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

Security

The First Amendment to the United States Constitution details certain liberties that Americans have come to expect in their daily lives. Among these are the prohibition of laws abridging the freedom of speech and interfering with the right to peaceably assemble. While the Internet is a multinational, borderless entity, it originated in the United States. Thus it may be the case that some of the morals and expectations of the American people have become deeply interwoven in the social fabric of the Internet. This is affirmed by the nearly worldwide availability of services such as Tor [12], Freenet [6], I2P [34], and so on, 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.

This expectation of freedom of speech in particular requires security and user privacy to 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 [29], it is much more difficult to compromise. In the interest of the continuation of a free and open web, 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 [2] [37] 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 [26].

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 [20] of such services that have occurred in the last few months, spreading the 'private' data of public figures across the internet. 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 [10], we argue that it is possible to do better [16], 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 [15], so it is logical to use as much layered, strong encryption as we can possibly 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 [23], 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: there are more parts that might potentially fail. Furthermore, the simpler and more sturdy a system and its parts are [30], 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 [26] as our final underlying design concern. Each component should do just one thing [1], 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 [18] [21] [4] [8] 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.

2. METHODS AND ARCHITECTURE

Our system has two main parts: a public key,value store similar to Tor relay directories, and a (secure) temporary data rendezvous point [11] [25] which would likely be a thin layer of encryption and randomization for data over storage space located on VPS platforms and contributed by volunteers or as a requirement for uploading data in the first place (i.e. the "pay to play" model). These two parts will both operate on the P2P-style assumption that *data is only sent when it is asked for by the receiver* [3].

Key storage and distribution

The storage of clients' public keys mapped to available files is handled by the key,value store, 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 (analogous to an HTTP PUT) are allowed to run an operation analogous to HTTP GET to see if their public key is a match against any data available in the rendezvous.

Suppose Alice wants to send a file to Bob. She first registers an identifier [17] which, for our purposes, consists of a PGP public key either previously in use and uploaded to Alice's client application, or generated for her by the client. A .onion address [32] (that isn't currently in use) is calculated by way of taking a shortened SHA-3 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 the public key(s) of her file's recipient(s), she can add them to the hash of her key with the file she is publishing. Data given a destination in this fashion cannot be reassembled and pulled out of the store without

a public key that, hashed with some file and origin point, matches the hash value attached to the file (and that data additionally cannot be decrypted without the matching private key). Data cannot even be *found* without knowledge of the public key belonging to its origin.

There is some possibility for malicious users who have their own data in the store to pull out random public keys from the directory and try to get those users' files, but as long as the malicious users' keys don't match the destination hatch, 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 is 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, her full public key, the hash $h(f)$ of the file f she would like to send, and a list of hashes of her onion address, the hash of the file, and the public key(s) of any recipients are sent

```
PUT { id : id_A,
      address : .onion_A,
      filehash: h(f),
      dest: { h( h(f), .onion_A, .onion_B),
              h( h(f), .onion_A, .onion_C),
                   ...
            }
      TTL : x
    }
```

to our public-facing key,value store. TTL, which stands for time to live, is a concept borrowed from 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 written over. The default TTL will be one. If the TTL is set to -1, the destination list must also be blank. If both of these occur, the file will be considered open for public download until its originator sends a DELETE. 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.

Efficient uploading to temporary data storage and crude nearest neighbor load balancing

When Alice's public key and the hash(es) of her files with their destination(s) are loaded into the key,value store, the actual data upload to the rendezvous can take place.

The store chooses a set of destination points from its list of currently available rendezvous(es). Since 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 sent. The choice of destination points is made by considering the "nearness" of a node's public key hash to the hash of Alice's public key. Nearness is determined by measuring the cosine similarity s (a decimal such that $0 \leq s \leq 1$) between Alice's onion address and a simple random sample of other onion addresses in the database, all interpreted as bit vectors. If Alice's address A_a and some randomly selected address R_a are being compared for similarity, we calculate

$$s(A_a, R_a) = \frac{A_a \bullet R_a}{\|A_a\|\|R_a\|} \tag{1}$$

where $\bullet$ indicates the dot product of the two addresses represented in binary form considered as bit vectors, and $\|vector\|$ indicates the length of *vector*.

The nearest x nodes (where x is the number of shards the file has been broken into after being encrypted with Alice's key) are chosen to populate the set of destination points for the file. A quick available space check is performed on each of the x nodes before any data is uploaded. If any of these nodes are full, a new set of x are chosen and the nearest of these replace the "full" nodes currently in the destination list.

Destination points are then randomly assigned from the list to shards of data. We ensure that the measure of nearness is not absolute, such that Alice is not storing the entirety of her file on her own node, even if her onion address is part of the random sample. Ideally, a mixture of data chunks from many sources would be stored on Alice's node. Once upload of all shards is complete, the key,value store returns a confirmation as well as a reasonably current estimate of service-wide statistics.

If we would like the file Alice has uploaded to the rendezvous point to persist for other downloads, we propose adding another value to Alice's initial PUT tuple, something analogous to a TTL value that gets set to "infinite" or decremented with each download. Once the TTL is zero, the rendezvous may write over the memory allocated to the file with random data, and then report that space as free, or as an alternative it will simply report that space as free and write over it in the course of hosting other anonymous P2P downloads. If Alice chooses a TTL greater than 1 in preparing her information for upload, perhaps she contatenates a dummy id_B that multiple people have the ability to replicate.

When Bob wants to retrieve the file Alice has left for him, he issues a GET$\langle id_A, id_B \rangle$to the key,value store. The store returns the .onion where the file is located, and the hash/checksum of the file. Bob now is armed with all the information he really needs to get his file.

Bob opens a GET$\langle id_A, id_B$, hash $\rangle$ on the .onion address of the rendezvous, which starts a peer-to-peer download from the rendezvous to Bob of the encrypted file. As Bob reaches certain points in the file, 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 to the network.

Layered encryption

PGP

Tor and the onion protocol

The Tor system [12] is a widely used, well documented project claiming to provide more security than its users would have otherwise conducting business on the open web. The Tor FAQ [35] 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. Not implementing our service over Tor gives us the freedom to use UDP at the transport layer where desired and also to employ specially tailored peer-to-peer protocols for communication among our "directory servers" and also 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 [8]. Tor hidden services, by way of a "hidden service descriptor", which consists of the service's public key and a summary of its introduction points, 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 uses an incremental path-building design, which means that 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 temporal modularity in our design; transport of a given chunk of data within the database is 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.

In direct contradiction to methods employed by Onionshare [22], Anonabox [9], and similar systems overlaid on the existing Tor system, we rolled our own underlying system which integrates only the features we really needed from Tor, mainly the hidden service/rendezvous point model, and the secrecy given by the latest iteration of onion routing.

We argue that the slowness of Tor compared to the open web is due more to an imbalance between users and contributors than to overhead from the anonymity precautions. Due (we hypothesize) to the Tor system's notoriety in mainstream media and additionally the general public's lack of understanding of what Tor is and how it works, the percentage of those using Tor and similar systems for anonymity who also contribute back by running a relay or an exit node or improving the codebase is much smaller than the number of people who want to use the service.

Furthermore it has been shown [5] 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.

Despite hope that Tor might one day be more robust (as it stands, it is one of the most well-known free anonymity services), we cannot count on volunteers signing up to support Tor soon enough or in large enough number to meet demand for anonymous information sharing that requires less bandwidth *and* additionally support the general public sharing files over Tor as over tracked torrent clients.

Similarly to Tor, I2P [19] is an anonymous network designed to sit below typical internet functions such as email, HTTP, file sharing, and so on, though it is message-based and one must use libraries which allow TCP and UDP traffic on top of it if desired. Like Tor, I2P encrypts all communication end-to-end. I2P also employs El Gamal for eepSite to eepSite dialogue where Tor uses RSA for hidden service to hidden service communication, and operates over a different routing scheme: "garlic" to Tor's "onion".

Single-peer and peer-to-peer transactions

Cooperative peer-to-peer services like BitTorrent [7], Gnutella [27], uTorrent [36], Freenet [6], and more experimental distributed architectures attempt to meet this goal. 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 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 the peer-to-peer model such as Pastry [28], Chord [31], Kademlia [24], Comet [14] and so on have advantages over widely used peer-to-peer services including greater consistency and fault tolerance, but none provide the kind of intrinsic, heavy-duty modular security we envision.

Control interface: the client

We propose a dual model for the client interface: a minimal browser extension that will work on both ordinary Firefox and the Tor browser bundle (which is based on Firefox), and a web site optimized for touchscreen (as well as keyboard) use. We do not wish to create actual client-side applications that run at a deeper layer than appended to the browser, if possible, to minimize any need for porting across platforms.

Rendezvous point operators should be able to see how much of their total space is being used, how many P2P connections they currently have open, and how much bandwidth that's taking up. Furthermore, the key,value store operator will have additional functionality (similar to that of the rendezvous operator) that allows him to keep an eye on the store and make sure it isn't being overwhelmed.

A snapshot of this information could be provided in JSON format from the key,value store as part of the response to adding oneself to the network in order to upload a file, or it could be available only upon request in keeping with the theme that no action of automated parts of this system shall happen without a single, authorized human request to facilitate it.

Control interface: GET, PUT, DELETE

Individual-node space checks and the highwater mark

Efficient retrieval of "sharded" encrypted data

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 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.

3. PRELIMINARY RESULTS

4. 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.

5. REFERENCES

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

[2] 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.

[3] 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).

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

[5] 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.

[6] 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.

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

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

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

[10] 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`.

[11] 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`.

[12] 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.

[13] 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.

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

[15] 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/`.

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

[17] 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.

[18] 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.

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

[20] 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`.

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

[22] M. F. Lee. Onionshare.

[23] 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.

[24] 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.

[25] 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.

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

[27] 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.

[28] 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.

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

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

[31] 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.

[32] 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.

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

[34] 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.

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

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

[37] 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.