A Peel of Onion

Paul Syverson
Center for High Assurance Computer Systems
U.S. Naval Research Laboratory
Washington DC, USA
syverson@itd.nrl.navy.mil

ABSTRACT

Onion routing was invented more than fifteen years ago to separate identification from routing in network communication. Since that time there has been much design, analysis, and deployment of onion routing systems. This has been accompanied by much confusion about what these systems do, what security they provide, how they work, who built them, and even what they are called. Here I give an overview of onion routing from its earliest conception to some of the latest research, including the design and use of Tor, a global onion routing network with about a half million users on any given day.

1. WHY ONION ROUTING

We (David Goldschlag, Michael Reed, and I) began work on onion routing in late 1995 with the goal to separate identification from routing. Onion routing has often been said (by ourselves and others) to provide anonymous communication, but 'anonymity' can be taken in many ways, and misunderstanding about that has led to significant missteps in system design and analysis. To quote from our paper at ACSAC '96, "Our motivation here is not to provide anonymous communication, but to separate identification from routing. Authenticating information must be carried in the data stream. Applications can (and usually should) identify themselves to each other. But, the use of a public network should not automatically reveal the identities of communicating parties. The goal here is anonymous routing, not anonymity." [43].

If I need to log into the workstation in my office at the Naval Research Laboratory from a remote location, I want to be sure that I am connecting to the right system before I start entering my username and password. Similarly, I want the system to be sure it's me before granting access. Nonetheless, I may not want, e.g., the wireless access point at my remote location or anyone in range of it to know that someone is connecting to NRL from there. Nor might I want this to be known by all the other elements on the communication path from me to my workstation.

In other cases, however, I may want to hide my identity from the system at the far end. For example, if I am reviewing a submission for publication or a proposal from someone, where review is to be anonymous and I need to check something at the submitter's web site, I would like to do so without revealing my identity to the web server. What I mean by 'identity' we will revisit more thoroughly below. But if I am originating the connection at my office workstation, it might include my IP address as well as anything else that could indicate who the reviewer is or possibly broader aspects, such as that the reviewer is from the Navy. This could be indicated by either things I type or things that applications send through the connection, for example, cookies. But again, the goal is to separate the choice to identify or not from the routing necessary for communication to take place.

2. WHO ONION ROUTING

Onion routing systems have a wide variety of users with a variety of purposes. Tor is the most well known and widely used onion routing system. As of the time of writing it serves over half a million concurrent users over a network infrastructure comprised of about three thousand onion routers of one type or another [53].

I have already described two ways users might want to protect themselves using onion routing. There are many types of users and many uses for onion routing. Law enforcement and intelligence agencies need to operate on the Internet without revealing their activities or intentions to those they are investigating or anyone else observing. And the "road warrior" mentioned above is protected both from profiling of his online activities and affiliation, and from potential physical harm by an adversary who can easily and discretely associate these with the hotel he is staying in [19]. Road warriors typically use Virtual Private Networks to help protect their communications. But VPNs provide limited protection against this threat if it is easy to associate users or affiliations with a VPN, and the adversary can observe connections to the VPN. Also, while onion routing distributes trust as we shall see, externally provided VPNs typically constitute a single point of trust and thus a single point of failure for protecting routing information.

Penetration testing engineers have found that if they have a limited IP space from which to launch attacks, defenders have been able to use this to spot attacks. Onion routing can help in this regard, although without additional safeguards this would make onion routing networks appear to be a vector for abuse [18].

Victims of various types of diseases, as well as of crimes and abuse have used onion routing to do research and to chat

ACSAC '11, December 5–9, 2011, Orlando, Florida USA
2011 ACM 978-1-4503-0672-0/11/12.

with fellow victims without risk of exposure [15]. Besides the more obvious information security protections these afford, the ability to speak freely and anonymously has long been recognized as an important therapeutic benefit.

Many ordinary citizens simply want to limit the amount of information about themselves that is being gathered every time they go online. How this information will be exploited by businesses or criminals is often not yet understood at the time it is gathered, so prudence counsels prevention. Or they may want to protect their children from being geolocated by those observing or engaging the children in online activity—geolocation that can take place simply from the act of connecting even before the child has entered anything into the application.

This is just a small sampling of the types of onion routing users. Central to the protection that onion routing provides is that all of these different types are sharing the network. Otherwise the problem noted above for VPNs would apply to onion routing as well. Anything coming out of, for example, a Navy-only onion routing network would be known to be coming from the Navy and anything entering it would be known to be headed to the Navy: this would not adequately separate identification from routing. But the diverse users needed to provide this protection also have diverse trust values. Thus the entire network infrastructure cannot be provided by or under the control of a single entity. And since those running the network will similarly have diverse trust, they must be able to examine for themselves the code that they run, or at least be sure that independents whom they trust can do so. These points were part of our vision for onion routing from the very beginning, and we obtained our first publication release for onion routing code in 1996, before 'open source' was a generally adopted concept.

Besides needing to adequately trust the system, both users and providers must want to use and/or provide the system. This was addressed in the papers cited above. But even before then, it was the central topic of "On the economics of anonymity" [1]. From the perspective of users, a basic dilemma arises from the difficulty of trying to be anonymous by yourself. Unlike other security properties, those who need strong protection will not be early adopters because the system will then be known to be protecting the relatively small set of most sensitive users. So merely connecting or sending to it will be revealing of something they will probably want to hide. Similarly any destination of traffic from the system is revealed as something of interest to one of those most sensitive users. Contrastingly, the less sensitive user will have inadequate incentive to pay for the system. This means that a certain degree of free riding must be built into the system. On the one hand, this means that less sensitive users will be getting protection virtually for free. On the other hand, by their use of the system they effectively provide better protection for the more sensitive users. So it is not free riding from the security perspective.

But even just looking at user cost, it is not really entirely for free. The security onion routing provides comes with some inevitable usability and performance overhead. We will get more into how it works below, but for now it is enough to note this and to note its impact on security. Back et al. have stated that "In anonymity systems usability, efficiency, reliability and cost become *security* objectives because they affect the size of user base which in turn affects the degree of anonymity it is possible to achieve." [4]. In fact perceived usability by others is also a factor because it affects expectations about how likely others are to use the system and provide us with cover. In a sense, a system that is not as secure by some analytic criterion may be more secure in fact if opinion drives more users to that system [15]. We will revisit such seemingly paradoxical issues below when we discuss security definitions and metrics.

3. HOW ONION ROUTING

Before getting into defining 'security', we should spend a little time defining 'onion routing' and describing how it works because there has been a surprising degree of confusion about what onion routing is and is not.

Symmetric-key cryptography is generally much cheaper computationally than public-key cryptography is. But symmetric-key cryptography only works if both communicating parties share a key. An advantage of public keys is that Alice can send an encrypted message to Bob without having to share a key with him first. A primary application of public-key cryptography is thus in protocols to establish a shared (symmetric) key for Alice and Bob to use in a communication session. If a large amount of data is sent in the session encrypted with this shared key, the computational overhead is much less than if public keys were used throughout the session. There are many other benefits to establishing a shared session key, but we focus on this feature for now.

Onion routing is in one sense just a generalization of this basic idea. It can use the more computationally expensive public-key cryptography to lay a cryptographic circuit of shared symmetric keys along an unpredictable route. The circuit can be bidirectional, and each session key is shared between the initiating client and one onion router in the path. Data that is sent by the client is wrapped in layers, except that the layers are created using the shared symmetric keys rather than public keys. Using the symmetric key it learned when the circuit was laid, each onion router removes a layer of encryption as the data passes. In this way the data emerges at the end of the circuit as plaintext (from the perspective of the onion routers. It could be ciphertext if that is what is being sent over the circuit by the application.) Data passing back from the responder, for example from a web server, has a layer of symmetric crypto added to it as it passes through each onion router node in the path. It thus arrives at the circuit initiator as a layered data structure. The circuit initiator set up the circuit and so is able to remove the layers using the symmetric keys for that circuit.

Onion routing was invented to facilitate anonymous low-latency bidirectional communication, such as occurs in web browsing, remote login, chat, and other interactive applications. By only using public-key cryptography to establish session keys it allows for throughput and latency that would not be feasible if public-key operations were needed for each message (or packet) passing through the system. By following a multihop free-route path selection through a network of independently managed onion routers, it makes it hard for an adversary to observe traffic entering and leaving the system.

3.1 An onion by any other name would smell as sweet

In onion routing, application data is passed inside a layered data structure as just described. In the first two generations of onion-routing design [27, 44, 50] that emerged from NRL, the circuit was also laid by means of a layered data structure, with two primary differences. (1) Each layer also provided the material to generate the symmetric keys used for passing the data back and forth. (2) The final layer contained neither a destination address nor meaningful content to be transmitted. Thus, abstractly the onion for a three-hop route through R_1, R_2, R_3 looked like this:

$$\mathrm{E}(\mathrm{PK}_{R_1}, [K_1, R_2, \mathrm{E}(\mathrm{PK}_{R_2}, [K_2, R_3, \mathrm{E}(\mathrm{PK}_{R_3}, [K_3, \mathrm{Pad}]\ )]\ )]\ )$$

where 'PK_{R_i}' is a public key for R_i, and 'K_i' is session-key material to be shared between the route originator and R_i. We are ignoring some details here to focus on these two features of the structure. In particular, note that what is encrypted for the last layer is just the symmetric-key material to be used for passing data back and forth and then just empty padding. Thus, there is no message at the innermost core of the onion: the onion is the layers themselves, much like the root vegetable from which it gets its name. Indeed, this structure is the basis for the choice of the name "onion routing": the network routes *onions*—not just layered data structures, but data structures that are comprised of layers and nothing else. Using "onion" to describe a layered data structure is not unique to onion routing or to structures specifically comprised of nothing but layers. Independently and contemporaneously with the first descriptions of onion routing, "onion" was used by Gülcü and Tsudik to describe a structure somewhat like the ones above for a mix-based[1] remailer network [28], and there may be even earlier uses. We are not averse ourselves to referring to the packets being sent over the cryptographic circuit as "data onions".

Furthermore, though the circuit-building onions gave onion routing its name, they are not what makes it onion routing. The essential feature is that public keys are used to lay a cryptographic circuit of symmetric keys, which is then used to pass data. Other onion routing designs that we will discuss later do this by effectively only sending one onion layer at a time through the partial circuit already created.

'Onion routing' has also at least occasionally been used in published research to refer to networks that do not lay a circuit at all. Though we will countenance somewhat varied use of 'onion' to cover "data onions" or "message onions", we will stick to using 'onion routing' for designs that lay cryptographic circuits—which was what those of us who coined the phrase had in mind.

It is instructive to explore the origins of this mistake in usage, but it requires that we examine the differences between mixes and onion routers. Mixes were invented by David Chaum in 1981 [9]. Briefly, a mix takes in a batch of packets or messages, reorders them, cryptographically changes their appearance, possibly adds new messages or previously delayed messages from an earlier batch, and possibly removes some received messages from the batch. It then forwards the resulting batch to other mixes or to ultimate destinations. The mix's operations make it difficult for anyone observing the mix to correlate observed input messages with observed output messages. Other than for purposes of contrast, we will not discuss mixes, which have many variations and an extensive literature in their own right [11, 21].

Mix networks get their security from the mixing done by their component mixes, and may or may not use route unpredictability to enhance security. Onion routing networks primarily get their security from choosing unpredictable routes through a network, and onion routers typically employ no mixing at all. This gets at the essence of the two even if it is a bit too quick on both sides. Other typical and highly salient distinctions include that all existing onion routing network designs are for carrying bidirectional low-latency traffic over cryptographic circuits while public mixnets are designed for carrying unidirectional high-latency traffic in connectionless messages[2]. Mixes are also usually intended to resist an adversary that can observe all traffic everywhere and, in some threat models, to actively change traffic. Onion routing networks are generally completely broken against an adversary who observes both ends of a communication path. Thus, onion routing networks are designed to resist a local adversary, one that can only see a subset of the network and the traffic on it.

Given the fundamental differences in the mechanisms they employ, the adversaries they are intended to resist, and their basic designs (not to mention typical applications), how could anyone who works in this area ever confuse the two? All deployed onion routing networks do use some form of layered encryption on traffic they carry, encryption that is gradually removed as it passes through the network. And this is also true of decryption mixnets (re-encryption mixes work differently). Thus there is a clear similarity between the two in at least this respect. Still, given the differences, how could this be enough to confuse an expert? The standard conception of security found in the literature—which in turn motivates system designs—assumes the goal of all systems that hide who is talking to whom is to make a given set of users (or user communications) less distinguishable. If one motivates design by starting with such a set and seeing how well the system obscures identification of its elements, the security contributions of an onion routing approach are harder to see. Any distinction between onion routing networks and mixnets, if recognized at all, is then likely to be couched only in terms of differences in intended application or engineering tradeoffs of security versus performance. Even we designers of onion routing systems have been occasionally guilty of falling into this idiom. We discuss adversaries and security definitions further in [48] and below in section 5. Our goal here was simply to make clear the difference between mix networks and onion routing networks.

Another confusion is that we have historically used 'onion routing' both to refer to the general approach described above and to refer to specific system designs that emerged from projects at NRL. To distinguish between these below, we will use 'onion routing' to refer to all onion routing systems and will use 'NRL onion routing' to refer to the specific designed and deployed versions of onion routing that have come out of the NRL onion routing projects, unless this is clear from context. References to one generation or another of onion routing will also be an indication that it is specifically one of the NRL-originated designs that is being discussed.

[1] We will say more about mixes presently.

[2] An exception is Web MIXes [6], which creates bidirectional circuits through mix cascades to carry public web traffic.

Though onion routers are not mixes, it is possible to have a combination, producing a subclass of onion routers (and of mixes). Specifically, in the second-generation design from NRL, some low-latency timed mixing was added [44]. Prior anonymizing networks had been based on mixing, and realtime mixing per se did not add much overhead. It was hoped that by introducing realtime mixing, it would be possible to explore its possible benefits and simplify security analysis at least with respect to an external passive attacker. It was also hoped that this would make onion routing more palatable to the community by making it more familiar. If mixing was to be anywhere near realtime, however, then it would be necessary to do some padding and/or rate limiting to avoid trivial timing correlations. Though a full-length padding scheme was immediately dismissed as too costly, padding and bandwidth limiting based on a sliding-window weighted average of prior traffic was considered potentially viable [51]. Just like the first and the third, this second-generation design of onion routing made use of TCP and stream ciphers on the links between onion routers as well as end to end. The mixing onion routers had to be low-latency, mixing only packets that had arrived during a very short prior interval. But, the use of TCP and stream ciphers also meant that mixing could only be of packets on different circuits going through an onion router: traffic on a given circuit was not mixed with itself so as to preserve order. (TCP (Transmission Control Protocol) is one of the two main ways that traffic is passed over the Internet. It provides guaranteed and in-order packet delivery at the expense of some timeliness. The other main protocol is UDP (User Datagram Protocol), which provides no such guarantee and is better for applications with tighter realtime constraints that can tolerate some lossiness, for example VOIP.) Though padding or other approaches might prevent most short-term passive eavesdropping, nothing practical that has been proposed to date is expected to be effective against an active attacker. Various schemes have been proposed to cope with active timing correlation attacks. The earliest was PipeNet, which was originally proposed in a cypherpunks mailing list post in 1995 [10], although it was never formally published. PipeNet essentially required that all sending be at a persistent constant rate. To fully avoid any correlation no sender or recipient can join or leave the network once it is formed, and the entire network must be shut down as a single originator of communication stops sending. Though not all subsequent proposals are as draconian, and even ignoring the padding overhead, those that aren't are generally ineffective: they impose too much delay, require that the responder be an active participant in the scheme, or some combination of these.

By the time the third-generation onion routing design was begun, nobody had proposed a practical traffic-shaping scheme that wasn't fundamentally weak in some way. Indeed that remains true to this day. An unknown research breakthrough remains possible; however, all thought of making onion routers function as mixes was abandoned when the third-generation design was begun until such time as this could be shown to have any value against realistic adversaries [17].

4. WHICH ONION ROUTING

This paper was invited to accompany an ACSAC classic paper talk. But, I must confess that I am not sure which classic onion routing paper prompted the invitation. Within a half year of publishing the first onion routing design at the Information Hiding Workshop in May of 1996 [27], we published another paper at ACSAC that first described separating a client proxy that constructs the circuit from the onion routers that can carry circuits from many clients [43], and the following May we presented another paper that introduced many of the second-generation features most closely associated with onion routing [52].

I have alluded above to three generations of onion routing. I mean by this the three generations of onion routing design done by U.S. Naval Research Laboratory (NRL) researchers and various people contracted to work with them—most notably for the third-generation (Tor), Roger Dingledine and Nick Mathewson of what is now the Tor Project. There are many differences between the generations and difference from other flavors of onion routing besides those that came from NRL. Most of these we will ignore for now, but some of the primary distinctions between the three generations I mentioned are the following.

First-generation onion routing:

- **fixed-length, five-node circuits:** Onion routing was intended to allow anonymous connections to Internet servers, such as web servers, that were not part of the onion routing network. But, a configuration that was also intended for protection was communication between two enclave firewalls, on both of which were onion routers. If routes were three hops, a hostile middle node would immediately learn both source and destination enclaves. If routes were four hops, someone compromising the node adjacent to the source or destination enclave would immediately know which other node to attack to learn an entire circuit and thus who was talking to whom. By having five hops in a circuit this adversary would not have this information unless it attacked another node first.

- **integrated configuration:** A client, as described above, and an onion router are fully combined. Client applications could proxy through onion routing without participating in the network or running any software. But, all elements of onion routing communication are essentially between peers, and building of routes is entirely determined by and under the control of the first onion router in the circuit.

- **static topology:** There were no provisions for topology change or network discovery in the first-generation design. It was assumed that there would be a modest size network of nodes (perhaps twenty to one hundred) run by stable organizations that were not all mutually trusting of each other and that information about network configuration, signing keys, etc. would be handled offline. This was thus seen as at most a detail to be addressed later.

- **loose-source routing:** In case a node was unable to connect to the subsequent node in a circuit, e.g., because of underlying network problems, it could build its own onion through which it would pass the onion it was given. Thus, it would attempt to complete the circuit by adding more indirection to the route.

- **numerous application-specific proxies:** In the mid nineteen nineties applications were increasingly written to be proxy-aware as firewalls became more common; however, most applications did not yet use the SOCKS protocol that standardizes such interaction. For this reason, the first-generation of onion routing contained specific proxies for web traffic, for remote login, for email, and for other applications.

- **rendezvous servers and reply onions:** A rendezvous server allowed two anonymous circuits to mate. This permitted two parties that were anonymous from both the server and each other to communicate, e.g., in IRC chat. Reply onions built a circuit to connect back to a source that wished to remain anonymous. This could be used for replies to anonymous email or for providing access to a hidden web server. In this way someone who wished to provide a server that could be contacted while remaining anonymous could do so without the risks that affected Penet [29].

Second-generation onion routing:

- **running a client separated from running an onion router:** Perhaps the most important distinction from the first generation. This permitted an increased flexibility in how clients could manage trust and resources. Previously users would be forced to either provide a computer to participate in the onion routing infrastructure themselves or to trust some remote onion router. Now clients could learn about the network and could build their own routes without being required to route traffic for others themselves or trusting a remote computer with control over their routing and knowledge of it.

- **variable length circuits:** The second generation had variable length circuits (up to eleven hops within a single onion, although tunneling to even greater lengths was also possible). In addition to other advantages, this allowed access under different configurations to route within the network in essentially the same way. For example, whether onions were built on a client desktop or at an enclave firewall the rest of the path could be the same.

- **application independent proxies:** A more generic SOCKS proxy was added for those applications that could use it. There was also a redirector that would force all TCP traffic over the onion routing network. This required a kernel shim to be installed, ran only on Windows NT, and (unlike all other onion routing code of all generations) was not built for open source distribution.

- **entry policies and exit policies:** It was recognized that different individuals and organizations running onion routers might have different policy needs. They might want to only permit access to the onion routing network from within an enclave on whose firewall the onion router resided, or only permit exits to that enclave. Alternatively, they might permit email traffic (SMTP) to exit only to some locations and web traffic (HTTP) to exit to arbitrary locations. Thus individual onion routers could set their own entrance and exit policies [50]. This has significant economic impact on network participation and scalability, although the impact of separating clients from onion routers may be even greater [1, 15].

- **dynamic network state:** As in the first-generation, network topology is assumed to be predefined with each node knowing its neighbors. Networks were expected to either be cliques of at most several dozen nodes or, if needed, composed of such cliques with multiple bridges between them. Thus, basic network membership and acceptance of long-term keys were still assumed to be based on outside communication. Nonetheless, link state and other network information were maintained by flooding signed information to the "database engines" (DBE) attached to each onion router. The DBEs were also used to propagate exit policies so that originators would only build circuits to exit the onion routing network at nodes where the intended traffic would be permitted to exit.

- **mixing of cells from different circuits:** Realtime mixing of cells from different anonymous circuits was added. This permitted the exploration of timing, padding, and the like. It was, however, ultimately abandoned in the third-generation until there is at least some theoretical justification for hoping it will provide any benefit against an active adversary.

- **padding and bandwidth limiting:** Traffic is shaped according to a sliding-window weighted average of prior traffic to support protection against a passive eavesdropper doing traffic analysis. Also abandoned in the third-generation.

Third-generation onion routing (Tor):

- **Onion skins not onions: Diffie-Hellman based circuit building:** Perhaps the most important distinction from the first and second generations. As noted above, in the first two generations, circuits were built using an onion structure to distribute session keys. This meant that processed onions had to be kept track of at onion routers until they expired to prevent someone from replaying an onion many times, which might support various kinds of attacks. It also meant that, because session keys were distributed in the onion, anyone who captured an onion and all the (encrypted) traffic on a corresponding circuit could recover all the plaintext if the public keys that encrypted the onion were later broken or otherwise obtained. The onion-based key-distribution protocol was said to lack *forward secrecy*. By extending the circuit one hop at a time, effectively sending a single skin of an onion, session keys could be obtained using a Diffie-Hellman key establishment protocol. This protocol allows the communicants to exchange ephemeral public keys that can be combined with a private key of the other to establish a session key. The session key is never sent, even in encrypted form. This provides forward secrecy, meaning that, even if longterm keys are compromised at some point in the future, session keys will not be. Also, since different ephemeral keys are used at each hop of extending the circuit, there is no threat of replay so there is no need to keep track of

onions that have already passed. Interestingly, we considered but abandoned in the spring of 1996 [40] the option of using public Diffie-Hellman values to achieve efficiency gains in computation. Our intended design was to include the public DH-values from the originator inside the layers of circuit building onions, which were used in the first few generations of onion routing designs, and then to combine these with public DH keys (that we assume are DH-values used for generating keys). This is very similar to one of the protocols we introduced much later [42]. Our focus was not on forward secrecy but simply to be more computationally efficient. We were certainly aware of forward secrecy and intentionally chose a protocol for securing links between onion routers that provided it, but we only pursued it with respect to outside attackers rather than against compromised network nodes as well. The idea of using Diffie-Hellman for basic circuit building was simply another dropped design idea until work began on the Tor design, when it was picked up for the forward secrecy it provided and for freedom from the need to store onions against replay. The first description [27, 43] and implementation of onion routing uses RSA public keys for distributing circuit session keys and DH-established link encryption between the server nodes. The current version of onion routing, Tor, uses both a Diffie-Hellman key exchange and an RSA encryption/decryption for each step on the anonymizing tunnel setup. The computational advantages of using Diffie-Hellman that we contemplated in 1996 lay dormant until 2007, when we picked it up again, as did others [42, 34, 33].

- **Fixed-length three-hop circuits:** In Tor, primary emphasis is placed on what was dubbed in the second-generation, the "customer-ISP configuration" [44, 50] or more generally, "remote-COR configuration" [51], meaning that circuits are built by a local client and enter the onion routing network at some remote onion router after passing over some publicly visible network. This, coupled with an abandonment of any pursuit of mixing, reduces motivation for more than three hops to a circuit. And three hops is minimal to protect against two concerns. The first concern is that either the entry or exit node for the onion routing network might be sensitive, e.g., if it is on an enclave firewall. In this case two-hop routes have a single point of failure to link a sensitive source to a destination or vice versa. The second concern is that, in general, with two-hop circuits a compromised entry or exit would immediately know for each connection through it the single other point to attack to reveal the entire route. If the adversary has resources that can be readily mobilized for attacking at some of the nodes in the network when needed, two-hop circuits would make his job much easier than three-hop circuits, for which he would need to simply be lucky in knowing where to strike and when, or would need to keep his resources persistently mobilized everywhere.

- **Rendezvous circuits and hidden servers:** First-generation onion routing introduced rendezvous servers, for example to permit people to onion route to a chat server. And, hidden services were to be accessed via reply onions. Tor did away with onions per se and thus introduced a design for hidden services based on rendezvous circuits. On the one hand, this meant that one did not need to worry if the onion routers in a reply onion would be functional at the time the reply onion was actually used. On the other hand, one had to maintain open circuits to introduction locations at which the hidden server could be contacted. (Actual data is passed over a separate circuit to a rendezvous point in order to minimize the traffic load and the responsibility of the introduction points.)

- **Directory servers:** First-generation onion routing simply assumed network information was static or distributed offline. Second-generation onion routing assumed basic network membership information to be established offline but used a flooding mechanism to distribute authenticated network link state, keying information, and policy information necessary to build circuits. Third-generation onion routing introduced directory servers to distribute network status, but also network membership information. This is less complex, more flexible, and more scalable than flooding for maintaining a consistent picture of the network. On the other hand, this change requires a handful of trusted directory servers. Caching at other onion routers reduces the risk of a performance bottleneck, but this remains a trust bottleneck. Since the original design, Tor has evolved through several versions of directory structure to reduce the trust placed in individual directory servers, to increase resilience of the network to faulty or malicious directory servers, and to reduce overhead.

- **most application proxies eliminated as unnecessary:** SOCKS is a protocol originally designed to permit applications to communicate across firewalls in a flexible and controlled manner. A SOCKS proxy was added to second-generation onion routing, but other proxies were still needed because adoption of SOCKS was still limited. By the third generation, SOCKS had become common enough that applications could be assumed to use it and specific application proxies were no longer needed. Nonetheless, some issues remained. For example the most advanced version of SOCKS did not proxy DNS requests, only an intermediate version did. In addition, much network identification information was often passed by applications regardless of anonymization of the connection. For this reason, the first two generations of onion routing included sanitizing proxies to reduce this. By the time of third generation, sanitizing proxies were being developed and made freely available by others. Thus onion routing could focus on its primary job of anonymizing the circuits and use what others provided if application sanitization were needed. Note that sanitization should be kept a separate function. (If one were logging into work via SSH or a VPN over onion routing from a remote location, for example, then such sanitization is unnecessary and might hamper some authentication or functionality.) As in the directory system, Tor itself has evolved since 2004. In addition to SOCKS support as above, it now allows the use of transparent proxies (similar in function to the second-generation redirec-

tor for Windows NT), and it can process DNS queries as if a DNS server itself.

The very first onion routing design was peer-to-peer in the literal sense that all system elements interacted as peers. But it was not a P2P design in the sense that all end-user computers were expected to contribute to the infrastructure. P2P onion routing designs in that sense have also received a lot of attention. I simply cite a sampling of these designs without analysis [45, 24, 39, 36, 35]. So far, every P2P onion routing system has been shown within a year or so of publication to have one or another significant break. Having a completely decentralized design for node discovery and/or route propagation, whether or not the lookup is structured, is complex. I expect research in this area to remain both interesting and volatile for a while, but we shall see.

Both second and third generation onion routing have a client-server architecture, but where the server nodes are entirely locally managed and network membership and status is itself distributed. The second generation design for this was via a flat propagation of network information (that was never fully implemented and deployed). The current public Tor network has on the order of ten directory authorities that are independently managed and exist in multiple locations and jurisdictions. We have already mentioned this above, but here we note that having such an architecture can allow for significant scaling that is often cited as a goal of P2P design. Allowing clients to participate without having to provide servers means that many more can participate. Allowing this while still having a large diverse network provides performance and protection that has yet to be achieved by P2P designs [13].

We have not covered all of the features and differences of the three generations on onion routing designed at NRL, but we have hopefully covered the most salient basic distinctions. Distinguishing the names of the different designs also requires some explanation. The title of the Tor design paper as originally published is "Tor: The Second-Generation Onion Router"[17]. This was a misleading title in at least three respects, which we will now set out. We will also explain the origins of the name 'Tor'. To some extent, getting all this straight will help readers have a clearer understanding of onion routing and its history.

When the second-generation design above was begun, it was called "Onion Routing, The Next Generation" out of homage to a certain television series that was popular at the time, particularly with some of the onioneers. It was thus not only a second generation of onion routing, but one that was deliberately considered another generation by its designers. By contrast, when work on the Tor design began second-generation prototypes had been languishing for a few years without much attention. Tor began in a project originally intended to simply clean up and update the implementation of the second-generation software and tweak the design where needed for better performance and fault tolerance. However, it was soon recognized that a more radical redesign and re-implementation would more quickly lead to a better system overall. One of Tor's designers tended to use "first-generation" for all work on onion routing prior to when he started. Thus to him, Tor was second-generation. One of the designer's should have known better because he was there for all the history of all the generations. However, at the time he was inclined to not care about titles and go along with whatever his co-authors suggested for a title, not realizing that it would make describing the various systems more complicated in the future. Thus, though structurally Tor is clearly a third generation of basic design, the title of this paper would lead one to think otherwise.

The title is also confusing because it refers to a second-generation onion router, that is an onion-routing network component through which circuits are built and that forwards traffic along those circuits. But the paper is also about the onion-routing clients, the hidden service design, the directory system, etc., in other words an entire onion routing system design. Though onion routers are a central part of onion routing, describing them is a small part of the paper. In addition, by phrasing the title "Tor: The Second-Generation Onion Router", we gave an at least implicit impression that 'Tor' stood for 'the onion router'. This further exacerbated the confusion about the paper's scope and focus, but it is also simply incorrect.

When Roger Dingledine began work on onion routing as part of an NRL project, there were many other onion routing systems that had been or were being designed, published, and/or deployed, for example, Freedom [26], Cebolla [8], and Tarzan [24]. Thus, when he told people he was working on onion routing, they would ask him which one. He would respond that it was *the* onion routing, the original program of projects from NRL. It was Rachel Greenstadt who noted to him that this was a nice acronym and gave Tor its name. Roger then observed that it also works well as a recursive acronym, 'Tor's onion routing'. It was also his decision that it should be written 'Tor' not 'TOR'. Making it more of an ordinary word in this way also emphasizes the overlap of meaning with the German word 'Tor', which is gate (as in a city gate).

To sum up, "Tor: The Second-Generation Onion Router" is about the design of onion-routing systems, not just onion routers themselves. Tor is the third generation of onion routing, not the second. And the 'r' in 'Tor' represents 'routing' not 'router'. In hindsight we probably should have spent a bit more time on the paper title.

4.1 Freedom's just another word for no peel left to loose

As noted, within a short time after onion routing was invented, there were several onion routing designs developed elsewhere than at NRL. A significant contribution that fed back into NRL onion routing design evolution was Zach Brown's Cebolla [8]. As also noted above, perhaps the most significant discriminator of the third generation NRL onion routing design from second generation was the introduction of incremental circuit building via a Diffie-Hellman exchange between each node in the route and the client over the existing partially built circuit. The idea for this came from Cebolla.

The most notable onion routing network prior to Tor in terms of the scale of its development and deployment was the Freedom Network from Zero-Knowledge Systems, Inc. It was publicly announced in late 1998, and a deployed network ran from late 1999 till late 2001. This was a commercial system in which users paid for service. That itself is not without security implications, which we have already touched on briefly. (This is not the only commercial onion routing network that has existed. For example, IronKey has sold protected flash drives since about 2006, including one that supports private web browsing through IronKey's own

onion routing network.) We focus here on Freedom's network design and protocols. Like the first two generations of onion routing, the first version of Freedom built a route using an onion that passed the session keys to each of the onion routers—called Anonymous Internet Proxies (AIP) in the Freedom Network [26]. The resulting encryption-layered route was dubbed a "telescope" by the Freedom designers, after the nested tube structure of spyglass telescopes.

But Freedom had several important differences from the NRL designs for onion routing. It transported traffic using UDP rather than TCP. It used block ciphers since, unlike NRL onion routing, packets could not be guaranteed to be in-order. However, since many of the applications it carried required TCP guarantees, the Freedom client managed its own TCP stack (similarly for network exit nodes). This required a kernel modification and thus root access on the computer on which the program ran. A resulting tradeoff was that this gave Freedom tighter control over making sure that applications did not bypass the anonymity network than any of the NRL versions of onion routing except perhaps the redirector mentioned above. Freedom also provided a deployed pseudonymous mail infrastructure, including standard features such as spam control but also *nymservers* that would allow one to look up a pseudonym and obtain a reply block (essentially like the reply onions described above) and send reply traffic through the Freedom Network to the appropriate pseudonymous recipient. While the first two generations of NRL onion routing supported SMTP for sending mail and some sort of integration with reply onions was envisioned, none was ever fully designed much less deployed.

Probably the most important difference between NRL-design onion routing and Freedom was the pseudonym system. This was not simply to enable replies to email from Freedom clients, it pervaded the Freedom architecture. The underlying communications architecture was still an onion routing system that separated identification from routing and was thus an anonymity system in that respect, but the pseudonym system that ran on top of it was central to the users' interactions with the whole system. As the designers noted, you should "think of Freedom as a pseudonymity product, rather than as an anonymity product" [25]. Users were "encouraged to create pseudonyms ('nyms') for each area of activity in which they want to preserve their privacy" [7]. Nyms were purchased in a process that prevented linking of nyms to a given client. Freedom circuits connecting clients to Internet destinations were authenticated at the exit node via a pseudonymous signature from the client. Any exit node would be able to link two connections made by a client using the same nym no matter when those connections were made. Thus, though the network and routing structure of Freedom onion routing and NRL-design onion routing are very similar, the goals are somewhat different. NRL-design onion routing does not have pseudonyms, but all traffic that passes over a single circuit can be linked together as coming from the same unknown client. First generation NRL onion routing built a new circuit for each HTTP request, which was in keeping with the way HTTP 1.0 worked. The circuit-building overhead of that approach is, however, quite high. Second and third generation onion routing would thus multiplex several application connections over a single circuit. By default Tor reuses a circuit for ten minutes, although a GUI button allows users to rotate to a new circuit at any time. Still an exit node or collaborating exit nodes in the Freedom Network seeing the same pseudonym at any time, would be able to link these together. Freedom was designed to prevent system components from linking a pseudonym to its client, not against pseudonymous profiling by the exit node or other properties. To the external destination, however, Freedom did not reveal nyms and thus provided client anonymity. But, it did not hide HTTP referer fields and thus permitted that degree of linking even across nyms. If this was a concern, it was recommended that in the course of your browsing session you return in between to a homepage that was a popular web page [25] or simply blank [3].

In the Freedom 2.0 architecture [7] mail reply blocks were replaced with a different design that was intended to improve capacity, performance, and security. It was, in too tight a nutshell, a bulletin board system that allowed people outside the Freedom system to post messages to a nym at a POP server. These could then be retrieved by Freedom clients via anonymous circuits through the network. Another important change to Freedom in the 2.0 architecture was to change to two hops for circuits to the Internet [2]. In Freedom 2.1, this was further reduced to one hop as a default, with a user setable option for two or three hops depending on the desired security/performance tradeoff [3]. This has two security implications. First, anonymity is reduced to a single point of failure, albeit a less predictable one in a distributed network like Freedom's. Also, in the case of Freedom, it is not just the dissociation of the source and destination(s) of that particular circuit that is lost; the association of a nym with a client IP address is also revealed to a compromised onion router. Second, letting users select which connections should have more hops automatically partitions the usage anonymity set, and it indicates to the exit node whether the connection is considered sensitive by the user, hence deserving of closer scrutiny. The designers were aware of various security implications of these choices but also recognized the potential for performance improvements if the number of circuit hops was reduced. These issues were discussed in the above-cited papers.

5. WHAT ONION ROUTING

Onion routing separates identification from routing. But can we describe what protection that provides or tell how strong that protection is? When the first onion routing paper was submitted, it was taken by at least some reviewers as an anonymous communications protocol akin to Chaum mixes and providing the same sort of security. Anonymity (security of anonymous communication) has since at least that time been measured by the size and sometimes distribution on an anonymity set. For protecting a communications initiator, this would thus reflect the uncertainty within a given set of who initiated that communication. The other primary ingredient in defining and measuring security besides the property being protected is the adversary attempting to undermine that security. For much of the history of anonymous communication, this has been the global passive adversary (GPA), one that can observe all messages everywhere, but cannot alter or delete messages or insert its own messages. Onion routing is completely broken against a GPA. It has long been documented that an adversary observing both ends of a circuit can trivially associate those ends by means of timing correlation or volume [46, 41]. In fact, the circuit does not even need to carry any traffic; circuit setup is enough [5]. To date, no padding or similar

scheme has been devised that will counter any sort of realistic adversary (not GPA, but we'll return to that momentarily) while preserving anything like the latency properties expected for most applications on onion routing networks. Work on this remains interesting and worth pursuing [23], but I am dubious that any practical solution exists.

Mixes, on the other hand, break up communication patterns for messages by introducing delays and reordering. They are not trivially broken against a GPA. For this reason, it is common to characterize onion routing as less secure than mixing, if more practical and thus a good engineering tradeoff. But this is too simple. There are realistic adversaries and realistic settings in which an onion routing network is more secure than the comparable mix network that is essentially the same except for swapping mixes for onion routers at the network nodes [49]. Another problem with a GPA that we have described since the early days of onion routing is that a GPA is both too strong and too weak. It is too strong because it is unlikely that an adversary can observe all activity at all places of a large global network. The adversary may be able to observe a large fraction, but not all traffic, everywhere, at all times. This could be viewed as a conservative adversary, as strong or stronger than anything one will face in practice. But it is also too weak because it cannot even delay packets anywhere for a nanosecond or interact with the system by acting as an ordinary user. We can strengthen the adversary by making it less passive, able to use the system as an ordinary user and to own some network nodes at which it can do anything computationally feasible with traffic at that node. This is done in some work on both mix networks [12] and onion routing networks [51].

More importantly, however, this whole approach to security misconstrues both adversary model and definition of security in a way that leads both design and analysis in the wrong direction. The entropist approach [48] assumes that an adversary has a known set of entities (for example, senders if we wish to protect the identity of senders) and that its goal is to reduce its uncertainty about the sender on that set.

A good measure of security is the amount of work an adversary must do to break it. If reducing the size of the anonymity set or improving the chance of guessing the right sender within it is indicative of the work the adversary must do, then this is a good measure. And this is indeed true for systems such as those for voting anonymity, where the goal is to prevent adversaries from associating individuals from a known set of registered voters with the votes cast. But that is not the case for large public onion routing networks. For example, if I am using Tor and both ends of my circuit go through adversary controlled or observed onion routers, it will not matter if there are five other users of the Tor network then or five hundred thousand. And it will not matter how many of those other users have circuits running through the same terminal onion routers or not. An adversary might make use of such numbers. For example, if he could determine that the network only has a small enough number of users that it is practical to just identify and attack them directly to observe their future behavior, then this would be useful. But directly determining a set and reducing uncertainty within it would be a waste of effort. (I am speaking generally. There are always exceptional circumstances.) As noted, a realistic typical attacker will not actually know or care about this when breaking anonymity on an onion routing system. Perhaps we are looking at a set of the wrong things. An adversary wins if he owns the onion routers at each end. So perhaps he should care about the number of onion routers and reducing uncertainty on that set.

First, there are attacks that identify terminal onion routers themselves for some network sizes and configurations [37]. But, outside of P2P designs, that does not connect the source client to the destination. It could help determine where to direct further attacks, but reducing uncertainty on that set is not itself a good measure of anonymity of system users.

Where to direct further attack does matter. This is especially true in the Tor network since 2006, since the deployment of guard nodes. Since that time, instead of building a circuit through three randomly chosen onion routers, the first hop is chosen from a small set (default size three) of guard nodes that are used by a client as long as they are available. Why are there guard nodes? We showed in 2005 that an attacker owning a single onion router could find a hidden server in a matter of minutes by repeatedly making connections to it until the hidden server's rendezvous circuit connected through the attacker's onion router [41]. (Hidden servers are briefly described above in section 4.) Because we wanted to show what could be accomplished owning a single node in the network, the attack only worked on hidden services. We noted that if the adversary owned two or more onion routers, similar attacks could be carried out on ordinary Tor connections to public servers. This was later empirically demonstrated [5]. These attacks were already known, and guard nodes are a version of the earlier introduced concept of "helper nodes" which were meant to help resist such attacks for a variety of anonymous communications protocols, not just onion routing. Though they had been shown possible, our 2005 analysis of hidden services was surprising in showing how cheap and efficient they were: a single adversary node could find a hidden server in just a few minutes. Thus guards were deployed as a countermeasure.

Guard nodes resist attacks that watch for or cause repeated circuit formation until an adversary node is chosen for the first onion router adjacent to the client. Introducing them also meant that an adversary that identifies a guard node, e.g., of someone repeatedly visiting a particular website, knows that if he attacks it he will be able to see one end of a large fraction of the target client's connections. He still needs to be able to watch the other end to take advantage of that. If he owns a destination website and wants to know who visits it, that part will already be accomplished.

This brings us back to defining and measuring security for onion routing. The measure of security is the amount of work an adversary must do to break it. If all nodes in the network are equally vulnerable to compromise by an adversary, then the number of nodes serves as a good indicator of the work an adversary must do to attack onion routing as we have been describing. And this brings us to another mistake of entropism and the designs it engenders. The number of nodes in the network is only a good indicator of the work an adversary must do if they *are* generally comparable in their vulnerability to compromise. But both designs and their analyses treat all parts of the network the same. Tor is a volunteer network. This has much to do with its growth and viability to the largest by far network for privacy and traffic analysis resistance [1, 15] and continues a trend

begun in the first generation design with open source of code and plans for a network controlled by multiple not necessarily mutually-trusting but mutually-collaborating node providers, and then continued in second generation's introduction of allowing node operators to choose their own individual exit policies, etc. But along with this comes an obvious diversity of how likely an adversary is to desire or be able to own or observe network nodes. That sort of diversity is there for any viable large open network; it should just be very obvious with Tor. And if the adversary has a botnet, he could easily, gradually, and stealthily add his own nodes to the network in wide portions of the geographic and IP space until he owns a significant fraction of the network and can observe much of the traffic. This means that the number of nodes tells us little about how resistant onion routing is to an adversary with more than trivial resources and skill level.

To deal with botnets or more generally this diversity of trust in various nodes in the network, we could just stick to those nodes that we trust, for example, that we run ourselves. This returns us to the original reason that onion routing was not deployed as a Navy-only or similarly restricted system. Nodes that are trusted by us may either be known by external indicators and/or identified through behavior. This would undermine the protection that onion routing is meant to provide. So we need a more subtle way to reason about trust. In our current work, we identify trust in a node with the complement of the probability that the node is compromised. We have shown some optimality results for choosing first and last nodes in an onion routing circuit against an adversary trying to own both of them [30]. We also have devised and analyzed algorithms for routing securely even when an adversary might own large portions of the network and when we might be somewhat inaccurate in assigning trust to nodes in the network [31]. We intend to continue to develop this area in various ways, for example, by incorporating link threats into the model [22, 38, 20].

5.1 Traffic Security and Identity

We must overcome one more vestige of the entropist view of anonymity: the notion that the paramount thing to protect is the identification of a specific individual. We have already noted that the standard (entropist) characterization of anonymous communications security starts with a given set of distinct users, messages, source IP addresses, etc. The goal of a security system is then presented as preventing identification of the unique one that matches a given communication elsewhere. This communication elsewhere might be a given message on the other side of a mix or a query or post at a website. More generally we might wish to merely resist reduction of uncertainty about which one is the target individual or about the overall matching of multiple communications with multiple targets. Often, however, this is simply not what matters. If I am a patent examiner wishing to hide that I am searching for prior art on someone's website, I am not trying to prevent him from knowing which patent examiner or source computer is visiting his site. I want to protect that a patent examiner is visiting his site. Similarly if I am using the Internet while traveling and wish to protect my U.S. Government affiliation from local network observers, I am primarily concerned with that. I may not care if he learns specific IP addresses I contact. What I do not want him to do is recognize my affiliation. He could do this if, e.g., a large fraction of my traffic appears most likely headed to .mil or .gov parts of IP space. He may not know any specific IP addresses, and that may not matter to him or be of secondary importance. It is true that an adversary who can disambiguate individual users or source addresses, and who has the appropriate ancillary information, might use this to help him achieve his primary goal. But it is not itself always his primary goal. Sometimes it is, but not always. The entropist view tries to fit all anonymous communication into this procrustean metric.

If protecting identity is sometimes about hiding association with a property that is possibly shared with many others, what about when a property to be protected is itself that you are trying to obtain such protection? Onion routing can hide your ultimate destination from a local observer, but as described so far it does not hide that you are using an onion routing network. The relays in the Tor network are publicly listed in directories. This allows Tor clients to gather the information needed to build circuits. It also means that someone who runs a website and does not want to allow access to it from the Tor network has an easy means to do so. There is even a script made publicly available by the Tor Project to facilitate this. Blocking typically does not have the intended effect; it usually only prevents access by the honest because the dishonest have many less salient ways of access available to them. For this reason blocking is discouraged; however, if someone wants to block access to his server from Tor, he can do so. But this also means that one can block access *to* the Tor network.

Many people use Tor to access search engines, such as Google, so that they can see the results that one would get in a different location. (Search results are in part determined by the IP address of a request. Tor user's are sometimes surprised the first time a search engine homepage loads for them in a different language.) And some search engines, media outlets, or other destinations are blocked or filtered by national or other jurisdictional firewalls. If people use Tor to obtain unfettered access to the Internet, the adversary may block access to the public Tor network itself. Separating identification from routing may then need to be applied to itself. How can we separate identification as onion routing communication from routing to and through an onion routing network?

The Tor network uses *bridges*. These are onion routers that provide access to the rest of the Tor network but are not themselves publicly listed. A major goal is to help unknown individuals behind such firewalls and filters to access the public Tor network in this way—which creates a new problem. How do you tell these people about the bridges without also telling those who want to block them and thus letting the bridges get blocked too? There are various strategies that we only have space to touch on here, such as trickling out the bridge addresses through a limited resource (such as to email requests from accounts with an email provider that does not make it trivial to sign up for numerous accounts), or rapidly rotating bridge addresses through a portion of IP space that an adversary does not want to permanently leave blocked [16, 14]. It also becomes important that traffic to and from the Tor network, even via bridges, is not trivially separable by some aspect of the connection protocol or of the communication patterns. An example of very recent work on blocking resistance via Tor bridges is BridgeSPA [47].

6. CONCLUSION

Why, you old soothsayer-humbug!
no Kaiser are you; you are nought but an onion.
I'm going to peel you now, my good Peer!
You won't escape either by begging or howling.
...
What an enormous number of swathings!
Isn't the kernel soon coming to light?
...
I'm blest if it is! To the innermost centre,
it's nothing but swathings-each smaller and smaller.
Nature is witty!
...
A queer enough business, the whole concern!
Life, as they say, plays with cards up its sleeve;
but when one snatches at them, they've disappeared,
and one grips something else, or else nothing at all.

—Henrik Ibsen, *Peer Gynt*, Act. V, Scene 5

In this paper, we have discussed some of the why, who, how, which, and what of onion routing (also some of the when, although that was scattered throughout). Onion routing has grown as both a research area and an applied solution to many real problems over the last decade and a half. We have barely scratched the outermost layer of many of its aspects in this paper and not touched others at all. If you now have the motivation to explore more deeply, I hope that unlike Peer Gynt in his introspection, you will find something of substance in the process.

7. REFERENCES

[1] Alessandro Acquisti, Roger Dingledine, and Paul Syverson. On the economics of anonymity. In Rebecca N. Wright, editor, *Financial Cryptography, 7th International Conference, FC 2003*, pages 84–102. Springer-Verlag, LNCS 2742, 2003.

[2] Adam Back, Ian Goldberg, and Adam Shostack. Freedom systems 2.0 security issues and analysis. White paper, Zero Knowledge Systems, Inc., October 2001. The attributed date is that printed at the head of the paper. The cited work is, however, superceded by documents that came before Oct. 2001, e.g., [3].

[3] Adam Back, Ian Goldberg, and Adam Shostack. Freedom systems 2.1 security issues and analysis. White paper, Zero Knowledge Systems, Inc., May 2001.

[4] Adam Back, Ulf Möller, and Anton Stiglic. Traffic analysis attacks and trade-offs in anonymity providing systems. In Ira S. Moskowitz, editor, *Information Hiding: 4th International Workshop, IH 2001*, pages 245–257, Pittsburgh, PA, USA, April 2001. Springer-Verlag, LNCS 2137.

[5] Kevin Bauer, Damon McCoy, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Low-resource routing attacks against Tor. In Ting Yu, editor, *WPES'07: Proceedings of the 2007 ACM Workshop on Privacy in the Electronic Society*, pages 11–20. ACM Press, October 2007.

[6] Oliver Berthold, Hannes Federrath, and Stefan Köpsell. Web MIXes: A system for anonymous and unobservable Internet access. In Hannes Federrath, editor, *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, pages 115–129. Springer-Verlag, LNCS 2009, July 2000.

[7] Philippe Boucher, Adam Shostack, and Ian Goldberg. Freedom systems 2.0 architecture. White paper, Zero Knowledge Systems, Inc., December 2000.

[8] Zach Brown. Cebolla: Pragmatic IP anonymity. In *Proceedings of the 2002 Ottawa Linux Symposium*, June 2002.

[9] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 4(2):84–88, February 1981.

[10] Wei Dai. Pipenet, February 1995. Original suggestion posted to the cypherpunks mailing list in Feb. 1995. Later versions are in the Free Haven Anonymity Bibliography.

[11] George Danezis, Claudia Diaz, and Paul Syverson. Anonymous communication. In Burton Rosenberg, editor, *Handbook of Financial Cryptography*. CRC Press, 2010.

[12] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a type III anonymous remailer protocol. In *Proceedings, 2003 IEEE Symposium on Security and Privacy*, pages 2–15, Berkeley, CA, May 2003. IEEE Computer Society.

[13] George Danezis and Paul Syverson. Bridging and fingerprinting: Epistemic attacks on route selection. In Nikita Borisov and Ian Goldberg, editors, *Privacy Enhancing Technologies: Eighth International Symposium, PETS 2008*, pages 151–166. Springer-Verlag, LNCS 5134, July 2008.

[14] Roger Dingledine. Strategies for getting more bridge addresses. `https://blog.torproject.org/blog/strategies-getting-more-bridge-addresses`, May 2011.

[15] Roger Dingledine and Nick Mathewson. Anonymity loves company: Usability and the network effect. In Ross Anderson, editor, *Fifth Workshop on the Economics of Information Security (WEIS 2006)*, June 2006.

[16] Roger Dingledine and Nick Mathewson. Design of a blocking-resistant anonymity system (draft). `https://svn.torproject.org/svn/projects/design-paper/blocking.html`, November 2006.

[17] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, pages 303–319. USENIX Association, August 2004.

[18] Roger Dingledine, Nick Mathewson, and Paul Syverson. Challenges in deploying low-latency anonymity (draft). NRL CHACS Report 5540-625, 2005.

[19] Roger Dingledine, Nick Mathewson, and Paul Syverson. Deploying low-latency anonymity: Design challenges and social factors. *IEEE Security & Privacy*, 5(5):83–87, September/October 2007.

[20] Matthew Edman and Paul Syverson. AS-awareness in Tor path selection. In Somesh Jha, Angelos D. Keromytis, and Hao Chen, editors, *CCS'09:*

Proceedings of the 16th ACM Conference on Computer and Communications Security, pages 380–389. ACM Press, 2009.

[21] Matthew Edman and Bülent Yener. On anonymity in an electronic society: A survey of anonymous communication systems. *ACM Computing Surveys*, 42(1), 2010.

[22] Nick Feamster and Roger Dingledine. Location diversity in anonymity networks. In Sabrina De Capitani di Vimercati and Paul Syverson, editors, *WPES'04: Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*, pages 66–76, Washington, DC, USA, October 2004. ACM Press.

[23] Joan Feigenbaum, Aaron Johnson, and Paul Syverson. Preventing active timing attacks in low-latency anonymous communication [extended abstract]. In Mikhail J. Attallah and Nicholas J. Hopper, editors, *Privacy Enhancing Technologies: 10th International Symposium, PETS 2010*, pages 166–183. Springer-Verlag, LNCS 2605, July 2010.

[24] Michael J. Freedman and Robert Morris. Tarzan: A peer-to-peer anonymizing network layer. In Vijay Atluri, editor, *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002*, pages 193–206. ACM Press, 2002.

[25] Ian Goldberg and Adam Shostack. Freedom 1.0 security issues and analysis. White paper, Zero Knowledge Systems, Inc., November 1999.

[26] Ian Goldberg and Adam Shostack. Freedom network 1.0 architecture and protocols. White paper, Zero Knowledge Systems, Inc., October 2001. The attributed date is that printed at the head of the paper. The cited work is, however, superceded by documents that came before Oct. 2001. The appendix indicates a change history with changes last made November 29, 1999. Also, in [25] the same authors refer to a paper with a similar title as an "April 1999 whitepaper".

[27] David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. Hiding routing information. In Ross Anderson, editor, *Information Hiding: First International Workshop*, pages 137–150. Springer-Verlag, LNCS 1174, May 1996.

[28] Ceki Gülcü and Gene Tsudik. Mixing E-mail with Babel. In *Proceedings of the Symposium on Network and Distributed Security Symposium - NDSS '96*, pages 2–16. IEEE, February 1996.

[29] Sabine Helmers. A brief history of anon.penet.fi - the legendary anonymous remailer. *CMC Magazine*, September 1997.

[30] Aaron Johnson and Paul Syverson. More anonymous onion routing through trust. In *22nd IEEE Computer Security Foundations Symposium, CSF 2009*, pages 3–12, Port Jefferson, New York, USA, July 2009. IEEE Computer Society.

[31] Aaron Johnson, Paul Syverson, Roger Dingledine, and Nick Mathewson. Trust-based anonymous communication: Adversary models and routing algorithms. In George Danezis, Vitaly Shmatikov, and Dongyan Xu, editors, *CCS'11: Proceedings of the 18th ACM Conference on Computer and Communications Security*. ACM Press, 2011.

[32] JonDonym – the internet anonymisation service. `https://www.jondos.de/en/`, 2008. Commercial version of the Java Anon Proxy (JAP). Initially published description in [6].

[33] Aniket Kate and Ian Goldberg. Using Sphinx to improve onion routing circuit construction (extended abstract). In Radu Sion, editor, *Financial Cryptography and Data Security, 14th International Conference, FC 2010, Revised Selected Papers*, pages 359–366. Springer-Verlag, LNCS 6052, 2010.

[34] Aniket Kate, Greg Zaverucha, and Ian Goldberg. Pairing-based onion routing. In Nikita Borisov and Philippe Golle, editors, *Privacy Enhancing Technologies: 7th International Symposium, PET 2007*, pages 95–112. Springer-Verlag, LNCS 4776, 2007.

[35] Jon McLachlan, Andrew Tran, Nicholas Hopper, and Yongdae Kim. Scalable onion routing with torsk. In Somesh Jha, Angelos D. Keromytis, and Hao Chen, editors, *CCS'09: Proceedings of the 16th ACM Conference on Computer and Communications Security*, pages 590–599. ACM Press, 2009.

[36] Prateek Mittal and Nikita Borisov. Shadowwalker: Peer-to-peer anonymous communication using redundant structured topologies. In Somesh Jha, Angelos D. Keromytis, and Hao Chen, editors, *CCS'09: Proceedings of the 16th ACM Conference on Computer and Communications Security*, pages 161–172. ACM Press, 2009.

[37] Steven J. Murdoch and George Danezis. Low-cost traffic analysis of Tor. In *2005 IEEE Symposium on Security and Privacy,(IEEE S&P 2005) Proceedings*, pages 183–195. IEEE CS, May 2005.

[38] Steven J. Murdoch and Piotr Zieliński. Sampled traffic analysis by internet-exchange-level adversaries. In Nikita Borisov and Philippe Golle, editors, *Privacy Enhancing Technologies: 7th International Symposium, PET 2007*, pages 167–183. Springer-Verlag, LNCS 4776, 2007.

[39] Arjun Nambiar and Matthew Wright. Salsa: A structured approach to large-scale anonymity. In Rebecca N. Wright, Sabrina De Capitani di Vimercati, and Vitaly Shmatikov, editors, *CCS'06: Proceedings of the 13th ACM Conference on Computer and Communications Security*, pages 17–26. ACM Press, 2006.

[40] Onion routing: Brief selected history. `http://www.onion-router.net/History.html`, 2006.

[41] Lasse Øverlier and Paul Syverson. Locating hidden servers. In *2006 IEEE Symposium on Security and Privacy (S& P 2006), Proceedings*, pages 100–114. IEEE CS, May 2006.

[42] Lasse Øverlier and Paul Syverson. Improving efficiency and simplicty of Tor circuit establishment and hidden services. In Nikita Borisov and Philippe Golle, editors, *Privacy Enhancing Technologies: 7th International Symposium, PET 2007*, pages 134–152. Springer-Verlag, LNCS 4776, 2007.

[43] Michael G. Reed, Paul F. Syverson, and David M. Goldschlag. Proxies for anonymous routing. In *Twelfth Annual Computer Security Applications Conference*, pages 95–104. IEEE CS Press, 1996.

[44] Michael G. Reed, Paul F. Syverson, and David M. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications*, 16(4):482–494, May 1998.

[45] Marc Rennhard and Bernhard Plattner. Introducing MorphMix: Peer-to-peer based anonymous internet usage with collusion detection. In Sabrina De Capitani di Vimercati and Pierangela Samarati, editors, *Proceedings of the ACM Workshop on Privacy in the Electronic Society, WPES 2002*, pages 91–102. ACM Press, 2002.

[46] Andrei Serjantov and Peter Sewell. Passive attack analysis for connection-based anonymity systems. In Einar Snekkenes and Dieter Gollmann, editors, *Computer Security – ESORICS 2003, 8th European Symposium on Research in Computer Security*, pages 141–159, Gjøvik, Norway, October 2003. Springer-Verlag, LNCS 2808.

[47] Rob Smits, Divam Jain, Sarah Pidcock, Ian Goldberg, and Urs Hengartner. Bridgespa: Improving tor bridges with single packet authorization. In Jaideep Vaidya, editor, *WPES'11: Proceedings of the 2011 ACM Workshop on Privacy in the Electronic Society*. ACM Press, October 2011.

[48] Paul Syverson. Why I'm not an entropist. In *Seventeenth International Workshop on Security Protocols*. Springer-Verlag, LNCS, 2009. Forthcoming.

[49] Paul Syverson. Sleeping dogs lie in a bed of onions but wake when mixed. In *4th Hot Topics in Privacy Enhancing Technologies (HotPETs 2011)*, July 2011.

[50] Paul Syverson, Michael Reed, and David Goldschlag. Onion Routing access configurations. In *Proceedings DARPA Information Survivability Conference & Exposition, DISCEX'00*, volume 1, pages 34–40. IEEE CS Press, 1999.

[51] Paul Syverson, Gene Tsudik, Michael Reed, and Carl Landwehr. Towards an analysis of onion routing security. In Hannes Federrath, editor, *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, pages 96–114. Springer-Verlag, LNCS 2009, July 2000.

[52] Paul F. Syverson, David M. Goldschlag, and Michael G. Reed. Anonymous connections and onion routing. In *Proceedings, 1997 IEEE Symposium on Security and Privacy*, pages 44–54. IEEE CS Press, May 1997.

[53] Tor Metrics Portal. `https://metrics.torproject.org/`, 2011.