Zero Knowledge Systems Freedom Network. `http://www.zks.net/`.