Rendezvous: toward a distributed secure storage model

Kelly Nicole Kaoudis
kaoudis@colorado.edu

ABSTRACT

We describe a distributed anonymous filesharing network based on insight from studying the efficacy and usability of anonymity-providing services. First we justify two related pillars of our approach: security built in from the start everywhere possible, and realizing the implications of usability (or lack thereof). We detail related literature and our takeaways from others' failures and successes.

Our work originates with prior study of the construction of the Tor network's performance, the onion protocol, and interaction between Tor clients and hidden services. We do not use Tor itself as the foundation for our system due to performance and protocol concerns. We also incorporate design lessons from currently widespread implementations of peer-to-peer filesharing systems, distributed databases such as Kademlia and Pastry, and key management systems such as public files and functionally trusted third parties.

Recently we have begun considering implications of Byzantine failure and other potential faults for our system. In light of these concerns, the third pillar of our system is simplicity. This means reducing the number of technologies and techniques used, without compromising our other priorities, characteristic security and usability. The fourth pillar of our completed system design will be modularity, as in the familiar Unix philosophy. Each element should work as a cog in the greater system without disrupting the user's experience of the system as a seamless entity. Components will be independently testable and secured, composable, and re-usable as needed as stand-alone modules in future work.

We detail the implementation to date and possible issues in our continued execution of our design. We conclude with analysis of possible threats to our system and design as it currently stands, and a brief description of continuing work.

1. INTRODUCTION

We first describe the basic principles our distributed temporary file rendezvous is based on.

Security

The United Nations' Universal Declaration of Human Rights details certain freedoms to which all human beings are entitled. Its twelfth and nineteenth articles in particular, similarly to the First Amendment to the United States Constitution and Section 2b of the Canadian Charter of Rights and Freedoms, state that all humans should have the right to freedom of expression and opinion and should not be subject to arbitrary interference with their assembly and privacy.

While the Internet is a multinational, borderless entity, its cultural heritage is based in the expectation of being able to speak freely and exchange ideas and information with anyone, at any time. Tor [13], Freenet [7], and I2P [47], which all claim to allow their users to transfer and/or receive information without needing to expose real-world credentials such as legal names and IPs, are utilized nearly worldwide. This speaks to a strong desire for uncensored communication.

If free speech in particular is to be effectively realized, security and user privacy must be top considerations at all levels of application design. When security is not a bolt-on afterthought but rather is included in the specification from the beginning [41], it is much more difficult to compromise. In the interest of the continuation of a free and open web [22], we consider this the most important principle our design is based on.

Usability

Our second concern is the ability of our users to easily understand and employ our service [3] [50] without accidentally revealing personally identifiable information or other sensitive data. Our system should offer the fewest (unpleasant) surprises possible to the user both in terms of how the interface works and how the underlying system operates [38].

The general public expects the convenience of being able to launch any sort of application with little fuss. Additionally, popular virtual private storage / storage as a service providers such as Dropbox and Apple's iCloud do nothing to dissuade the general belief that properly securing data is trivial, or that one's data will be secure once entrusted to the service provider.

We argue that neither of these suppositions are currently true, especially in light of the high-profile breaches [27] [32] of such services that have occurred in the last few months, spreading the 'private' data of public figures across the in-

ternet. However, if a service requires extra effort and time compared to 'the usual', it is unlikely that most human beings will put in that effort even if it is understood that they *should* (e.g. avoiding dentist appointments, and similar sociological phenomena).

While Tor and similar services were designed with some measure of usability in mind [11], we argue that it is possible to do better [20], especially considering recent events: hidden service operators and certain users likely lacking the technical knowledge and/or desire to read and understand parts of the Tor specification relevant to configuring their torrc file and software properly have had their anonymity and privacy compromised.

Simplicity

Snowden argues that some possible adversaries are capable of a trillion hashes per second [19], so it is logical to use as much layered, strong encryption as we can manage without overcomplicating the user experience or the back end code. File storage and retrieval is entirely based on whether one has the right PGP keypair [30], and all information entrusted to the system is encrypted with source and final destination in mind before it is temporarily stored in the cloud. Encrypted transit protocols are also employed both to/from the system and between storage nodes within the system itself.

During the beginning of the implementation process, we realized our design at the time used a lot of moving parts simply because the technology was interesting, new, or partially did one thing we needed. Byzantine failure is more likely to occur in a more complicated system [11]: there are more parts that might potentially fail. Furthermore, the simpler and more sturdy a system and its parts are [42], the easier it is to extend that system and reuse its components for other projects. The simpler, more flexible, and more extensible a system is, the more likely we are to convince others to hack on and improve it.

Modularity

After simplifying as much as we could at the time, we realized we had considered the current wants and needs of the user and the author, but had failed to account for future individuals who might want to pick up this project or some part of it and make use of it in other ways. Therefore, we include the concept of modularity [38] as our final underlying design concern. Each component should do just one thing [2], should be composable, and should be designed and tested with limited reliance on elements belonging to other components. These modular units should compose to create the foundation of a system that fits the following problem statement and is addtionally extendable.

Design Overview

Alice should be able to easily exchange important information with Bob without allowing anyone else knowledge of what is being transferred or where the data are located at any point during the process. Neither Alice or Bob should need extensive low-level technical knowledge to properly configure and use the service in a secure, correct fashion. Neither the adversary nor Bob should know Alice's geolocation or IP. Neither Alice nor the adversary should know Bob's geolocation or IP.

The ideal is an accessible service that allows the general public to share files once (or multiple times) using at least some level of encryption and randomization to keep the adversary from having knowledge of the mapping between *any* users' personal information such as location and real name, and the encrypted data temporarily stored within the service in a randomized fashion.

We believe further exploration in this space is necessary and that anonymity is not a solved problem, especially given the number of possible attacks on Tor [24] [28] [5] [9] [25] employed lately by governments and other entities. Many have attempted to create a reasonably reliable, secure method for information transfer and have partially succeeded, standing upon the shoulders of those who tried and did not succeed quite so well previously. We propose yet another partial solution: a secure service that is simple to use and to extend, fulfilling the general expectation that data entrusted to the cloud is well protected. We wish to facilitate anonymous, distributed sharing of encrypted information in a peer-to-peer fashion. Perhaps future work, possibly even building on this, will come closer to realizing freedom of speech and information.

2. RELATED WORK

We present relevant parts of a few examples of work in the security and privacy space and explore differences and similarities with our own architecture.

Tor and the onion protocol

Tor [13] is a widely used, well documented project claiming to provide more security than its users would have conducting business on the open web. The Tor FAQ [48] states that simple filesharing is not desired within the Tor network, especially using P2P protocols, which are not anonymized by Tor and SOCKS proxies in general, so we chose not to create another layer on top of Tor.

In direct contradiction to methods employed by Onionshare [29] and similar services deployed on top of the existing Tor system, we chose to roll our own underlying system, reimagining and integrating only the features we really needed from Tor, mainly the hidden service/introduction point model and the anonymity given by the latest iteration of onion routing.

We argue that slowness of Tor compared to the open web [14] is due more to an imbalance [18] between users and certain kinds of contributors (and additionally, in part, to load balancing issues [1]) than to overhead from the anonymity precautions. Due (we hypothesize) to the Tor system's notoriety and additionally the general public's lack of under-

standing of what Tor really is and how it works, the percentage of those using Tor and similar systems for anonymity who also contribute back [36] by running a relay or an exit node or improving the codebase is much smaller than the number of people use the service to protect themselves. The percentage of volunteers running an exit node, a critical part of the infrastructure, is currently smallest of all [37], due to abuse by malicious users and backlash from ISPs. Systems such as TorWard [31] are being developed to categorize and reduce the risk of ill-meaning traffic, but work in this space is just beginning. Our future work may include contributions to Tor in this direction.

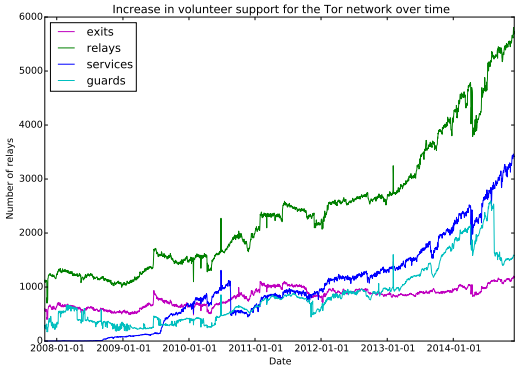

Figure 1: From the official Tor Metrics data, it is evident that relays and bridges with the 'exit' flag declared comprise the smallest number of active nodes at this time.

Furthermore it has been shown [6] using the Cisco traffic analysis tool Netflow that it is possible to uncloak around 81% of Tor users, especially in cases where the users did not have enough deep knowledge of the inner workings of the service. While we strongly advocate for an improved Tor, since it is already in widespread use, we propose a single-purpose partial alternative that addresses some of Tor's limitations.

Not implementing our service over Tor gives us the freedom to use UDP at the transport layer where desired (SOCKS proxies only protect TCP traffic) and also to employ a specially tailored peer-to-peer protocol for uploading and downloading files from the network.

Tor offers users the ability to set up rendezvous points known as "hidden services", or alternatives to sites on the open web offering anything servable over a SOCKS proxy. While the idea is that operating such a service would be anonymous in that the operator's IP address and physical location should never be revealed, it has been proven that this assumption is flawed in some cases [9]. Hidden services, by way of a "hidden service descriptor", which consists of the service's public key and a list of its acknowledged introduction points, can advertise themselves in a globally available distributed hash table directory. The descriptor is what's first found when a user types the service's DESC.onion address into their Tor browser's URL bar. The "DESC" is a base-32 encoding of a 10-octet hash of the service's public key.

Tor currently uses an incremental path-building design: each successive hop in a circuit (a path between a user's Tor client's entry point into the network and her destination) has to negotiate its own session keys. We include this kind of modularity in our initial design; transport of a given chunk of data within the database is roughly modelled after Tor circuits. Tor multiplexes many TCP streams on a single circuit to allow for a noisier background and also provide more efficiency than just one stream on a circuit would have. Therefore the next step from the client point of view in navigation to DESC.onion once the service descriptor has been fetched from a randomly chosen hidden service directory is as follows. The client picks one of the service's introduction (rendezvous) points, and establishes a rendezvous circuit which will facilitate connection to that introduction point. Once the introduction point has been established by the client, the *service* builds itself a new Tor circuit ending at that point as well. The rendezvous point authenticates and relays cells between the client circuit and the server circuit. We also incorporate the idea of rendezvous points into our design in two places.

I2P

Somewhat like Tor, I2P [26] is an anonymous network based around the *I2P router*. I2P is designed to safeguard typical internet functions such as email, file sharing, messaging, and so on, mainly within a separate network called the *darknet* which is built atop the bespoke UDP and TCP-like protocols SSU and NTCP. These protocols run on top of the peer-to-peer network, which itself runs on top of IP.

Unlike Tor and similar architectures, the peer-to-peer network that is I2P's ground-floor layer is completely decentralized. Decentralization implies better scalability, and of course no central trusted authority which may be potentially compromised. All peer and service metadata is instead stored in a distributed hash table called *netDB*. Network database records are stored in another Kademlia-like DHT [33] *floodfill*. Since netDB is run alongside regular I2P nodes, it may be considered less a trusted *group* of authorities than functionally interwoven in the network.

All connections inside the darknet are end-to-end encrypted; the router multiplexes these connections over paired, single-direction *I2P tunnels*, which are (when taken as a paired connection from server to client) analogous to Tor's circuits. These tunnels are constructed with the onion protocol. All participants are meant to be anonymous.

However, in practice this does not always appear to be the case. I2P is a much smaller project than Tor; this means fewer users providing white noise around one's traffic, fewer developers, and fewer volunteers running services.

Additionally, the port of Tahoe-LAFS [52] allowing decentralized storage similar to Freenet within I2P is not maintained, and is not up to date with the current version of the

filesystem. The website for the maintainer listed by I2P was down at time of writing. Actually, the only version the author could find that wasn't available from the Tahoe website was available as a package in the Arch Linux User Repository, and was not the port to I2P but rather the original version.

The largest difference between Tahoe and Freenet, according to Tahoe's creator [51], is Freenet's file distribution is entirely random and proper usage of Tahoe-LAFS depends on having a large network of friends and family one can cajole into also using the system, since Tahoe assumes discovery of storage happens out-of-band. Further, Tahoe makes no attempt to obscure the source or destination of requests. While it is exceedingly difficult to guarantee a high probability of remaining anonymous against sophisticated attackers (see the Tor section above) that doesn't mean that trying (and explicitly stating that users employ the service at their own risk) is not worthwhile.

Freenet

Freenet [7] is a location-independent peer-to-peer distributed filesystem of nodes that collaborate to store and retrieve encrypted files, which are named by 160-bit SHA-1 hashes that do not depend on the files' location. Each user maintains their own local datastore as a portion of their hard drive which might otherwise go under-utilized. Freenet maintains a *hops-to-live* limit on active requests for files, which is decremented at each node the request reaches, to prevent infinite cycling through the system.

We repurpose the *time-to-live* concept detailed by the IP specification and employed by Freenet to limit the number of downloads that can take place before the space dedicated to a file's chunks and identifier will be repurposed to hold other information (Freenet uses an LRU caching policy to determine what gets deleted from an individual node when there is not enough available space). Further, we do not store information locally on users' machines– our system requires obtaining a virtual private storage space to contribute instead.

Tracked and untracked peer-to-peer data transactions

Other cooperative peer-to-peer services like BitTorrent [8], Gnutella [39], uTorrent [49], and more experimental distributed architectures attempt to meet the goal of providing file transfer and/or temporary storage in a relatively private way. However, in the case of services based on the BitTorrent protocol and similar, a tracking server or some other method of keeping a list of users' IPs and ports is typically necessary. Every user whose torrent client requests a list of peers from which to obtain pieces of the file in question is not anonymous to those peers; nor are those peers anonymous to the user in question.

Distributed information stores related to the peer-to-peer model such as Pastry [40], Chord [43], Kademlia [33], Comet [17] and so on have advantages over widely used peer-to-peer services including greater consistency and fault tolerance, but none quite provide the kind of intrinsic, randomized, modular security we envision.

In our system, each file chunk gets its own peer-to-peer connection to a randomly selected node of the datastore, and then a further random transfer is made. Each of the initial peer-to-peer connections is untracked, and the destination node is not known to the client until transfer of a given data chunk is initiated.

Why Johnny can't stay anonymous

Johnny can't encrypt if the available encryption is non-intuitive to use. Whitten et al discovered by way of lab trials that PGP's user interface [50] was difficult to properly employ even for people with some technical background, at time of publication.

While the field of user experience has made huge progress since then and is one of the foremost considerations in many sorts of application and tools design, usability is still less frequently considered when security applications, tools, and frameworks are developed. If Johnny has some information he needs to get to Alice without others virtually reading over his shoulder, he shouldn't have to become an expert in low-level topics in order to understand and properly configure such services as will let him encrypt, sign, and send his documents with little fuss and without trusting a third party.

Let's say our hypothetical good guy Johnny is a journalist, activist, or just someone in a non-technical role at a corporation or other entity with shady regular practices which are actively hurting people or will in the long run be highly detrimental to a significant sector of the population. If services such as Tor are hard to set up or require too much knowledge Johnny doesn't already have or can't acquire quickly [34], he won't be able to get the word out. He might even be arrested if perhaps the use of such services is against local law. We propose creating a simple client-facing architecture and testing it on as many actual humans as we can convince to sit still long enough to try out Rendezvous before making it publicly available.

3. METHODS AND ARCHITECTURE

Our system has two main parts. The first is a clientside API which allows the user to prepare files for peer-to-peer upload, keep track of current in-progress and completed uploads, and orchestrate downloads. The second is a secure distributed database [12] [35] called a *rendezvous*, which (taken as a black box) behaves similarly to Tor's hidden service introduction points. The rendezvous consists of a layer of encryption and randomization for data over storage space located on VPS platforms and contributed to the database as a requirement for uploading one's own data, pay-to-play style. These two parts will both operate on the assumption that *data is only sent when it is asked for by the receiver* [4].

Control interface: GET, PUT, DELETE

A client has the ability to perform three operations on the datastore. One PUTs one's onion id and the hashes of any files available, then one PUTs the file chunks themselves. The file receiver will perform a GET, which entails sending the hash of the file they want as well as their own onion address and public key (not to be stored) to be verified against the file in question. A DELETE can only be issued by the file originator, in much the same way GETs can only be issued by those specified in the file's list of destination hashes. All of these operations are signed by their originator in much the same way a given encrypted, whole file itself is.

Key-hash storage and distribution

The storage of clients' onion addresses mapped to hashes of their available files (and the files' destinations) is handled by a small additional key,value store that sits alongside the rendezvous. This additional store is a functionally trusted entity which is entirely public and may at a later date become distributed similarly to the rendezvous store itself. Those who have made an upload of their credentials to the store are allowed to GET any data matching their public key, so long as they also have the file source's full public key.

Suppose Alice wants to send a file to Bob. She first registers an identifier [21] which, for our purposes, consists of a PGP public key either previously in use and added to Alice's client application, or generated for her by the client. A .onion address [45] (that isn't currently in use) is the base-32 encoding of a 10-octet selection from a SHA2-512 hash of this public key. If she would also like to upload content, the hash of the (encrypted) file in question is also calculated.

Assuming Alice has obtained the public key(s) of her file's recipient(s) out-of-band, she can essentially concatenate each one at a time with her public key and the file and take the hash of that. Data given a destination in this fashion cannot be reassembled and pulled out of the store without a public key that, hashed with the file and the public key of its origin point, matches the hash value attached to the file (and that data, once recovered in full, additionally cannot be decrypted without the matching private key). Privately destined data cannot even be *found* without knowledge of the public key belonging to its origin and a public key of a specified recipient.

There is some possibility for malicious users who have their own data in the store to pull out random onion addresses from the directory and try to get those users' files, but as long as the malicious users' keys don't match any part of the destination hash list, the unsuspecting "good" users' data should remain safe. Data without destination should naturally not be as sensitive as data directed at a specific entity. In the client interface, the default option will be to add a destination (or multiple destinations – which turn the single hash into a list of possible hash matches); the option to upload data without a specific recipient will be available but slightly less visible.

Alice's .onion address and a list of hashes of her full public key, the hash of the file, and the public key(s) of any recipients are sent

```
PUT { onion : .onion_A,
      dest: { h( h(f), idA, idB),
              h( h(f), idA, idC),
                   ... },
      TTL : x }
```

to our public-facing metadata store. TTL, which stands for time to live, is an eight-bit number historically indicating the time an IP datagram can exist in an internet system. For instance, when using the network utility *ping*, each ICMP packet sent out has a certain number of hops between routers it can make before it is discarded.

Our version of the TTL value accounts for the number of downloads that can take place before the shards of Alice's file will be unconditionally written over. The default TTL will be one. If the TTL is blank, the destination list must also be blank. If both of these are true, the file will be considered open for public download until its originator sends a DELETE command on the hash of the file to be deleted to the key,value store. The list of destination hashes, the time-to-live value, and Alice's address will also be attached to fragments of the file such that each can be retrieved from its rendezvous point.

PGP

We chose to use PGP clientside to encrypt and sign data and requests. The user has the option to either generate a new PGP keypair or to re-use the same one they might keep for email. Files are encrypted and signed with Alice's keypair (and destined for Bob) with an implementation of OpenPGP that we augmented with a few extra cryptographic functions and the ability to "chunk" a file once encrypted.

Layered encryption

Before any upload can take place, Alice's file(s) must be encrypted with her generated keypair. The encrypted files are then signed, and a destination list is added to the file encryption via OpenPGP. The file is then chunked. Each chunk is transmitted individually to a randomly selected datastore node over LEDBAT over TLS over UDP. What this means is that a TLS-encrypted channel for UDP datagrams is created between Alice and each datastore node, and peer-to-peer upload of each chunk then occurs.

Within the datastore itself, all transactions between nodes are automated and secured with session keys negotiated at transaction time. Each node has a persistent address, but the keys for onion routing are merely temporary and last for exactly one circuit, over which multiple data chunk transfers might occur if the same destination is selected multiple times..

Efficient uploading to temporary data storage and crude randomized load balancing

Each person who is allowed to upload data is also running their own storage node. For example, Alice's .onion address indicates (from the viewpoint of the datastore nodes) her storage node and, from the viewpoint of other clients connected to the network, the public key to which data destined for Alice should be hashed with.

The choice of destination points for file chunks is as random as possible given certain factors like storage space available on the chosen nodes and number of nodes currently on the network. We select a simple random sample of *n* destinations, where *n* is the number of file chunks. We re-select from the list of nodes currently up in the case that a given possible destination's storage space is occupied up to the high-water mark. It is possible that a chunk of Alice's file could end up in Alice's storage node, though hopefully by using multiple random number generators and several selection steps, most of the file will be stored elsewhere.

Destination points are randomly assigned, using a different random number generator from the one used for node selection, from the list to chunks of data. Ideally, a mixture of data chunks from many sources would be stored on Alice's node. Once upload is complete, the metadata store returns a confirmation to Alice as well as a reasonably current estimate of service-wide statistics.

Next, each chunk of data makes at minimum one more randomized hop. The initial destination point *n* for a chunk of data *d* will randomly select another node *m* from the remaining available nodes (if the chunk comes from anywhere that isn't another node in the datastore), negotiate a set of session keys, and construct an onion-style circuit to *m*. The data chunk is sent over the circuit and the session keys are discarded. If *m* has reached its highwater mark, another secondary destination is chosen by *n* prior to key negotiation.

When Bob wants to retrieve the file Alice has left for him, he issues a GET$\langle idA, idB, .onion_B \rangle$ to the key,value store. The store returns the .onion where each file chunk is located, and the complete file's hash with the provided identifiers $h(h(f), idA, idB)$. Bob's client is now armed with the information needed to retrieve the full file.

Bob makes a GET request to each .onion address provided along with his hash, which starts a *trackerless* peer-to-peer download from each relevant node of the rendezvous directly to Bob of the encrypted file. As Bob's storage node acquires each file chunk, he checks against his hash to ensure he's got it all correctly. Potentially the rendezvous point already did the same when Alice was uploading the file as an integrity check. This also would eliminate the possibility of Alice sending a file to the rendezvous point that does not match the hash she already registered with the network.

Download procedure and probability

Node entry, exit, and information loss

Whenever Alice has spun up her contribution to the network (her storage node), she is allowed to upload her onion address and list of files on offer. Thus, node entry can happen at any time, at the client's will. Node exit, similarly, is random, depending upon how long the user wants their node operational. Ideally, they would leave their storage node up all the time and ensure greater storage capacity as well as randomness for the network, but this might not always be the case.

Information loss could naturally occur when a node unexpectedly drops out of contact with the rest of the data store. However, we know that having perfect replication of all data across all nodes is unsustainable. In the future we plan on benchmarking to determine the proper amount of data redundancy across nodes. Due to the nature of the network, the redundancy will also be randomized; but no two pieces of information that are the same should be stored on the same node.

Each node, upon selection by the metadata store as a potential initial location for a file chunk, is checked to see if the amount of storage space available is less than that node's declared highwater mark. Each highwater mark is unique to the amount of storage space initially obtained for a given node by its owner.

Virtual private storage

Rendezvous points will mostly be run by volunteers who use the service regularly. This model could easily be extended to multiple stores which are updated through P2P every time someone uploads their information tuple to the store most convenient to them.

A shared-nothing (no storage space of any kind is common to any two nodes) multiple-node model would provide both greater fault tolerance and additionally reduce the number of hops needed to query the network, hopefully adding efficiency. We also propose a study on what proportion of rendezvous points to regular users to key,value stores would make this architecture most efficient.

4. POTENTIAL THREATS

5. CONCLUSIONS AND FUTURE WORK

We present some justification for and description of a service that allows anyone to protect the privacy of their data while sharing it as desired.

6. REFERENCES

[1] M. Alsabah, K. Bauer, T. Elahi, and I. Goldberg. The path less travelled: Overcoming tor's bottlenecks with traffic splitting. In *Proceedings of the 13th Privacy Enhancing Technologies Symposium (PETS 2013)*, July 2013.

[2] M. J. Bach. *The Design of the UNIX Operating System*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1986.

[3] D. Balfanz, G. Durfee, D. Smetters, and R. Grinter. In search of usable security: five lessons from the field. *Security Privacy, IEEE*, 2(5):19–24, Sept 2004.

[4] M. Belshe, R. Peon, and M. Thomson. Hypertext transfer protocol version 2. In *HTTPbis Working Group*, Fremont, CA, USA, July 30, 2014. IETF (Internet Engineering Task Force).

[5] T. Brewster. Swedish researchers uncover dirty tor exit relays. `http://www.techweekeurope.co.uk/workspace/malicious-tor-exit-relays-136828`.

[6] S. Chakravarty, M. V. Barbera, G. Portokalidis, M. Polychronakis, and A. D. Keromytis. On the effectiveness of traffic analysis against anonymity networks using flow records. In *Proceedings of the 15th Passive and Active Measurements Conference (PAM '14)*, March 2014.

[7] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *International Workshop on Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability*, pages 46–66, New York, NY, USA, 2001. Springer-Verlag New York, Inc.

[8] B. Cohen. The bittorrent protocol specification. `http://www.bittorrent.org/beps/bep_0003.html`.

[9] N. Cubrilovic. Analyzing the fbi's explanation of how they located silk road. `https://www.nikcub.com/posts/analyzing-fbi-explanation-silk-road/`.

[10] C. Dewey. This debunked kickstarter project may be the biggest crowdfunding fail to date. *The Washington Post*, October 16 2014.

[11] R. Dingledine, N. Mathewson, and P. Syverson. Deploying low latency anonymity: Design challenges and social factors. `http://www.dtic.mil/dtic/tr/fulltext/u2/a527761.pdf`.

[12] R. Dingledine, N. Mathewson, and P. Syverson. Rendezvous points and hidden services. `https://svn.torproject.org/svn/projects/design-paper/tor-design.html#sec:rendezvous`.

[13] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 21–21, Berkeley, CA, USA, 2004. USENIX Association.

[14] R. Dingledine and S. J. Murdoch. Why tor is slow and what we're going to do about it. `https://svn.torproject.org/svn/projects/roadmaps/2009-03-11-performance.pdf`.

[15] C. Egger, J. Schlumberger, C. Kruegel, and G. Vigna. Practical attacks against the i2p network. In *Proceedings of the 16th International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2013)*, October 2013.

[16] A. J. Feldman, W. P. Zeller, M. J. Freedman, and E. W. Felten. Sporc: Group collaboration using untrusted cloud resources. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, Berkeley, CA, USA, 2010. USENIX Association.

[17] R. Geambasu, A. Levy, T. Kohno, A. Krishnamurthy, and H. M. Levy. Comet: An active distributed key/value store. In *OSDI*, 2010.

[18] J. Geddes, R. Jansen, and N. Hopper. How low can you go: Balancing performance with anonymity in tor. In *Proceedings of the 13th Privacy Enhancing Technologies Symposium (PETS 2013)*, July 2013.

[19] A. Greenberg and L. Poitras. These are the emails snowden sent to first introduce his epic nsa leaks. `http://www.wired.com/2014/10/snowdens-first-emails-to-poitras/`.

[20] P. Gutmann and I. Grigg. Security usability. *IEEE Security and Privacy*, 3:56–58, 2005.

[21] S. Han, V. Liu, Q. Pu, S. Peter, T. E. Anderson, A. Krishnamurthy, and D. Wetherall. Expressive privacy control with pseudonyms. In D. M. Chiu, J. Wang, P. Barford, and S. Seshan, editors, *SIGCOMM*, pages 291–302. ACM, 2013.

[22] F. House. 2014: Freedom on the net report. `https://freedomhouse.org/report/freedom-net/freedom-net-2014#.VIHcFOrOpp9`.

[23] U. N. International Community. The universal declaration of human rights. 1948.

[24] R. Jansen, F. Tschorsch, A. Johnson, and B. Scheuermann. The sniper attack: Anonymously deanonymizing and disabling the Tor network. In *Proceedings of the Network and Distributed Security Symposium - NDSS '14*. IEEE, February 2014.

[25] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson. Users get routed: Traffic correlation on tor by realistic adversaries. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 337–348, New York, NY, USA, 2013. ACM.

[26] jrandom (Pseudonym). Invisible internet project (i2p) project overview. Design document, August 2003.

[27] N. Kerris and T. Muller. Apple media advisory: update

to celebrity photo investigation. http://www.apple.com/pr/library/2014/09/02Apple-Media-Advisory.html.

[28] E. Kovacs. Operation against tor dark markets raises security concernts. http://www.securityweek.com/operation-against-tor-dark-markets-raises-security-concerns.

[29] M. F. Lee. Onionshare.

[30] J. Li, M. Krohn, D. Mazières, and D. Shasha. Secure untrusted data repository (sundr). In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, Berkeley, CA, USA, 2004. USENIX Association.

[31] Z. Ling, L. Junzhou, K. Wu, W. Yu, and Z. Fu. Torward: Discovery of malicious traffic over tor. In *Proceedings of Infocom: IEEE Conference on Computer COmmunications*. IEEE, 2014.

[32] D. Macmillan and D. Yadron. Dropbox blames security breach on password reuse. *Wall Street Journal: Digits Blog*.

[33] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 53–65, London, UK, UK, 2002. Springer-Verlag.

[34] G. Norcie, J. Blythe, K. Caine, and L. J. Camp. Why johnny can't blow the whistle: Identifying and reducing usability issues in anonymity systems. In *Proceedings of the 2014 Workshop on Usable Security (USEC)*, February 2014.

[35] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. Cryptdb: Protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 85–100, New York, NY, USA, 2011. ACM.

[36] T. T. Project. https://blog.torproject.org/blog/call-arms-helping-internet-services-accept-anonymous-users.

[37] T. T. Project. Tor metrics portal. metrics.torproject.org.

[38] E. S. Raymond. *The Art of UNIX Programming*. Pearson Education, 2003.

[39] M. Ripeanu. A peer-to-peer architecture case study: The gnutella network. In *Proceedings of the First International Conference on Peer-to-Peer Computing*, P2P '01, pages 99–, Washington, DC, USA, 2001. IEEE Computer Society.

[40] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, Middleware '01, pages 329–350, London, UK, UK, 2001. Springer-Verlag.

[41] B. Schneier. More crypto wars ii. https://www.schneier.com/blog/archives/2014/10/more_crypto_war.html.

[42] R. W. Stevens and S. A. Rago. *Advanced Programming in the UNIX(R) Environment (2Nd Edition)*. Addison-Wesley Professional, 2005.

[43] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Comput. Commun. Rev.*, 31(4):149–160, Aug. 2001.

[44] R. Sumbaly, J. Kreps, L. Gao, A. Feinberg, C. Soman, and S. Shah. Serving large-scale batch computed data with Project Voldemort. In *Proceedings of the 10th USENIX conference on File and Storage Technologies*, FAST'12, Berkeley, CA, USA, 2012. USENIX Association.

[45] P. F. Syverson, D. M. Goldschlag, and M. G. Reed. Anonymous connections and onion routing. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, SP '97, Washington, DC, USA, 1997. IEEE Computer Society.

[46] Tails. Tails: the amnesiac incognito live system. https://tails.boum.org/.

[47] J. P. Timpanaro, I. Chrisment, and O. Festor. A bird's eye view on the i2p anonymous file-sharing environment. In *Proceedings of the 6th International Conference on Network and System Security*, Wu Yi Shan, China, November 2012.

[48] Tor. Tor project: Faq. https://www.torproject.org/docs/faq.html.en#FileSharing.

[49] uTorrent. Elegant, efficient torrent downloading. www.utorrent.com.

[50] A. Whitten and J. D. Tygar. Why johnny can't encrypt: A usability evaluation of pgp 5.0. In *Proceedings of the 8th Conference on USENIX Security Symposium - Volume 8*, SSYM'99, pages 14–14, Berkeley, CA, USA, 1999. USENIX Association.

[51] Z. Wilcox-O'hearn. Freenet compared to tahoe-lafs. http://emu.freenetproject.org/pipermail/tech/2012-March/003093.html.

[52] Z. Wilcox-O'Hearn and B. Warner. Tahoe: The least-authority filesystem. In *Proceedings of the 4th ACM International Workshop on Storage Security and Survivability*, StorageSS '08, pages 21–26, New York, NY, USA, 2008. ACM.