Vanish: Increasing Data Privacy with Self-Destructing Data

Roxana Geambasu Tadayoshi Kohno Amit A. Levy Henry M. Levy

University of Washington
{*roxana, yoshi, levya, levy*}*@cs.washington.edu*

Abstract

Today's technical and legal landscape presents formidable challenges to personal data privacy. First, our increasing reliance on Web services causes personal data to be cached, copied, and archived by third parties, often without our knowledge or control. Second, the disclosure of private data has become commonplace due to carelessness, theft, or legal actions.

Our research seeks to protect the privacy of *past, archived data* — such as copies of emails maintained by an email provider — against accidental, malicious, and legal attacks. Specifically, we wish to ensure that *all* copies of certain data become unreadable after a user-specified time, *without* any specific action on the part of a user, and even if an attacker obtains both a cached copy of that data and the user's cryptographic keys and passwords.

This paper presents *Vanish*, a system that meets this challenge through a novel integration of cryptographic techniques with global-scale, P2P, distributed hash tables (DHTs). We implemented a proof-of-concept Vanish prototype to use both the million-plus-node Vuze BitTorrent DHT and the restricted-membership OpenDHT. We evaluate experimentally and analytically the functionality, security, and performance properties of Vanish, demonstrating that it is practical to use and meets the privacy-preserving goals described above. We also describe two applications that we prototyped on Vanish: a Firefox plugin for Gmail and other Web sites and a Vanishing File application.

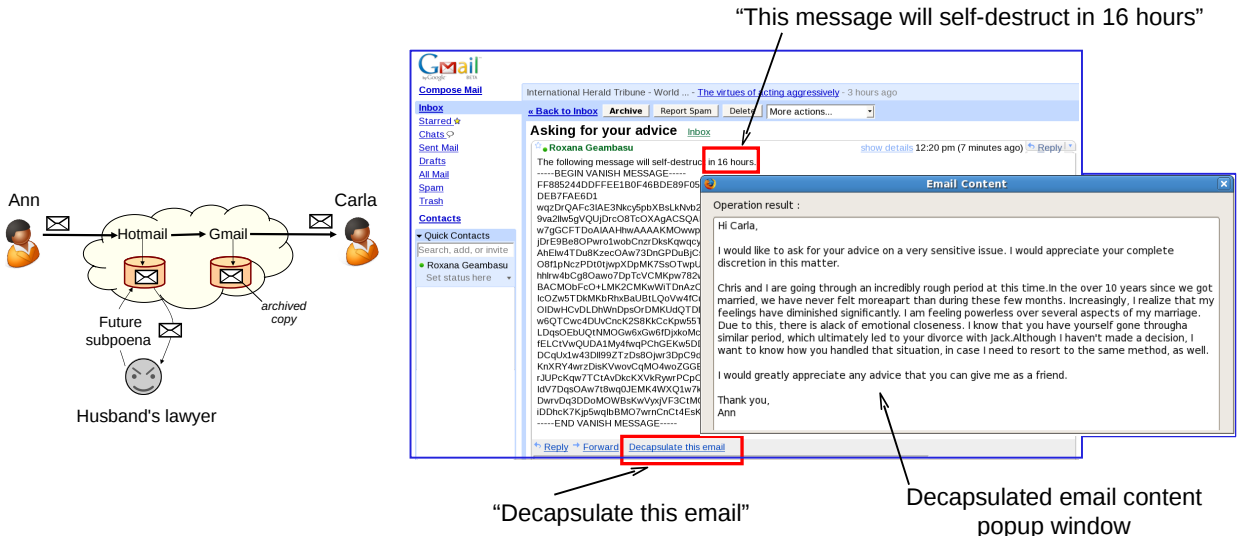
1 Introduction

We target the goal of creating data that self-destructs or vanishes *automatically* after it is no longer useful. Moreover, it should do so *without* any explicit action by the users or any party storing or archiving that data, in such a way that *all* copies of the data vanish simultaneously from all storage sites, online or offline.

Numerous applications could benefit from such self-destructing data. As one example, consider the case of email. Emails are frequently cached, stored, or archived by email providers (e.g., Gmail, or Hotmail), local backup systems, ISPs, etc. Such emails may cease to have value to the sender and receiver after a short period of time. Nevertheless, many of these emails are private, and the act of storing them indefinitely at intermediate locations creates a potential privacy risk. For example, imagine that Ann sends an email to her friend discussing a sensitive topic, such as her relationship with her husband, the possibility of a divorce, or how to ward off a spurious lawsuit (see Figure 1(a)). This email has no value as soon as her friend reads it, and Ann would like that *all* copies of this email — regardless of where stored or cached — be automatically destroyed after a certain period of time, rather than risk exposure in the future as part of a data breach, email provider mismanagement [41], or a legal action. In fact, Ann would prefer that these emails disappear early — and *not* be read by her friend — rather than risk disclosure to unintended parties. Both individuals and corporations could benefit from self-destructing emails.

More generally, self-destructing data is broadly applicable in today's Web-centered world, where users' sensitive data can persist "in the cloud" indefinitely (sometimes even after the user's account termination [61]). With self-destructing data, users can regain control over the lifetimes of their Web objects, such as private messages on Facebook, documents on Google Docs, or private photos on Flickr.

Numerous other applications could also benefit from self-destructing data. For example, while we do not condone their actions, the high-profile cases of several politicians [4, 62] highlight the relevance for self-destructing SMS and MMS text messages. The need for self-destructing text messages extends to the average user as well [42, 45]. As a news article states, "don't ever say anything on e-mail or text messaging that you don't want



(a) Example Scenario. (b) Vanishing Emails plugin for Gmail.

Figure 1: **Example Scenario and Vanish Email Screenshot.** (a) Ann wants to discuss her marital relationship with her friend, Carla, but does not want copies stored by intermediate services to be used in a potential child dispute trial in the future. (b) The screenshot shows how Carla reads a vanishing email that Ann has already sent to her using our Vanish Email Firefox plugin for Gmail.

to come back and bite you [42]." Some have even argued that the right and ability to destroy data are essential to protect fundamental societal goals like privacy and liberty [34, 44].

As yet another example, from a data sanitation perspective, many users would benefit from self-destructing trash bins on their desktops. These trash bins would preserve deleted files for a certain period of time, but after a timeout the files would self-destruct, becoming unavailable even to a forensic examiner (or anyone else, including the user). Moreover, the unavailability of these files would be guaranteed even if the forensic examiner is given a pristine copy of the hard drive from *before* the files self-destructed (e.g., because the machines were confiscated as part of a raid). Note that employing a whole disk encryption scheme is not sufficient, as the forensic examiner might be able to obtain the user's encryption passwords and associated cryptographic keys through legal means. Other time-limited temporary files, like those that Microsoft Word periodically produces in order to recover from a crash [17], could similarly benefit from self-destructing mechanisms.

Observation and Goals. A key observation in these examples is that users need to keep certain data for only a limited period of time. After that time, access to that data should be revoked for *everyone* — including the legitimate users of that data, the known or unknown entities holding copies of it, and the attackers. This mechanism will not be universally applicable to all users or data types; instead, we focus in particular on sensitive data that a user would prefer to see destroyed early rather than fall into the wrong hands.

Motivated by the above examples, as well as our observation above, we ask whether it is possible to create a system that can permanently delete data after a timeout:

1. even if an attacker can retroactively obtain a pristine copy of that data *and* any relevant persistent cryptographic keys and passphrases from *before* that timeout, perhaps from stored or archived copies;
2. without the use of any explicit delete action by the user or the parties storing that data;
3. without needing to modify any of the stored or archived copies of that data;
4. without the use of secure hardware; and
5. without relying on the introduction of any *new* external services that would need to be deployed (whether trusted or not).

A system achieving these goals would be broadly applicable in the modern digital world as we've previously noted, e.g., for files, private blog posts, on-line documents, Facebook entries, content-sharing sites, emails, messages, etc. In fact, the privacy of any digital content could potentially be enhanced with self-deleting data.

However, implementing a system that achieves this goal set is challenging. Section 2 describes many natural approaches that one might attempt and how they all fall short. In this paper we focus on a specific self-deleting data scheme that we have implemented, using email as an example application.

Our Approach. The key insight behind our approach and the corresponding system, called *Vanish*, is to leverage the services provided by decentralized, global-scale P2P infrastructures and, in particular, Distributed Hash Tables (DHTs). As the name implies, DHTs are designed

to implement a robust index-value database on a collection of P2P nodes [64]. Intuitively, Vanish encrypts a user's data locally with a random encryption key not known to the user, *destroys* the local copy of the key, and then sprinkles bits (Shamir secret shares [49]) of the key across random indices (thus random nodes) in the DHT.

Our choice of DHTs as storage systems for Vanish stems from three unique DHT properties that make them attractive for our data destruction goals. First, their huge scale (over 1 million nodes for the Vuze DHT [28]), geographical distribution of nodes across many countries, and complete decentralization make them robust to powerful and legally influential adversaries. Second, DHTs are designed to provide reliable distributed storage [35, 56, 64]; we leverage this property to ensure that the protected data remains available to the user for a desired interval of time. Last but not least, DHTs have an inherent property that we leverage in a unique and non-standard way: the fact that the DHT is constantly changing means that the sprinkled information will naturally disappear (vanish) as the DHT nodes churn or internally cleanse themselves, thereby rendering the protected data permanently unavailable over time. In fact, it may be impossible to determine retroactively which nodes were responsible for storing a given value in the past.

Implementation and Evaluation. To demonstrate the viability of our approach, we implemented a proof-of-concept Vanish prototype, which is capable of using either Bittorrent's Vuze DHT client [3] or the PlanetLab-hosted OpenDHT [54]. The Vuze-based system can support 8-hour timeouts in the basic Vanish usage model and the OpenDHT-based system can support timeouts up to one week.[1] We built two applications on top of the Vanish core — a Firefox plugin for Gmail and other Web sites, and a self-destructing file management application — and we intend to distribute all of these as open source packages in the near future. While prototyping on existing DHT infrastructures not designed for our purpose has limitations, it allows us to experiment at scale, have users benefit immediately from our Vanish applications, and allow others to build upon the Vanish core. Figure 1(b) shows how a user can decapsulate a vanishing email from her friend using our Gmail plugin (complete explanation of the interface and interactions is provided in Section 5). Our performance evaluation shows that simple, Vanish-local optimizations can support even latency-sensitive applications, such as our Gmail plugin, with acceptable user-visible execution times.

Security is critical for our system and hence we consider it in depth. Vanish targets *post-facto, retroactive attacks*; that is, it defends the user against future attacks on old, forgotten, or unreachable copies of her data. For example, consider the subpoena of Ann's email conversation with her friend in the event of a divorce. In this context, the attacker does not know what specific content to attack until *after* that content has expired. As a result the attacker's job is very difficult, since he must develop an infrastructure capable of attacking *all* users at *all* times. We leverage this observation to estimate the cost for such an attacker, which we deem too high to justify a viable threat. While we target no formal security proofs, we evaluate the security of our system both analytically and experimentally. For our experimental attacks, we leverage Amazon's EC2 cloud service to create a Vuze deployment and to emulate attacks against medium-scale DHTs.

Contributions. While the basic idea of our approach is simple conceptually, care must be taken in handling and evaluating the mechanisms employed to ensure its security, practicality, and performance. Looking ahead, and after briefly considering other tempting approaches for creating self-destructing data (Section 2), the key contributions of this work are to:

- identify the principal requirements and goals for self-destructing data (Section 3);
- propose a novel method for achieving these goals that combines cryptography with decentralized, global-scale DHTs (Section 4);
- demonstrate that our prototype system and applications are deployable today using existing DHTs, while achieving acceptable performance, and examine the tensions between security and availability for such deployments (Section 5);
- experimentally and analytically evaluate the privacy-preservation capabilities of our DHT-based system (Section 6).

Together, these contributions provide the foundation for empowering users with greater control over the lifetimes of private data scattered across the Internet.

2 Candidate Approaches

A number of existing and seemingly natural approaches may appear applicable to achieving our objectives. Upon deeper investigation, however, we find that none of these approaches are sufficient to achieve the goals enumerated in Section 1. We consider these strawman approaches here and use them to further motivate our design constraints in Section 3.

The most obvious approach would require users to explicitly and manually delete their data or install a `cron` job to do that. However, because Web-mails and other Web data are stored, cached, or backed up at numerous places throughout the Internet or on Web servers,

[1] We have an external mechanism to extend Vuze timeouts beyond 8 hours, which we describe later.

this approach does not seem plausible. Even for a self-destructing trash bin, requiring the user to explicitly delete data is incompatible with our goals. For example, suppose that the hard disk fails and is returned for repairs or thrown out [15]; or imagine that a laptop is stolen and the thief uses a cold-boot [32] attack to recover its primary whole-disk decryption keys (if any). We wish to ensure data destruction even in cases such as these.

Another tempting approach might be to use a standard public key or symmetric encryption scheme, as provided by systems like PGP and its open source counterpart, GPG. However, traditional encryption schemes are insufficient for our goals, as they are designed to protect against adversaries without access to the decryption keys. Under our model, though, we assume that the attacker will be able to obtain access to the decryption keys, e.g., through a court order or subpoena.[2]

A potential alternative to standard encryption might be to use forward-secure encryption [6, 13], yet our goal is strictly stronger than forward secrecy. Forward secrecy means that if an attacker learns the state of the user's cryptographic keys at some point in time, they should not be able to decrypt data encrypted at an earlier time. However, due to caching, backup archives, and the threat of subpoenas or other court orders, we allow the attacker to either view past cryptographic state or force the user to decrypt his data, thereby violating the model for forward-secure encryption. For similar reasons, plus our desire to avoid introducing new trusted agents or secure hardware, we do not use other cryptographic approaches like key-insulated [5, 23] and intrusion-resilient [21, 22] cryptography. Finally, while exposure-resilient cryptography [11, 24, 25] allows an attacker to view parts of a key, we must allow an attacker to view all of the key.

Another approach might be to use steganography [48], deniable encryption [12], or a deniable file system [17]. The idea is that one could hide, deny the contents of, or deny the existence of private historical data, rather than destroying it. These approaches are also attractive but hard to scale and automate for many applications, e.g., generating plausible cover texts for emails and photos. In addition to the problems observed with deniable file systems in [17] and [38], deniable file systems would also create additional user hassles for a trash bin application, whereas our approach could be made invisible to the user.

For online, interactive communications systems, an ephemeral key exchange process can protect derived symmetric keys from future disclosures of asymmetric private keys. A system like OTR [1, 10] is particularly attractive, but as the original OTR paper observes, this approach is not directly suited for less-interactive email applications, and similar arguments can be made for OTR's unsuitability for the other above-mentioned applications as well.

An approach with goals similar to ours (except for the goal of allowing users to create self-destructing objects without having to establish asymmetric keys or passphrases) is the Ephemerizer family of solutions [39, 46, 47]. These approaches require the introduction of one or more (possibly thresholded) trusted third parties which (informally) escrow information necessary to access the protected contents. These third parties destroy this extra data after a specified timeout. The biggest risks with such centralized solutions are that they may either not be trustworthy, or that even if they are trustworthy, users may still not trust them, hence limiting their adoption. Indeed, many users may be wary to the use of dedicated, centralized trusted third-party services after it was revealed that the Hushmail email encryption service was offering the cleartext contents of encrypted messages to the federal government [59]. This challenge calls for a decentralized approach with fewer real risks *and* perceived risks.

A second lesson can be learned from the Ephemerizer solutions in that, despite their introduction several years ago, these approaches have yet to see widespread adoption. This may in part be due to the perceived trust issues mentioned above, but an additional issue is that these solutions require the creation of new, supported and maintained services. We theorize that solutions that require *new* infrastructures have a greater barrier to adoption than solutions that can "parasitically" leverage *existing* infrastructures. A variant of this observation leads us to pursue approaches that do not require secure hardware or other dedicated services.

3 Goals and Assumptions

To support our target applications (self-destructing emails, Facebook messages, text messages, trash bins, etc.), we introduce the notion of a *vanishing data object* (*VDO*). A VDO encapsulates the user's data (such as a file or message) and prevents its contents from persisting indefinitely and becoming a source of retroactive information leakage. Regardless of whether the VDO is copied, transmitted, or stored in the Internet, it becomes unreadable after a predefined period of time even if an attacker *retroactively* obtains both a *pristine* copy of the VDO from before its expiration, and all of the user's past persistent cryptographic keys and passwords. Figure 2 illustrates the above properties of VDOs by showing the timeline for a typical usage of and attack against a VDO. We crystallize the assumptions underlying our

[2] U.S. courts are debating whether citizens are required to disclose private keys, although the ultimate verdict is unclear. We thus target technologies robust against a verdict in either direction [29, 40]. Other countries such as the U.K. [43] require release of keys, and coercion or force may be an issue in yet other countries.

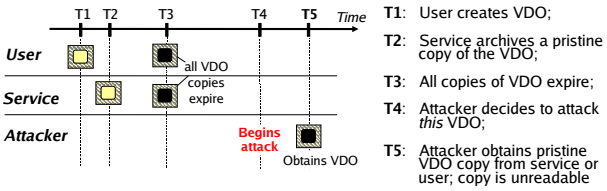


Figure 2: **Timeline for VDO usage and attack.**

VDO model and the central aspects of the threat model below.

Assumptions. Our VDO abstraction and Vanish system make several key assumptions:

1. *Time-limited value.* The VDO will be used to encapsulate data that is only of value to the user for a limited period of time.
2. *Known timeout.* When a user encapsulates data in a VDO, she knows the approximate lifetime that she wants for her VDO.
3. *Internet connectivity.* Users are connected to the Internet when interacting with VDOs.
4. *Dispensability under attack.* Rather than risk exposure to an adversary, the user prefers the VDO to be destroyed, even if prematurely.

We consider encapsulation of data that only needs to be available for hours or days; *e.g.*, certain emails, Web objects, SMSs, trash bin files, and others fall into this category. Internet connectivity is obviously required for many of our applications already, such as sending and receiving emails. More generally, the promise of ubiquitous connectivity makes this assumption reasonable for many other applications as well. Internet connectivity is not required for deletion, i.e., a VDO will become unreadable even if connectivity is removed from its storage site (or if that storage site is offline). Finally, Vanish is designed for use with data that is private, but whose persistence is not critical. That is, while the user prefers that the data remain accessible until the specified timeout, its premature destruction is preferable to its disclosure.

Goals. Having stated these assumptions, we target the following functional goals and properties for Vanish:

1. *Destruction after timeout.* A VDO must expire automatically and *without* any explicit action on the part of its users or any party storing a copy of the VDO. Once expired, the VDO must also be inaccessible to any party who obtains a *pristine* copy of the VDO from *prior* to its expiration.
2. *Accessible until timeout.* During its lifetime, a VDO's contents should be available to legitimate users.
3. *Leverage existing infrastructures.* The system must leverage existing infrastructures. It must not rely on external, special-purpose dedicated services.
4. *No secure hardware.* The system must not require the use of dedicated secure hardware.
5. *No new privacy risks.* The system should not introduce new privacy risks to the users.

A corollary of goal (1) is that the VDO will become unavailable to the legitimate users after the timeout, which is compatible with our applications and assumption of time-limited value.

Our desire to leverage existing infrastructure (goal (3)) stems from our belief that special-purpose services may hinder adoption. As noted previously, Hushmail's disclosure of the contents of users' encrypted emails to the federal government [59] suggests that, even if the centralized service or a threshold subset of a collection of centralized services is trustworthy, users may still be unwilling to trust them.

As an example of goal (5), assume that Ann sends Carla an email *without* using Vanish, and then another email using Vanish. If an attacker cannot compromise the privacy of the first email, then we require that the same attacker — regardless of how powerful — cannot compromise the privacy of the second email.

In addition to these goals, we also seek to keep the VDO abstraction as generic as possible. In Vanish, the process of encapsulating data in a VDO does *not* require users to set or remember passwords or manage cryptographic keys. However, to ensure privacy under stronger threat models, Vanish applications may compose VDOs with traditional encryption systems like PGP and GPG. In this case, the user will naturally need to manipulate the PGP/GPG keys and passphrases.

Threat Models. The above list enumerates the intended properties of the system *without* the presence of an adversary. We now consider the various classes of potential adversaries against the Vanish system, as well as the desired behavior of our system in the presence of such adversaries.

The central security goal of Vanish is to ensure the destruction of data after a timeout, despite potential adversaries who might attempt to access that data after its timeout. Obviously, care must be taken in defining what a plausible adversary is, and we do that below and in Section 6. But we also stress that we explicitly do *not* seek to preserve goal (2) — accessible prior to a timeout — in the presence of adversaries. As previously noted, we believe that users would prefer to sacrifice availability pre-timeout in favor of assured destruction for the types of data we are protecting. For example, we do not defend against denial of service attacks that could prevent reading of the data during its lifetime. Making this assumption allows us to focus on the primary novel insights in this work: methods for leveraging decentralized, large-scale P2P networks in order to make data vanish over time.

We therefore focus our threat model and subsequent analyses on attackers who wish to compromise data privacy. Two key properties of our threat model are:

1. *Trusted data owners.* Users with legitimate access to the same VDOs trust each other.
2. *Retroactive attacks on privacy.* Attackers do not know which VDOs they wish to access until *after* the VDOs expire.

The former aspect of the threat model is straightforward, and in fact is a shared assumption with traditional encryption schemes: it would be impossible for our system to protect against a user who chooses to leak or permanently preserve the cleartext contents of a VDO-encapsulated file through out-of-band means. For example, if Ann sends Carla a VDO-encapsulated email, Ann must trust Carla not to print and store a hard-copy of the email in cleartext.

The latter aspect of the threat model — that the attacker does not know the identity of a specific VDO of interest until *after* its expiration — was discussed briefly in Section 1. For example, email or SMS subpoenas typically come long after the user sends a particular sensitive email. Therefore, our system defends the user against *future attacks against old copies of private data*.

Given the retroactive restriction, an adversary would have to do some precomputation prior to the VDO's expiration. The precise form of precomputation will depend on the adversary in question. The classes of adversaries we consider include: the user's employer, the user's ISP, the user's web mail provider, and unrelated malicious nodes on the Internet. For example, foreshadowing to Section 6, we consider an ISP that might spy on the connections a user makes to the Vuze DHT on the off chance that the ISP will later be asked to assist in the retroactive decapsulation of the user's VDO. Similarly, we consider the potential for an email service to proactively try to violate the privacy of VDOs prior to expiration, for the same reason. Although we deem both situations unlikely because of public perception issues and lack of incentives, respectively, we can also provide defenses against such adversaries.

Finally, we stress that we do not seek to provide privacy against an adversary who gets a warrant to intercept *future* emails. Indeed, such an attacker would have an arsenal of attack vectors at his disposal, including not only *a priori* access to sensitive emails but also keyloggers and other forensic tools [37].

4 The Vanish Architecture

We designed and implemented Vanish, a system capable of satisfying all of the goals listed in Section 3. A key contribution of our work is to leverage existing, decentralized, large-scale Distributed Hash Tables (DHTs). After providing a brief overview of DHTs and introducing the insights that underlie our solution, we present our system's architecture and components.

Overview of DHTs. A DHT is a distributed, peer-to-peer (P2P) storage network consisting of multiple participating *nodes* [35, 56, 64]. The design of DHTs varies, but DHTs like Vuze generally exhibit a put/get interface for reading and storing data, which is implemented internally by three operations: `lookup`, `get`, and `store`. The data itself consists of an $(index, value)$ pair. Each node in the DHT manages a part of an astronomically large index name space (e.g., 2^{160} values for Vuze). To store data, a client first performs a `lookup` to determine the nodes responsible for the index; it then issues a `store` to the responsible node, who saves that $(index, value)$ pair in its local DHT database. To retrieve the value at a particular index, the client would `lookup` the nodes responsible for the index and then issue `get` requests to those nodes. Internally, a DHT may replicate data on multiple nodes to increase availability.

Numerous DHTs exist in the Internet, including Vuze, Mainline, and KAD. These DHTs are *communal*, *i.e.*, any client can join, although DHTs such as OpenDHT [54] only allow authorized nodes to join.

Key DHT-related Insights. Three key properties of DHTs make them extremely appealing for use in the context of a self-destructing data system:

1. *Availability.* Years of research in availability in DHTs have resulted in relatively robust properties of today's systems, which typically provide good availability of data prior to a specific timeout. Timeouts vary, e.g., Vuze has a fixed 8-hour timeout, while OpenDHT allows clients to choose a per-data-item timeout of up to one week.
2. *Scale, geographic distribution, and decentralization.* Measurement studies of the Vuze and Mainline DHTs estimate in excess of one million simultaneously active nodes in each of the two networks [28]. The data in [63] shows that while the U.S. is the largest single contributor of nodes in Vuze, a majority of the nodes lie outside the U.S. and are distributed over 190 countries.
3. *Churn.* DHTs evolve naturally and dynamically over time as new nodes constantly join and old nodes leave. The average lifetime of a node in the DHT varies across networks and has been measured from minutes on Kazaa [30] to hours on Vuze/Azureus [28].

The first property provides us with solid grounds for implementing a useful system. The second property makes DHTs more resilient to certain types of attacks than centralized or small-scale systems. For example,

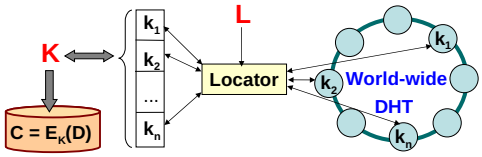


Figure 3: **The Vanish system architecture.**

while a centrally administered system can be compelled to release data by an attacker with legal leverage [59], obtaining subpoenas for multiple nodes storing a VDO's key pieces would be significantly harder, and in some cases impossible, due to their distribution under different administrative and political domains.

Traditionally, DHT research has tried to counter the negative effects of churn on availability. For our purposes, however, the constant churn in the DHT is an advantage, because it means that data stored in DHTs will naturally and irreversibly disappear over time as the DHT evolves. In many cases, trying to determine the contents of the DHT one week in the past — let alone several months or years — may be impossible, because many of the nodes storing DHT data will have left or changed their locations in the index space. For example, in Vuze, a node changes its location in the DHT whenever its IP address or port number changes, which typically happens periodically for dynamic IP addresses (*e.g.*, studies show that over 80% of the IPs change within 7 days [65]). This self-cleansing property of DHTs, coupled with its scale and global decentralization, makes them a felicitous choice for our self-destructing data system.

Vanish. Vanish is designed to leverage one or more DHTs. Figure 3 illustrates the high-level system architecture. At its core, Vanish takes a data object D (and possibly an explicit timeout T), and encapsulates it into a VDO V.

In more detail, to encapsulate the data D, Vanish picks a random data key, K, and encrypts D with K to obtain a ciphertext C. Not surprisingly, Vanish uses threshold secret sharing [58] to split the data key K into N pieces (*shares*) $K_1,\ldots,K_N$. A parameter of the secret sharing is a *threshold* that can be set by the user or by an application using Vanish. The threshold determines how many of the N shares are required to reconstruct the original key. For example, if we split the key into $N = 20$ shares and the threshold is 10 keys, then we can compute the key given any 10 of the 20 shares. In this paper we often refer to the *threshold ratio* (or simply threshold) as the percentage of the N keys required, e.g., in the example above the threshold ratio is 50%.

Once Vanish has computed the key shares, it picks at random an *access key*, L. It then uses a cryptographically secure pseudorandom number generator [7], keyed by L, to derive N indices into the DHT, $I_1,\ldots,I_N$. Vanish then sprinkles the N shares $K_1,\ldots,K_N$ at these pseudorandom locations throughout the DHT; specifically, for each $i \in \{1,\ldots,N\}$, Vanish stores the share K_i at index I_i in the DHT. If the DHT allows a variable timeout, e.g., with OpenDHT, Vanish will also set the user-chosen timeout T for each share. Once more than $(N - threshold)$ shares are lost, the VDO becomes permanently unavailable.

The final VDO V consists of $(L,C,N,threshold)$ and is sent over to the email server or stored in the file system upon encapsulation. The decapsulation of V happens in the natural way, assuming that it has not timed out. Given VDO V, Vanish (1) extracts the access key, L, (2) derives the locations of the shares of K, (3) retrieves the required number of shares as specified by the threshold, (4) reconstructs K, and (5) decrypts C to obtain D.

Threshold Secret Sharing, Security, and Robustness. For security we rely on the property that the shares $K_1,\ldots,K_N$ will disappear from the DHT over time, thereby limiting a retroactive adversary's ability to obtain a sufficient number of shares, which must be $\geq$ the threshold ratio. In general, we use a ratio of $< 100\%$, otherwise the loss of a single share would cause the loss of the key. DHTs do lose data due to churn, and therefore a smaller ratio is needed to provide robust storage prior to the timeout. We consider all of these issues in more detail later; despite the conceptual simplicity of our approach, significant care and experimental analyses must be taken to assess the durability of our use of large-scale, decentralized DHTs.

Extending the Lifetime of a VDO. For certain uses, the default timeout offered by Vuze might be too limiting. For such cases, Vanish provides a mechanism to refresh VDO shares in the DHT. While it may be tempting at first to simply use Vuze's republishing mechanism for index-value pairs, doing so would re-push the same pairs $(I_1,K_1),\ldots,(I_N,K_N)$ periodically, until the timeout. This would, in effect, increase the exposure of those key shares to certain attackers. Hence, our refresh mechanism retrieves the original data key K before its timeout, re-splits it, obtaining a fresh set of shares, and derives new DHT indices $I_1,\ldots,I_N$ as a function of L and a weakly synchronized clock. The weakly synchronized clock splits UTC time into roughly 8-hour epochs and uses the epoch number as part of the input to the location function. Decapsulations then query locations generated from both the current epoch number and the neighboring epochs, thus allowing clocks to be weakly synchronized.

Naturally, refreshes require periodic Internet connectivity. A simple home-based setup, where a broadband connected PC serves as the user's refreshing proxy, is in our view and experience a very reasonable choice given today's highly connected, highly equipped homes. In fact, we have been using this setup in our in-house deployment of Vanish in order to achieve longer timeouts for our emails (see Section 5).

Using multiple or no DHTs. As an extension to the scheme above, it is possible to store the shares of the data key K in *multiple* DHTs. For example, one might first split K into two shares K' and K'' such that both shares are required to reconstruct K. K' is then split into N' shares and sprinkled in the Vuze DHT, while K'' is split into N'' shares and sprinkled in OpenDHT. Such an approach would allow us to argue about security under different threat models, using OpenDHT's closed access (albeit small scale) and Vuze's large scale (albeit communal) access.

An alternate model would be to abandon DHTs and to store the key shares on distributed but managed nodes. This approach bears limitations similar to Ephemerizer (Section 2). A hybrid approach might be to store shares of K' in a DHT and shares of K'' on managed nodes. This way, an attacker would have to subvert both the privately managed system *and* the DHT to compromise Vanish.

Forensic Trails. Although not a common feature in today's DHTs, a future DHT or managed storage system could additionally provide a forensic trail for monitoring accesses to protected content. A custom DHT could, for example, record the IP addresses of the clients that query for particular indices and make that information available to the originator of that content. The existence of such a forensic trail, even if probabilistic, could dissuade third parties from accessing the contents of VDOs that they obtain prior to timeout.

Composition. Our system is not designed to protect against all attacks, especially those for which solutions are already known. Rather, we designed both the system and our applications to be composable with other systems to support defense-in-depth. For example, our Vanish Gmail plugin can be composed with GPG in order to avoid VDO sniffing by malicious email services. Similarly, our system can compose with Tor to ensure anonymity and throttle targeted attacks.

5 Prototype System and Applications

We have implemented a Vanish prototype capable of integrating with both Vuze and OpenDHT. In this section, we demonstrate that (1) by leveraging existing, unmodified DHT deployments we can indeed achieve the core functions of vanishing data, (2) the resulting system supports a variety of applications, and (3) the performance of VDO operations is reasonable. We focus our discussions on Vuze because its large scale and dynamic nature make its analysis both more interesting and more challenging. A key observation derived from our study is a tension in setting VDO parameters (N and threshold) when targeting both high availability prior to the timeout and high security. We return to this tension in Section 6.

To integrate Vanish with the Vuze DHT, we made two minor changes ($<$ 50 lines of code) to the existing Vuze BitTorrent client: a security measure to prevent `lookup` sniffing attacks (see Section 6.2) and several optimizations suggested by prior work [28] to achieve reasonable performance for our applications. All these changes are *local* to Vanish nodes and do not require adoption by any other nodes in the Vuze DHT.

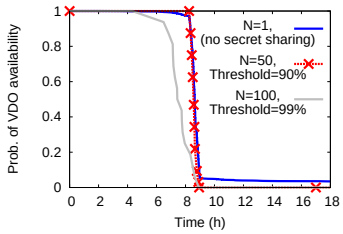


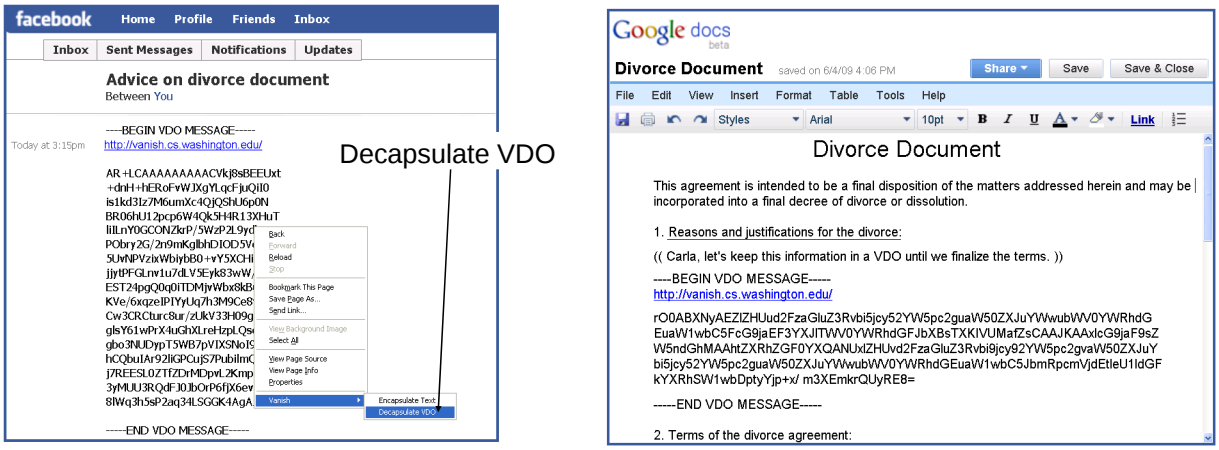
Figure 4: **VDO availability in the Vuze-based Vanish system.** The availability probability for single-key VDOs ($N = 1$) and for VDOs using secret sharing, averaged over 100 runs. Secret sharing is required to ensure pre-timeout availability and post-timeout destruction. Using $N = 50$ and a threshold of 90% achieves these goals.

5.1 Vuze Background

The Vuze (*a.k.a.* Azureus) DHT is based on the Kademlia [35] protocol. Each DHT node is assigned a "random" 160-bit ID based on its IP and port, which determines the index ranges that it will store. To store an $(\text{index}, \text{value})$ pair in the DHT, a client looks up 20 nodes with IDs closest to the specified index and then sends `store` messages to them. Vuze nodes republish the entries in their cache database every 30 minutes to the other 19 nodes closest to the value's index in order to combat churn in the DHT. Nodes further *remove* from their caches all values whose `store` timestamp is more than 8 hours old. This process has a 1-hour grace period. The originator node must re-push its 8-hour-old $(\text{index}, \text{value})$ pairs if it wishes to ensure their persistence past 8 hours.

5.2 VDO Availability and Expiration in Vuze

We ran experiments against the real global Vuze P2P network and evaluated the availability and expiration guarantees it provides. Our experiments pushed 1,000 VDO shares to pseudorandom indices in the Vuze DHT and then polled for them periodically. We repeated this experiment 100 times over a 3-day period in January 2009. Figure 4 shows the average probability that a VDO remains available as a function of the time since creation, for three different N and threshold values. For these experiments we used the standard 8-hour Vuze timeout (i.e., we did not use our refreshing proxy to re-push shares).



(a) Vanishing Facebook messages.

(b) Google Doc with vanishing parts.

Figure 5: **The Web-wide applicability of Vanish.** Screenshots of two example uses of vanishing data objects on the Web. (a) Carla is attempting to decapsulate a VDO she received from Ann in a Facebook message. (b) Ann and Carla are drafting Ann's divorce document using a Google Doc; they encapsulate sensitive, draft information inside VDOs until they finalize their position.

The $N = 1$ line shows the lifetime for a single share, which by definition does not involve secret sharing. The single-share VDO exhibits two problems: non-negligible probabilities for premature destruction ($\approx$1% of the VDOs time out before 8 hours) and prolonged availability ($\approx$5% of the VDOs continue to live long after 8 hours). The cause for the former effect is churn, which leads to early loss of the unique key for some VDOs. While the cause for the latter effect demands more investigation, we suspect that some of the single VDO keys are stored by DHT peers running non-default configurations. These observations suggest that the naive (one share) approach for storing the data key K in the DHT meets neither the availability nor the destruction goals of VDOs, thereby motivating our need for redundancy.

Secret sharing can solve the two lifetime problems seen with $N = 1$. Figure 4 shows that for VDOs with $N = 50$ and threshold of 90%, the probability of premature destruction and prolonged availability both become vanishingly small ($< 10^{-3}$). Other values for $N \geq 20$ achieve the same effect for thresholds of 90%. However, using very high threshold ratios leads to poor pre-timeout availability curves: e.g., $N = 100$ and a threshold of 99% leads to a VDO availability period of 4 hours because the loss of only two shares share makes the key unrecoverable. We will show in Section 6 that increasing the threshold increases security. Therefore, the choice of N and the threshold represents a tradeoff between security and availability. We will investigate this tradeoff further in Section 6.

5.3 Vanish Applications

We built two prototype applications that use a Vanish daemon running locally or remotely to ensure self-destruction of various types of data.

FireVanish. We implemented a Firefox plugin for the popular Gmail service that provides the option of sending and reading self-destructing emails. Our implementation requires no server-side changes. The plugin uses the Vanish daemon both to transform an email into a VDO before sending it to Gmail and similarly for extracting the contents of a VDO on the receiver side.

Our plugin is implemented as an extension of FireGPG (an existing GPG plugin for Gmail) and adds Vanish-related browser overlay controls and functions. Using our FireVanish plugin, a user types the body of her email into the Gmail text box as usual and then clicks on a *"Create a Vanishing Email"* button that the plugin overlays atop the Gmail interface. The plugin encapsulates the user's typed email body into a VDO by issuing a VDO-create request to Vanish, replaces the contents of the Gmail text box with an encoding of the VDO, and uploads the VDO email to Gmail for delivery. The user can optionally wrap the VDO in GPG for increased protection against malicious services. In our current implementation, each email is encapsulated with its own VDO, though a multi-email wrapping would also be possible (*e.g.*, all emails in the same thread).

When the receiving user clicks on one of his emails, FireVanish inspects whether it is a VDO email, a PGP email, or a regular email. Regular emails require no further action. PGP emails are first decrypted and then inspected to determine whether the underlying message is a VDO email. For VDO emails, the plugin overlays a link "*Decapsulate this email*" atop Gmail's regular interface (shown previously in Figure 1(b)). Clicking on this link causes the plugin to invoke Vanish to attempt to retrieve the cleartext body from the VDO email. If the VDO has not yet timed out, then the plugin pops up a new window showing the email's cleartext body; otherwise, an error message is displayed.

FireVanish Extension for the Web. Self-destructing data is broadly applicable in today's Web-oriented world, in which users often leave permanent traces on many Web sites [61]. Given the opportunity, many privacy-concerned users would likely prefer that certain messages on Facebook, documents on Google Docs, or instant messages on Google Talk disappear within a short period of time.

To make Vanish broadly accessible for Web usage, FireVanish provides a simple, generic, yet powerful, interface that permits all of these applications. Once the FireVanish plugin has been installed, a Firefox user can select text in any Web page input box, right click on that selected text, and cause FireVanish to replace that text *inline* with an ecapsulated VDO. Similarly, when reading a Web page containing a VDO, a user can select that VDO and right click to decapsulate it; in this case, FireVanish leaves the VDO in place and displays the cleartext in a separate popup window.

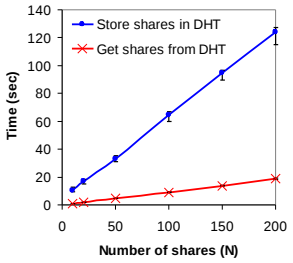
Figure 5 shows two uses of FireVanish to encapsulate and read VDOs within Facebook and Google Docs. The screenshots demonstrate a powerful concept: FireVanish can be used seamlessly to empower privacy-aware users with the ability to limit the lifetime of their data on Web applications that are unaware of Vanish.

Vanishing Files. Finally, we have implemented a vanishing file application, which can be used directly or by other applications, such as a self-destructing trash bin or Microsoft Word's autosave. Users can wrap sensitive files into self-destructing VDOs, which expire after a given timeout. In our prototype, the application creates a VDO wrapping one or more files, deletes the cleartext files from disk, and stores the VDO in their place. This ensures that, even if an attacker copies the raw bits from the laptop's disks after the timeout, the data within the VDO will be unavailable. Like traditional file encryption, Vanishing Files relies upon existing techniques for securely shredding data stored on disks or memory.

5.4 Performance Evaluation

We measured the performance of Vanish for our applications, focusing on the times to encapsulate and decapsulate a VDO. Our goals were to (1) identify the system's performance bottlenecks and propose optimizations, and (2) determine whether our Vuze-based prototype is fast enough for our intended uses. Our measurements use an Intel T2500 DUO with 2GB of RAM, Java 1.6, and a broadband network.

To identify system bottlenecks, we executed VDO operations and measured the times spent in the three main runtime components: DHT operations (storing and getting shares), Shamir secret sharing operations (splitting/recomposing the data key), and encryption/decryption. In general, the DHT component accounts for over 99% of the execution time for all Vanish operations on small and medium-size data (up to tens of MB, like most emails). For much larger data sizes (*e.g.*, files over hundreds of MB), the encryption/decryption becomes the dominating component.



(a) Scalability of DHT operations.

N	Time (seconds)		
	Encapsulate VDO		Decapsulate VDO
	Without prepush	With prepush	
10	10.5	0.082	0.9
20	16.9	0.082	2.0
50	32.8	0.082	4.7
100	64.5	0.082	9.2
150	94.7	0.082	14.0
200	124.3	0.082	19.0

(b) VDO operation execution times.

Figure 6: **Performance in the Vuze-based Vanish system.** (a) The scalability of DHT operation times as a function of the number of shares being gotten from or stored in the DHT (results are averages over 20 trials and error bars indicate standard deviations). (b) Total VDO encapsulation (with and without pre-push) and decapsulation times for FireVanish for a 2KB email, $N = 50$, and threshold 90%.

Our experiments also revealed the importance of configuring Vuze's parameters on our latency-aware applications. With no special tuning, Vuze took 4 minutes to store 50 shares, even using parallel stores. By employing several Vuze optimizations we lowered the 50-share store time by a factor of 7 (to 32 seconds). Our most effective optimization — significantly lowering Vuze's UDP timeout based on suggestions from previous research [28] — proved non-trivial, though. In particular, as we deployed Vanish within our group, we learned that different Internet providers (*e.g.*, Qwest, Comcast) exhibited utterly different network behaviors and latencies, making the setting of any one efficient value for the timeout impossible. Hence, we implemented a control-loop-based mechanism by which Vanish automatically configures Vuze's UDP timeout based on current network conditions. The optimization requires only node-local changes to Vuze.

Figure 6(a) shows how the optimized DHT operation times scale with the number of shares (N), for a fixed threshold of 90%, over a broadband connection (Comcast). Scaling with N is important in Vanish, as its se-

curity is highly dependent on this parameter. The graph shows that getting DHT shares are relatively fast — under 5 seconds for $N = 50$, which is reasonable for emails, trash bins, etc. The cost of storing VDO shares, however, can become quite large (about 30 seconds for $N = 50$), although it grows liniarly with the number of shares. To mask the store delays from the user, we implemented a simple optimization, where Vanish proactively generates data keys and pre-pushes shares into the DHT. This optimization leads to an unnoticeable DHT encapsulation time of $82ms$.

Combining the results in this section and Section 6, we believe that parameters of $N = 50$ and a threshold of 90% provide an excellent tradeoff of security and performance. With these parameters and the simple pre-push optimization we've described, user-visible latency for Vanish operations, such as creating or reading a Vanish email, is relatively low — just a few seconds for a 2KB email, as shown in Figure 6(b).

5.5 Anecdotal Experience with FireVanish

We have been using the FireVanish plugin within our group for several weeks. We also provided Vanish to several people outside of our group. Our preliminary experience has confirmed the practicality and convenience of FireVanish. We also learned a number of lessons even in this short period; for example, we found our minimalistic interface to be relatively intuitive, even for a non-CS user to whom we gave the system, and the performance is quite acceptable, as we noted above.

We also identified several limitations in the current implementation, some that we solved and others that we will address in the future. For example, in the beginning we found it difficult to search for encrypted emails or data, since their content is encrypted and opaque to the Web site. For convenience, we modified FireVanish to allow users to construct emails or other data by mixing together non-sensitive cleartext blocks with self-destructing VDOs, as illustrated in Figure 5(b). This facilitates identifying information over and above the subject line. We did find that certain types of communications indeed require timeouts longer than 8 hours. Hence, we developed and used Vanish in a proxy setting, where a Vanish server runs on behalf of a user at an online location (e.g., the user's home) and refreshes VDO shares as required to achieve each VDO's intended timeout in 8-hour units. The user can then freely execute the Vanish plugin from any connection-intermittent location (*e.g.*, a laptop).

We are planning an open-source release of the software in the near future and are confident that this release will teach us significantly more about the usability, limitations, and security of our system.

6 Security Analyses

To evaluate the security of Vanish, we seek to assess two key properties: that (1) Vanish does not introduce any *new* threats to privacy (goal (5) in Section 3), and (2) Vanish is secure against adversaries attempting to retroactively read a VDO post-expiration.

It is straightforward to see that Vanish adds no new privacy risks. In particular, the key shares stored in the DHT are *not* a function of the encapsulated data D; only the VDO is a function of D. Hence, if an adversary is unable to learn D when the user does not use Vanish, then the adversary would be unable to learn D if the user does use Vanish. There are three caveats, however. First, external parties, like the DHT, might infer information about who is communicating with whom (although the use of an anonymization system like Tor can alleviate this concern). Second, given the properties of Vanish, users might choose to communicate information that they might not communicate otherwise, thus amplifying the consequences of any successful data breach. Third, the use of Vanish might raise new legal implications. In particular, the new "eDiscovery" rules embraced by the U.S. may require a user to preserve emails and other data once in anticipation of a litigious action. The exact legal implications to Vanish are unclear; the user might need to decapsulate and save any relevant VDOs to prevent them from automatic expiration.

We focus the remainder of this section on attacks targeted at retroactively revoking the privacy of data encapsulated within VDOs (this attack timeline was shown in Figure 2). We start with a broad treatment of such attacks and then dive deeply into attacks that integrate adversarial nodes directly into the DHT.

6.1 Avoiding Retroactive Privacy Attacks

Attackers. Our motivation is to protect against retroactive data disclosures, e.g., in response to a subpoena, court order, malicious compromise of archived data, or accidental data leakage. For some of these cases, such as the subpoena, the party initiating the subpoena is the obvious "attacker." The final attacker could be a user's ex-husband's lawyer, an insurance company, or a prosecutor. But executing a subpoena is a complex process involving many other actors, including potentially: the user's employer, the user's ISP, the user's email provider, unrelated nodes on the Internet, and other actors. For our purposes, we define *all* the involved actors as the "adversary."

Attack Strategies. The architecture and standard properties of the DHT cause significant challenges to an adversary who does *not* perform any computation or data interception prior to beginning the attack. First, the key

shares are unlikely to remain in the DHT much after the timeout, so the adversary will be incapable of retrieving the shares directly from the DHT. Second, even if the adversary could legally subpoena the machines that hosted the shares in the past, the churn in Vuze makes it difficult to determine the identities of those machines; many of the hosting nodes would have long disappeared from the network or changed their DHT index. Finally, with Vuze nodes scattered throughout the globe [63], gaining legal access to those machines raises further challenges. In fact, these are all reasons why the use of a DHT such as Vuze for our application is compelling.

We therefore focus on what an attacker might do *prior* to the expiration of a VDO, with the goal of amplifying his ability to reveal the contents of the VDO in the *future*. We consider three principal strategies for such precomputation.

Strategy (1): Decapsulate VDO Prior to Expiration. An attacker might try to obtain a copy of the VDO and revoke its privacy *prior* to its expiration. This strategy makes the most sense when we consider, e.g., an email provider that proactively decapsulates all VDO emails in real-time in order to assist in responding to future subpoenas. The natural defense would be to further encapsulate VDOs in traditional encryption schemes, like PGP or GPG, which we support with our FireVanish application. The use of PGP or GPG would prevent the webmail provider from decapsulating the VDO prior to expiration. And, by the time the user is forced to furnish her PGP private keys, the VDO would have expired. For the self-destructing trash bin and the Vanishing Files application, however, the risk of this attack is minimal.

Strategy (2): Sniff User's Internet Connection. An attacker might try to intercept and preserve the data users push into or retrieve from the DHT. An ISP or employer would be most appropriately positioned to exploit this vector. Two natural defenses exist for this: the first might be to use a DHT that by default encrypts communications between nodes. Adding a sufficient level of encryption to existing DHTs would be technically straightforward assuming that the ISP or employer were passive and hence not expected to mount man-in-the-middle attacks. For the encryption, Vanish could compose with an ephemeral key exchange system in order to ensure that these encrypted communications remain private even if users' keys are later exposed. Without modifying the DHT, the most natural solution is to compose with Tor [19] to tunnel one's interactions with a DHT through remote machines. One could also use a different exit node for each share to counter potentially malicious Tor exit nodes [36, 66], or use Tor for only a subset of the shares.

Strategy (3): Integrate into DHT. An attacker might try to integrate itself into the DHT in order to: create copies of all data that it is asked to `store`; intercept internal DHT `lookup` procedures and then issue `get` requests of his own for learned indices; mount a Sybil attack [26] (perhaps as part of one of the other attacks); or mount an Eclipse attack [60]. Such DHT-integrated attacks deserve further investigation, and we provide such an analysis in Section 6.2.

We will show from our experiments in Section 6.2 that an adversary would need to join the 1M-node Vuze DHT with approximately 80,000—90,000 malicious nodes to mount a `store`-based attack and capture a reasonable percentage of the VDOs (*e.g.*, 25%). Even if possible, sustaining such an attack for an extended period of time would be prohibitively expensive (close to $860K/year in Amazon EC2 computation and networking costs). The `lookup`-based attacks are easy to defeat using localized changes to Vanish clients. The Vuze DHT already includes rudimentary defenses against the Sybil attack and a full deployment of Vanish could leverage the existing body of works focused on hardening DHTs against Sybil and Eclipse attacks [9, 14, 16, 26, 51].

Deployment Decisions. Given attack strategies (1) and (2), a user of FireVanish, Vanishing Files, or any future Vanish-based application is faced with several options: to use the basic Vanish system or to compose Vanish with other security mechanisms like PGP/GPG or Tor. The specific decision is based on the threats to the user for the application in question.

Vanish is oriented towards personal users concerned that old emails, Facebook messages, text messages, or files might come back to "bite" them, as eloquently put in [42]. Under such a scenario, an ISP trying to assist in future subpoenas seems unlikely, thus we argue that composing Vanish with Tor is unnecessary for most users. The use of Tor seems even less necessary for some of the threats we mentioned earlier, like a thief with a stolen laptop.

Similarly, it is reasonable to assume that email providers will not proactively decapsulate and archive Vanishing Emails prior to expiration. One factor is the potential illegality of such accesses under the DMCA, but even without the DMCA this seems unlikely. Therefore, users can simply employ the FireVanish Gmail plugin without needing to exchange public keys with their correspondents. However, because our plugin extends FireGPG, any user already familiar with GPG could leverage our plugin's GPG integration.

Data Sanitization. In addition to ensuring that Vanish meets its security and privacy goals, we must verify that the surrounding operating environment does not preserve information in a non-self-destructing way. For this reason, the system could leverage a broad set of ap-

proaches for sanitizing the Vanish environment, including secure methods for overwriting data on disk [31], encrypting virtual memory [50], and leveraging OS support for secure deallocation [15]. However, even absent those approaches, forensic analysis would be difficult if attempted much later than the data's expiration for the reasons we've previously discussed: by the time the forensic analysis is attempted relevant data is likely to have disappeared from the user's machine, the churn in the DHT would have made shares (and nodes) vanish irrevocably.

6.2 Privacy Against DHT-Integrated Adversaries

We now examine whether an adversary who interacts with the DHT *prior* to a VDO's expiration can, in the future, aid in retroactive attacks against the VDO's privacy. During such a precomputation phase, however, the attacker does not know which VDOs (or even which users) he might eventually wish to attack. While the attacker could compile a list of worthwhile targets (*e.g.*, politicians, actors, etc.), the use of Tor would thwart such targeted attacks. Hence, the principle strategy for the attacker would be to create a copy of as many key shares as possible. Moreover, the attacker must do this continuously — 24x7 — thereby further amplifying the burden on the attacker.

Such an attacker might be *external* to the DHT — simply using the standard DHT interface in order to obtain key shares — or *internal* to the DHT. While the former may be the only available approach for DHTs like OpenDHT, the approach is also the most limiting to an attacker since the shares are stored at pseudorandomly generated and hence unpredictable indices. An attacker integrating into a DHT like Vuze has significantly more opportunities and we therefore focus on such DHT-integrating adversaries here.

Experimental Methodology. We ran extensive experiments on a private deployment of the Vuze DHT. In each experiment, a set of honest nodes pushed VDO shares into the DHT and retrieved them at random intervals of time, while malicious nodes sniffed `stores` and `lookups`.[3] Creating our own Vuze deployment allowed us to experiment with various system parameters and workloads that we would not otherwise have been able to manipulate. Additionally, experimenting with attacks against Vuze at sufficient scale would have been prohibitively costly for us, just as it would for an attacker.

Our experiments used 1,000, 2,000, 4,500, and 8,000-node DHTs, which are significantly larger than those used for previous empirical DHT studies (e.g. 1,000 nodes in [53]). For the 8,000-node experiments we used 200 machine instances of Amazon's EC2 [2] compute cloud. For smaller experiments we used 100 of Emulab's 3GHz, 2GB machines [27]. In general, memory is the bottleneck, as each Vuze node must run in a separate process to act as a distinct DHT node. Approximately 50 Vuze nodes fit on a 2-GB machine.

Churn (node death and birth) is modeled by a Poisson distribution as in [53]. Measurements of DHT networks have observed different median lifetime distributions, e.g., 2.4 minutes for Kazaa [30], 60 minutes for Gnutella [57], and 5 hours with Vuze [28] (although this measurement may be biased towards longer-lived nodes). We believe that these vast differences stem from different content and application types that rely on these networks (e.g., the difference between audio and video clips). We chose a 2-hour median node lifetime, which provides insight into the availability—security tradeoffs under high churn.

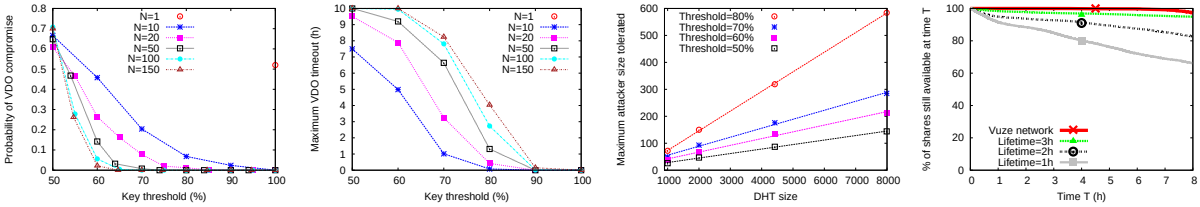
6.2.1 The Store Sniffing Attack

We first examine a `store` sniffing attack in which the adversary saves all of the index-to-value mappings it receives from peers via `store` messages. Such an attacker might receive a VDO's key shares in one of two ways: directly from the user during a VDO's creation or refresh, or via replication. In Vuze, nodes replicate their cached index-to-value mappings every 30 minutes by pushing each mapping to 20 nodes whose IDs are closest to the mapping's index.

Effects of VDO Parameters on Security. Our first goal is to assess how security is affected by the VDO parameters N (the number of key shares distributed for each VDO) and the key threshold (the percent of the N shares required to decrypt a VDO). Figure 7(a) plots the probability that an attacker can capture sufficient key shares to revoke the privacy of a given VDO as a function of N and the threshold. This figure assumes the attacker has compromised 5% of the nodes in a 1,000-node DHT. Not surprisingly, as the number of shares N increases, the attacker's success probability drops significantly. Similarly, increasing the threshold increases security (i.e., decreases the attacker's success probability).

Availability is also affected by the VDO parameters and the tradeoff is shown in Figure 7(b). Here we see the maximum timeout (i.e., the VDO's lifetime) as a function of N and the threshold. The maximum VDO timeout is the largest time at which 99% of a set of 1,000 VDOs remained available in our experiment. The timeout is capped by our 10-hour experimental limit. From the figure, we see that increasing N improves not only security, but also availability. We also see that smaller thresholds support longer timeouts, because the system can toler-

[3] Vuze `get` messages do not reveal additional information about values stored in the DHT, so we do not consider them.



(a) Parameters and security. (b) Parameters and availability. (c) Tolerated attacker sizes. (d) Churn effect on availability.

Figure 7: **Analysis of the store sniffing attack.** Fig. (a): the attacker's success probability with increasing N and key threshold for a 1000-node DHT with 50 malicious nodes. Larger N and high thresholds ($\geq 65\%$) provide good security. Fig. (b): maximum VDO timeout supported for a .99 availability level. Large N with smaller key thresholds ($\leq 70\%$) provide useful VDO timeouts. Fig. (c): maximum number of attacker nodes that a DHT can tolerate, while none of the 1,000 VDOs we pushed were compromised. Fig. (a), (b), and (c) assume 2-hour churn. Fig. (d): the single-share availability decreases over time for different churn models in our private network and for the real Vuze network.

ate more share loss. The choice of threshold thus involves a tradeoff between security and availability: high thresholds provide more security and low thresholds provide longer lifetime. For example, if a lifetime of only 4 hours is needed — which might be reasonable for certain emails or SMSs — then choosing $N = 50$ and threshold 75% leads to good security and performance. If a timeout of 8 hours is required, $N = 100$ and threshold of 70% is a good tradeoff for the 2-hour churn. Thus, by tuning N and the key threshold, we can obtain high security, good availability, and reasonable performance in the context of a small 1,000-node DHT and 5% attackers.

Attacker Sizes. We now consider how many attacker nodes a DHT deployment of a given size can tolerate with small chance that the attacker succeeds in pre-obtaining a sufficient number of shares for any VDO. Figure 7(c) shows the maximum attacker sizes tolerated by DHTs of increasing sizes, for various key thresholds. The values are calculated so as to ensure that none of the 1,000 VDOs we experimented with was compromised. We computed these values from experiments using $N = 150$, 2-hour churn, and various attacker sizes for each DHT size. For an 8,000-node DHT, even if 600 nodes are controlled by a store-sniffing attacker, the adversary would still not obtain any of our 1,000 VDOs.

More important, Figure 7(c) suggests that the number of attackers that the DHT can tolerate grows linearly with DHT size. Assuming this trend continues further, we estimate that, in a 1M-node DHT, an attacker with 35,000 nodes would still have less than 10^{-3} probability of recording a sufficient number of shares to compromise a single VDO with $N = 150$ and a threshold of 70%.

We have also experimented with a different metric of success: requiring an attacker to obtain enough key shares to compromise at least 25% of all VDOs. Concretely, for $N = 150$ and a threshold of 80%, our experiment with a 8,000 node DHT required the attacker to control over 710 nodes. This value also appears to grow linearly in the size of the DHT; extrapolating to a 1M-node DHT, such an attack would require at least 80,000 malicious nodes. We believe that inserting this number of nodes into the DHT, while possible for limited amounts of time, is too expensive to do continuously (we provide a cost estimate below).

Finally, we note that our refresh mechanism for extending Vuze timeouts (explained in Section 4) provides good security properties in the context of store sniffing attacks. Given that our mechanism pushes new shares in each epoch, an attacker who fails to capture sufficient shares in one epoch must start anew in the next epoch and garner the required threshold from zero.

Setting Parameters for the Vuze Network. These results provide a detailed study of the store sniffing attack in the context of a 2-hour churn model induced on a private Vuze network. We also ran a selected set of similar availability and store attack experiments against a private network with a 3-hour churn model, closer to what has been measured for Vuze.[4] The resulting availability curve for the 3-hour churn now closely resembles the one in the real Vuze network (see Figure 7(d)). In particular, for both the real network and the private network with a 3-hour churn model, a ratio of 90% and $N \geq 20$ are enough to ensure VDO availability of 7 hours with .99 probability. Thus, from an availability standpoint, the longer lifetimes allow us to raise the threshold to 90% to increase security.

From a security perspective, our experiments show that for an 8,000-node DHT, 3-hour churn model, and VDOs using $N = 50$ and threshold 90%, the attacker requires at least 820 nodes in order to obtain $\geq$25% of the VDOs. This extrapolates to a requirement of $\approx$87,000 nodes on Vuze to ensure attack effectiveness. Returning to our cost argument, while cloud computing in a system such as Amazon EC2 is generally deemed in-

[4] We used VDOs of $N = 50$ and thresholds of 90% for these experiments.

expensive [18], the cost to mount a year-long 87,000-node attack would be over $860K for processing and Internet traffic alone, which is sufficiently high to thwart an adversary's compromise plans in the context of our personal use targets (*e.g.*, seeking sensitive advice from friends over email). Of course, for larger N (*e.g.*, 150), an attacker would be required to integrate even more nodes and at higher cost. Similarly, the cost of an attack would increase as more users join the Vuze network.

Overall, to achieve good performance, security and availability, we recommend using $N = 50$ and a threshold of 90% for VDOs in the current Vuze network. Based on our experiments, we conclude that under these parameters, an attacker would be required to compromise between 8—9% of the Vuze network in order to be effective in his attack.

6.2.2 The Lookup Sniffing Attack

In addition to seeing `store` requests, a DHT-integrated adversary also sees `lookup` requests. Although Vuze only issues lookups prior to `storing` and `getting` data objects, the lookups pass through multiple nodes and hence provide additional exposure for VDO key shares. In a lookup sniffing attack, whenever an attacker node receives a lookup for an index, it actively fetches the value stored at that index, if any. While more difficult to handle than the passive `store` attack, the `lookup` attack could increase the adversary's effectiveness.

Fortunately, a simple, *node-local* change to the Vuze DHT thwarts this attack. Whenever a Vanish node wants to store to or retrieve a value from an index I, the node looks up an *obfuscated* index I', where I' is related to but different from I. The client then issues a `store`/`get` for the original index I to the nodes returned in response to the lookup for I'. In this way, the retrieving node greatly reduces the number of other nodes (and potential attackers) who see the real index.

One requirement governs our simple choice of an obfuscation function: the same set of replicas must be responsible for both indexes I and I'. Given that Vuze has 1M nodes and that IDs are uniformly distributed (they are obtained via hashing), all mappings stored at a certain node should share approximately the higher-order $\log_2(10^6) \approx 20$ bits with the IDs of the node. Thus, looking up only the first 20b of the 160b of a Vuze index is enough to ensure that the nodes resulted from the lookup are indeed those in charge of the index. The rest of the index bits are useless in lookups and can be randomized, and are rehabilitated only upon sending the final `get`/`store` to the relevant node(s). We conservatively choose to randomize the last 80b from every index looked up while retrieving or storing mappings.

Lacking full index information, the attacker would have to try retrieving all of the possible indexes starting with the obfuscated index (2^{80} indexes), which is impossible in a timely manner. This Vuze change was trivial (only 10 lines of modified code) and it is completely local to Vanish nodes. That is, the change does not require adoption by any other nodes in the DHT to be effective.

6.2.3 Standard DHT Attacks

In the previous sections we offered an in-depth analysis of two data confidentiality attacks in DHTs (store and lookup sniffing), which are specific in the context of our system. However, the robustness of communal DHTs to more general attacks has been studied profusely in the past and such analyses, proposed defenses, and limitations are relevant to Vanish, as well. Two main types of attacks identified by previous works are the Sybil attack [26] and the Eclipse (or route hijacking) attack [60]. In the Sybil attack, a few malicious nodes assume a large number of identities in the DHT. In the Eclipse attack, several adversarial nodes can redirect most of the traffic issued by honest nodes toward themselves by poisoning their routing tables with malicious node contact information [60].

The Vuze DHT already includes a rudimentary defense against Sybil attacks by constraining the identity of a Vuze node to a function of its IP address and port modulo 1999. While this measure might be sufficient for the early stages of a Vanish deployment, stronger defenses are known, e.g., certified identities [26] and periodic cryptographic puzzles [9] for defense against Sybil attacks and various other defenses against Eclipse attacks [14, 51]. Given that the core Vanish system is network-agnostic, we could easily port our system onto more robust DHTs implementing stronger defenses. Moreover, if Vanish-style systems become popular, it would also be possible to consider Vanish-specific defenses that could leverage, e.g., the aforementioned tight coupling between Vanish and the identities provided by PGP public keys. Finally, while we have focused on the Vuze DHT — and indeed its communal model makes analyzing security more interesting and challenging — Vanish could also split keys across *multiple* DHTs, or even DHTs and managed systems, as previously noted (Section 4). The different trust models, properties, and risks in those systems would present the attacker with a much more difficult task.

7 Related Work

We have discussed a large amount of related work in Section 2 and throughout the text. As additional related work, the Adeona system also leverages DHTs for increased privacy, albeit with significantly different goals [55]. Several existing companies aim to achieve

similar goals to ours (e.g., self-destructing emails), but with very different threat models (company servers must be trusted) [20]. Incidents with Hushmail, however, may lead users to question such trust models [59]. There also exists research aimed at destroying archived data where the data owner has the ability to explicitly and manually erase extra data maintained elsewhere, e.g., [8]; we avoid such processes, which may not always succeed or may be vulnerable to their own accidental copying or disclosures. Finally, albeit with different goals and perspectives, Rabin proposes an information-theoretically secure encryption system that leverages a decentralized collection of dedicated machines that continuously serve random pages of data [52], which is related to the limited storage model [33]. Communicants, who pre-share symmetric keys, can download and xor specific pages together to derive a one-time pad. The commonality between our approach and Rabin's is in the use of external machines to assist in privacy; the model, reliance on dedicated services, and pre-negotiation of symmetric keys between communicants are among the central differences.

8 Conclusions

Data privacy has become increasingly important in our litigious and online society. This paper introduced a new approach for protecting data privacy from attackers who retroactively obtain, through legal or other means, a user's stored data and private decryption keys. A novel aspect of our approach is the leveraging of the essential properties of modern P2P systems, including churn, complete decentralization, and global distribution under different administrative and political domains. We demonstrated the feasibility of our approach by presenting *Vanish*, a proof-of-concept prototype based on the Vuze global-scale DHT. Vanish causes sensitive information, such as emails, files, or text messages, to irreversibly self-destruct, without any action on the user's part and without any centralized or trusted system. Our measurement and experimental security analysis sheds insight into the robustness of our approach to adversarial attacks.

Our experience also reveals limitations of existing DHTs for Vanish-like applications. In Vuze, for example, the fixed data timeout and large replication factor present challenges for a self-destructing data system. Therefore, one exciting direction of future research is to redesign existing DHTs with our specific privacy applications in mind. Our plan to release the current Vanish system will help to provide us with further valuable experience to inform future DHT designs for privacy applications.

9 Acknowledgements

We offer special thanks to Steve Gribble, Arvind Krishnamurthy, Mark McGovern, Paul Ohm, Michael Piatek, and our anonymous reviewers for their comments on the paper. This work was supported by NSF grants NSF-0846065, NSF-0627367, and NSF-614975, an Alfred P. Sloan Research Fellowship, the Wissner-Slivka Chair, and a gift from Intel Corporation.

References

[1] C. Alexander and I. Goldberg. Improved user authentication in off-the-record messaging. In *WPES*, 2007.

[2] Amazon.com. Amazon elastic compute cloud (EC2). `http://aws.amazon.com/ec2/`, 2008.

[3] Azureus. `http://www.vuze.com/`.

[4] BBC News. US mayor charged in SMS scandal. `http://news.bbc.co.uk/2/hi/americas/7311625.stm`, 2008.

[5] M. Bellare and A. Palacio. Protecting against key exposure: Strongly key-insulated encryption with optimal threshold. *Applicable Algebra in Engineering, Communication and Computing*, 16(6), 2006.

[6] M. Bellare and B. Yee. Forward security in private key cryptography. In M. Joye, editor, *CT-RSA 2003*, 2003.

[7] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. In *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science (FOCS '82)*, 1982.

[8] D. Boneh and R. Lipton. A revocable backup system. In *USENIX Security*, 1996.

[9] N. Borisov. Computational puzzles as Sybil defenses. In *Proc. of the Intl. Conference on Peer-to-Peer Computing*, 2006.

[10] N. Borisov, I. Goldberg, and E. Brewer. Off-the-record communication, or, why not to use PGP. In *WPES*, 2004.

[11] R. Canetti, Y. Dodis, S. Halevi, E. Kushilevitz, and A. Sahai. Exposure-resilient functions and all-or-nothing transforms. In B. Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 453–469, Bruges, Belgium, May 14–18, 2000. Springer-Verlag, Berlin, Germany.

[12] R. Canetti, C. Dwork, M. Naor, and R. Ostrovsky. Deniable encryption. In B. S. K. Jr., editor, *CRYPTO'97*, 1997.

[13] R. Canetti, S. Halevi, and J. Katz. A forward-secure public-key encryption scheme. In *EUROCRYPT 2003*, 2003.

[14] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure routing for structured peer-to-peer overlay networks. In *Proc. of OSDI*, 2002.

[15] J. Chow, B. Pfaff, T. Garfinkel, and M. Rosenblum. Shredding your garbage: Reducing data lifetime through secure deallocation. In *USENIX Security*, 2005.

[16] T. Condie, V. Kacholia, S. Sankararaman, J. M. Hellerstein, and P. Maniatis. Induced churn as shelter from routing table poisoning. In *Proc. of NDSS*, 2006.

[17] A. Czeskis, D. J. S. Hilaire, K. Koscher, S. D. Gribble, T. Kohno, and B. Schneier. Defeating encrypted and deniable file systems: TrueCrypt v5.1a and the case of the tattling OS and applications. In *3rd USENIX HotSec*, July 2008.

[18] M. Dama. Amazon EC2 scalable processing power. `http://www.maxdama.com/2008/08/amazon-ec2-scalable-processing-power.html`, 2008.

[19] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *USENIX Security*, 2004.

[20] Disappearing Inc. Disappearing Inc. product page. `http://www.specimenbox.com/di/ab/hwdi.html`, 1999.

[21] Y. Dodis, M. K. Franklin, J. Katz, A. Miyaji, and M. Yung. Intrusion-resilient public-key encryption. In *CT-RSA 2003*, volume 2612, pages 19–32. Springer-Verlag, Berlin, Germany, 2003.

[22] Y. Dodis, M. K. Franklin, J. Katz, A. Miyaji, and M. Yung. A generic construction for intrusion-resilient public-key encryption. In T. Okamoto, editor, *CT-RSA 2004*, volume 2964 of *LNCS*, pages 81–98, San Francisco, CA, USA, Feb. 23–27, 2004. Springer-Verlag, Berlin, Germany.

[23] Y. Dodis, J. Katz, S. Xu, and M. Yung. Key-insulated public key cryptosystems. In *EUROCRYPT 2002*, 2002.

[24] Y. Dodis, A. Sahai, and A. Smith. On perfect and adaptive security in exposure-resilient cryptography. In *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 301–324. Springer-Verlag, Berlin, Germany, 2001.

[25] Y. Dodis and M. Yung. Exposure-resilience for free: The case of hierarchical ID-based encryption. In *IEEE International Security In Storage Workshop*, 2002.

[26] J. R. Douceur. The sybil attack. In *International Workshop on Peer-to-Peer Systems*, 2002.

[27] Emulab. Emulab – network emulation testbed. `http://www.emulab.net/`, 2008.

[28] J. Falkner, M. Piatek, J. John, A. Krishnamurthy, and T. Anderson. Profiling a million user DHT. In *Internet Measurement Conference*, 2007.

[29] D. Goodin. Your personal data just got permanently cached at the US border. `http://www.theregister.co.uk/2008/05/01/electronic_searches_at_us_borders/`, 2008.

[30] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *Proc. of SOSP*, 2003.

[31] P. Gutmann. Secure deletion of data from magnetic and solid-state memory. In *USENIX Security*, 1996.

[32] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest we remember: Cold boot attacks on encryption keys. In *USENIX Security*, 2008.

[33] U. M. Maurer. Conditionally-perfect secrecy and a provably-secure randomized cipher. *Journal of Cryptology*, 5:53–66, 1992.

[34] V. Mayer-Schoenberger. Useful Void: the art of forgetting in the age of ubiquitous computing. *Working Paper, John F. Kennedy School of Government, Harvard University*, 2007.

[35] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the XOR metric. In *Proc. of Peer-to-Peer Systems*, 2002.

[36] D. McCoy, K. Bauer, D. Grunwald, T. Kohno, and D. Sicker. Shining light in dark places: Understanding the Tor network. In *Privacy Enhancing Technologies Symposium*, July 2008.

[37] D. McCullagh. Feds use keylogger to thwart PGP, Hushmail. `news.cnet.com/8301-10784_3-9741357-7.html`, 2008.

[38] D. McCullagh. Security guide to customs-proofing your laptop. `http://www.news.com/8301-13578_3-9892897-38.html`, 2008.

[39] S. K. Nair, M. T. Dashti, B. Crispo, and A. S. Tanenbaum. A hybrid PKI-IBC based ephemerizer system. In *International Information Security Conference*, 2007.

[40] E. Nakashima. Clarity sought on electronic searches. `http://www.washingtonpost.com/wp-dyn/content/article/2008/02/06/AR2008020604763.html`, 2008.

[41] New York Times. F.B.I. Gained Unauthorized Access to E-Mail. `http://www.nytimes.com/2008/02/17/washington/17fisa.html?_r=1&hp=&adxnnl=1&oref=slogin&adxnnlx=1203255399-44ri626iqXg7QNmwzoeRkA`, 2008.

[42] News 24. Think before you SMS. `http://www.news24.com/News24/Technology/News/0,,2-13-1443_1541201,00.html`, 2004.

[43] Office of Public Sector Information. Regulation of Investigatory Powers Act (RIPA), Part III – Investigation of Electronic Data Protected by Encryption etc. `http://www.opsi.gov.uk/acts/acts2000/ukpga_20000023_en_8`, 2000.

[44] P. Ohm. The Fourth Amendment right to delete. *The Harvard Law Review*, 2005.

[45] PC Magazine. Messages can be forever. `http://www.pcmag.com/article2/0,1759,1634544,00.asp`, 2004.

[46] R. Perlman. The Ephemerizer: Making data disappear. *Journal of Information System Security*, 1(1), 2005.

[47] R. Perlman. File system design with assured delete. In *Security in Storage Workshop (SISW)*, 2005.

[48] F. A. P. Petitcolas, R. J. Anderson, and M. G. Kuhn. Information hiding: A survey. *Proceedings of the IEEE*, 87(7), 1999.

[49] B. Poettering. "ssss: Shamir's Secret Sharing Scheme". `http://point-at-infinity.org/ssss/`, 2006.

[50] N. Provos. Encrypting virtual memory. In *USENIX Security*, 2000.

[51] K. P. N. Puttaswamy, H. Zheng, and B. Y. Zhao. Securing structured overlays against identity attacks. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2008.

[52] M. O. Rabin. Provably unbreakable hyper-encryption in the limited access model. In *IEEE Information Theory Workshop on Theory and Practice in Information-Theoretic Security*, 2005.

[53] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a DHT. In *Proc. of the Annual Technical Conf.*, 2004.

[54] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. OpenDHT: A public DHT service and its uses. In *Proc. of ACM SIGCOMM*, 2005.

[55] T. Ristenpart, G. Maganis, A. Krishnamurthy, and T. Kohno. Privacy-preserving location tracking of lost or stolen devices: Cryptographic techniques and replacing trusted third parties with DHTs. In *17th USENIX Security Symposium*, 2008.

[56] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Lecture Notes in Computer Science*, 2001.

[57] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proc. of Multimedia Computing and Networking*, 2002.

[58] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[59] R. Singel. Encrypted e-mail company Hushmail spills to feds. `http://blog.wired.com/27bstroke6/2007/11/encrypted-e-mai.html`, 2007.

[60] A. Singh, T. W. Ngan, P. Druschel, and D. S. Wallach. Eclipse attacks on overlay networks: Threats and defenses. In *Proc. of INFOCOM*, 2006.

[61] Slashdot. `http://tech.slashdot.org/article.pl?sid=09/02/17/2213251&tid=267`, 2009.

[62] Spitzer criminal complaint. `http://nytimes.com/packages/pdf/nyregion/20080310spitzer-complaint.pdf`, 2008.

[63] M. Steiner and E. W. Biersack. Crawling Azureus. Technical Report RR-08-223, 2008.

[64] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of ACM SIGCOMM*, pages 149–160, 2001.

[65] Y. Xie, F. Yu, K. Achan, E. Gillum, M. Goldszmidt, and T. Wobber. How dynamic are IP addresses? In *Proc. of SIGCOMM*, 2007.

[66] K. Zetter. Tor researcher who exposed embassy e-mail passwords gets raided by Swedish FBI and CIA. `http://blog.wired.com/27bstroke6/2007/11/swedish-researc.html`, 2007.