Enlisting ISPs to Improve Online Privacy: IP Address Mixing by Default

Barath Raghavan, Tadayoshi Kohno, Alex C. Snoeren, and David Wetherall

University of California, San Diego and University of Washington

Abstract. Today's Internet architecture makes no deliberate attempt to provide identity privacy—IP addresses are, for example, often static and the consistent use of a single IP address can leak private information to a remote party. Existing approaches for rectifying this situation and improving identity privacy fall into one of two broad classes: (1) building a privacy-enhancing overlay layer (like Tor) that can run on top of the existing Internet or (2) research into principled but often fundamentally different new architectures. We suggest a middle-ground: enlisting ISPs to assist in improving the identity privacy of users in a manner compatible with the existing Internet architecture, ISP best practices, and potential legal requirements[1].

1 Introduction

Today's Internet service providers (ISPs) log user behavior for security purposes as a matter of best common practice. Legislators have also ceased to rest on this matter. In February 2009, U.S. House Resolution 1076 was introduced. Though its purported aim is not to monitor users' online behavior, it requires that "A provider of an electronic communication service or remote computing service shall retain for a period of at least two years all records or other information pertaining to the identity of a user of a temporarily assigned network address the service assigns to that user." Combined, these trends—administrative and legislative—indicate that many or most Internet users will soon be indelibly associated with an Internet address. Equally as important are the privacy effects of ordinary Internet use. It is well-known that Internet services provide poor privacy for users. Every time users visit websites or use networked applications, they leave a trail of bread crumbs sprinkled around the Internet. These crumbs can manifest themselves in many ways, such as the IP addresses stored in the logs of a remote web server.

To improve their online privacy, some sophisticated users either choose to avoid certain activities online or choose to use special applications designed to help scramble the remote logs of their activities. As a flagship example of the latter, Tor [7] is a peer-to-peer overlay system that operates on top of the existing Internet and that is very effective at destroying these bread crumbs. Taking an

[1] This work was supported in part by NSF awards CNS-0722000, CNS-0722004, CNS-0722031 and the Alfred P. Sloan Foundation.

I. Goldberg and M. Atallah (Eds.): PETS 2009, LNCS 5672, pp. 143–163, 2009.

egalitarian view of the Internet, however, the principal disadvantage of Tor is that it only benefits those knowledgeable enough to know to download and run it. (There are additional barriers to the use of overlay systems like Tor, including usability and performance. For our purposes, however, these issues are important but of secondary concern.)

We propose a new perspective to improving the privacy of Internet users. Extending an observation from Dingledine and Mathewson [6] about usability, security, and privacy, we argue that users would benefit greatly if their ISPs chose to proactively assist in improving users' privacy. ISPs should be able to do this seamlessly and by default for all their users. Moreover, we wish for a privacy-enhancing approach that ISPs can deploy today, not one that must wait for some future "redesign" of the Internet.

We overcome these challenges in this paper. We show not only that it is possible to enlist ISPs to improve the base privacy of Internet users, but also that it is possible to do so efficiently and cheaply, and in a way that ISPs would actually *want* to deploy. The best analogy to our high-level goals (though not our design) is "caller-ID blocking" in traditional telephone networks. Telephone companies provide caller-ID blocking because of the value-add to consumers. Using a combination of cryptographic and systems-oriented techniques, our solution—the Address Hiding Protocol (AHP)—provides an equivalent "IP address blocking" for the Internet. Informally, the effect of IP Address Hiding is that—from the perspective of a third-party service—every flow that a client node initiates will appear to come from a different (random) IP address within the ISP's address block. One might be tempted to refer to our approach as creating a "super NAT" capable of mixing and scrambling all the IP addresses within an ISP so that they are "anonymized" from the perspective of parties within other ISPs. Such terminology, while somewhat accurate from a functionality perspective, ignores architectural complexities and design constraints that we discuss below.

Returning to our goals in the broader context of encouraging deployment, we observe that ISPs can advertise the value-add of Address Hiding for Internet users just as telephone companies advertise the value add of caller-ID blocking. However, we must overcome other challenges associated with the constraints imposed on ISPs. The first—just as for telephone networks—is that even if an ISP provides Address Hiding to external parties, the ISP must be able to associate a given network flow with a network end-point upon legal intervention (such as when presented with a warrant). As noted above, today many ISPs retain DHCP logs, and it is possible that in the near future all ISPs will be compelled to do so by law. As we shall see, this need, coupled with other architectural complexities like support for multiple ingress and egress points for a single flow and minimal space consumption, imposes challenges on our design space and is what makes our technical solutions more complex than simply deploying a large-scale NAT.

Our approach (AHP), Tor, and applications. We do *not* aim to compete with stronger, pure Internet anonymity overlay systems like Tor, but rather aim to improve the base privacy of all Internet users in a way that is compatible with the existing Internet architecture and the incentive structure for ISPs. We believe

that our system thus provides the best of both worlds—if an ISP deploys our Address Hiding protocol, then the IP addresses of its users would be meaningless to third-party remote services. Thus, such an ISP will have successfully increased the privacy of all of its users from the vantage of external hosts and services. At the same time, we experimentally show that it is straightforward for Internet users to layer Tor on top of our system. We do, however, share one property with Tor and other anonymity systems: the applications (like an email or IM client) running on top of these systems can still compromise a user's privacy (for example, if the application uses cookies or sends users' login names and passwords in the clear). Providing privacy at the lowest network layer is still fundamentally valuable because it can serve as an enabling technology and is immediately useful if a user's application is also privacy preserving, such as if the user configures his or her browser to not store cookies, as offered by Safari and Firefox with "Private Browsing" mode and Explorer with "InPrivate" mode.

2 Address Hiding Goals

Consider a scenario in which a user, Alice, installs one of the latest versions of a popular browser such as Safari, Explorer, or Firefox. She reads the "new features" list and has learned of the "private browsing modes" for these browsers—modes that will (among other things) not allow cookies to be stored or will always scrub cookies upon exit. While such application-level control will improve Alice's privacy, it is fundamentally limited since the websites Alice visits will still be able to record, recognize, and profile Alice's originating IP address. Anonymity solutions, like Tor [7], can help improve Alice's anonymity but will require Alice to install a separate application package, are less usable than simply clicking a control within the browser like "Private Browsing" or "Reset Safari," may be too heavyweight for all applications, and may bring with them their own risks of surveillance by P2P exit nodes [18].

In contrast, AHP enlists ISPs to assist in improving the privacy of users like Alice by scrubbing their outgoing IP addresses. In order for AHP to have any hope of being deployed in practice, AHP must respect the forensic requirements and existing practices of ISPs—including the need to maintain identity information in compliance with legislative requirements or corporate policies. Thus, AHP strikes a balance: increased privacy in the common case when the average Internet user is interacting with webservers, but not so much privacy as to force ISPs into an awkward state of non-compliance. As we show later, users can still easily layer Tor (and other applications) on top of AHP. We elaborate on these specific goals, requirements, and assumptions below.

System requirements and goals. Informally, we have five requirements and goals: (1) hide the network-layer identity (IP address) of the two parties involved in a network flow from an outsider; (2) prevent the correlation of any two network flows between the same two parties by an outsider through network or transport-layer information; (3) for legal compliance and compatibility with existing practices, enable high-speed, long-term forensic attributability of packets

without onerous storage or bandwidth requirements on the part of the ISP; (4) ensure that AHP is compatible with popular network applications and that it composes well with existing anonymity systems (such as Tor); (5) require no modifications to the client applications participating in traditional client-server communications.

Trust. We begin with the assumption that we trust the ISP, as users already do today; i.e., we do not introduce new trust assumptions so users are no worse off than they are today[2]. AHP is a protocol implemented in the network by a trusted provider. The network provider can log all network traffic, and moreover, all address mappings, thereby enabling it to revoke client-side address privacy for network administration. Indeed, once we place trust in the ISP to perform address hiding, there is little incentive for it to *not* hide its customers' addresses to outsiders. Any system that does not concede this ability to network providers is unlikely to be deployed.

Types of attackers. Beyond the trust relationship required with the service provider, our threat model is straightforward. We divide the path a flow traverses into three components, with the end two pieces of the path within the client's and the server's network provider domains respectively. We consider two types of attacker: the insider and the outsider. The insider is an attacker within a trusted network provider's domain that is capable of sniffing and/or injecting packets; for example, an insider (from the perspective of a client) might be a server with which it is communicating or a neighboring host that can sniff packets, provided that the server or host are within the same ISP's network. An outsider is a transit provider between the client and server networks.

Space- and time-efficient forensic support. To comply with deployment constraints, we wish to enable an ISP to recover the true source of any packet that was hidden by one of its AHP gateways, thereby ensuring that all packets are attributable to their sender. A naive solution for attributable address hiding would require the storage of an ever-growing table of source-to-public-flow-identifier mappings on the order of several gigabytes per day per router for a large ISP. Our aim is to support attribution regardless of how far in the past the packet was sent with minimal state stored at the ISP.

Compatibility and composability. ISPs are unlikely to deploy any system that breaks popular network applications in the process; backward-compatibility is crucial. In Appendix A we present a case study of several common user applications—including Firefox, Tor, and BitTorrent—while using our prototype of AHP. AHP provides network-layer IP address privacy. However, some users will wish to use anonymity systems, which are more heavyweight but also have broader aims and stronger guarantees. We believe it is essential that AHP not decrease the options a user has to protect her privacy, and thus, we design AHP to be composable with existing systems such as Tor.

[2] Those users who do not trust their ISP can and do use anonymity systems such as Tor; we aim for defense in depth, so such users can continue using Tor.

Non-goals. While AHP is designed to improve users' privacy, we do not aim to provide "anonymity" in the usual sense. More generally, we enumerate several non-goals that inform our design—that is, goals that we do not seek to achieve: (1) to prevent insider attacks, regardless of outsider cooperation; (2) to prevent attacks that involve application-layer payloads; (3) to prevent timing or other side-channel attacks; (4) to provide data privacy or authenticity; (5) to provide privacy for dedicated server hosts; (6) to support per-flow privacy for non-TCP transport protocols.

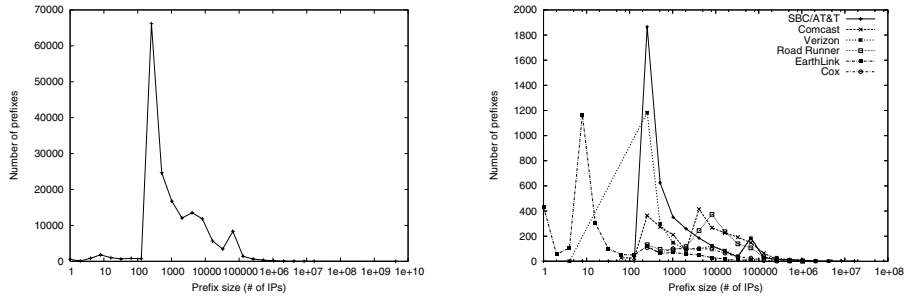
Summary. This specific collection of non-goals, as well as the earlier goals, were chosen to be supportive of the example applications such as the one we mentioned earlier, as well as the needs of ISPs. As Dingledine and Mathewson [6] noted, users would benefit greatly if their ISPs chose to proactively assist in improving users' privacy, and our goal is to instantiate their vision. Power users can, however, continue to layer stronger mechanisms like Tor on top of AHP.

3 Measurement Study: Your ISP Is Crowded

Privacy researchers have long held that identity privacy can only be provided by hiding within a "crowd" of other users [26]. When an adversary cannot distinguish between the members of the crowd, each member of the crowd's privacy is preserved. The larger the crowd, the better the privacy. In the past, researchers have designed systems to artificially induce a crowd of privacy-seeking users, typically through an overlay network. Then, by measuring the size and properties of the induced crowd, we can ask "how much privacy does the induced crowd provide?" This approach has yielded many fruitful results in the anonymity research literature.

In this paper, we learn from the crowd-based approach and apply it to the new research area at hand. Specifically, since we aim to raise the privacy bar across the board for users of an ISP, we ask: "how much privacy can we provide by default?" The answer to this question comes in two parts. First, we must determine whether an appropriate crowd exists in the Internet today. Second, we must design a system to leverage this crowd appropriately. In this section we address the first part, and show that **ISPs are already crowds** of sufficient size to provide privacy given an appropriate system design. Our key observation is that each IP address prefix provides a "crowd" of addresses within which we can provide identity privacy[3]. Thus, the requirement is simply that the system multiplex the hosts within that address space across the available addresses in a manner that is opaque to an outside party.

[3] The Internet's routing system today is structured hierarchically, with so-called "Tier-1" ISPs at the top of the hierarchy—such ISPs have complete routing information for all valid destinations in the Internet. Other ISPs and networks attach to these Tier-1 ISPs to perform routing. Each ISP or network is assigned one or more IP address blocks or "prefixes" within which it can assign public addresses for its hosts. These prefixes are publicly announced to other networks via the Border Gateway Protocol (BGP).



(a) Prefix size distribution for ASes in the RouteViews BGP feed

(b) Prefix size distribution for 6 largest ISPs

Fig. 1. Address block sizes in the Internet

Since we wish to understand how much potential for privacy already exists in today's Internet, we first need to look at where the potential crowd comes from. The Internet consists of numerous Autonomous Systems (ASes)—each of which is typically an ISP or large organization—that route traffic to each other. An AS contains thousands or millions of hosts, each of which is typically assigned an IP address. Although routing protocols operate on the level of ASes, packet forwarding operates on the level of IP addresses—each packet must name both a source and a destination IP. As such, each packet identifies a host, which is a crowd of size one. What if we view each AS or ISP as a crowd within which outside parties cannot peek? Each ISP controls some portion of the Internet's address space; an ISP can provide the required opaqueness by obfuscating packets' source addresses as they traverse the network boundary to the outside Internet.

Thus, our challenge is to understand the size of crowds that are possible when hiding hosts within existing ISP address spaces. To this end, we examine the BGP routing advertisements as seen by RouteViews on Sept. 7, 2007 [19][4]. As a baseline, Figure 1(a) shows the size distribution of all advertised IP prefixes; we can see quite clearly that many prefixes are small—on the order of a few thousand addresses at most. The most prevalent prefix size advertised is /24, few prefixes that are advertised are smaller than that. Thus it appears that prefixes as advertised today provide insufficiently large crowd size.

However, the deployment of AHP is of most value in larger ISPs, within which there is both more room to hide and perhaps more commercial incentive for deployment. To explore such a scenario, we examine in Figure 1(b) the sizes of address space advertisements for the six largest consumer ISPs—SBC/AT&T, Comcast, Verizon, Road Runner, EarthLink, and Cox[5]. The results show that many small prefixes are being advertised even in these large ISPs. However,

[4] The specific date has no special significance and is simply a snapshot of route advertisements taken at the time of our analysis.

[5] We isolate the advertisements for these ISPs by searching the text identifiers of the IANA AS number allocations for these ISPs' common names.

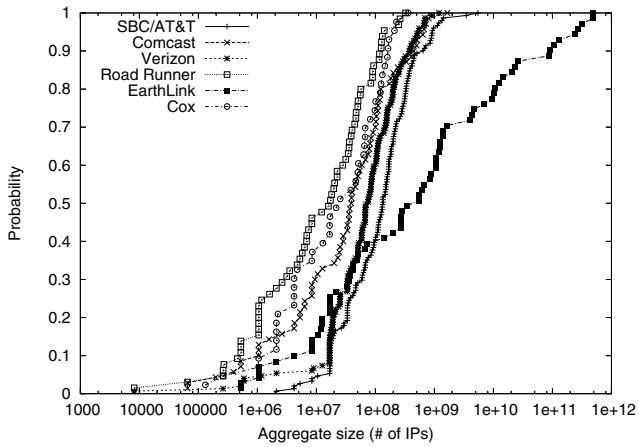


Fig. 2. Size of crowds if ISPs were to aggregate based upon physical location; CDFs of the size of the aggregated crowds (log scale) based upon BGP advertisements from RouteViews

upon aggregation, many of these small prefixes are subsumed. ISPs have an interest in advertising larger address blocks, if for no other reason than to reduce management overhead and reduce routing table sizes. With small prefixes comes greater routing flexibility. These two factors are in tension. However, prefixes can be reasonably aggregated together if they originate from the same geographic region; this is aided by the network structure of large ISPs, which have points of presence (PoPs) in most major cities.

To discover how much potential there is for such geographic aggregation, we used the Oasis [9] and NetGeo [20] geolocation services to map all the IP prefixes of the six ISPs above and aggregated them based upon location. Many IP prefixes map to the same location, likely indicating that they originate from the same PoP. This approach is not perfect, as the services contain necessarily incomplete and inaccurate data; about 15% of the prefixes were unmappable, and we omit them since we are interested in the potential for large aggregates, not small aggregates. Thus, our results represent a lower bound on the aggregation possible within the studied ISPs. Figure 2 shows CDFs of the aggregated address spaces with prefixes aggregated if they mapped to the same physical location irrespective of numerical proximity. We immediately see that address spaces that are geographically close have great potential for aggregation on those grounds. While in the scope of the Internet's address space a million addresses is relatively small, such a space is likely ample for hiding[6]. For example, 50% of Road Runner crowds (that is, 50% of prefixes) would contain over ten million IPs if aggregated by location; 50% of Earthlink crowds would contain over 100 million IPs.

[6] We leave open the question of when and where from an ISP traffic engineering perspective it is appropriate to actually perform such aggregation among geographically-proximate IP prefixes.

Since address hiding is a fundamentally different service than mix-net style anonymity systems, direct comparisons of the sizes of IP prefixes to that of anonymity sets is not possible. However, it is possible to look at the raw numbers for other systems, to check that the values are in the same range. Tor is estimated to have on the order of 200,000 active users. Architecturally, each of these users appears the same from the perspective of a destination host. To consider a parallel concept—what level of identifiability in the real world is acceptable—we can look at the Census. The U.S. Census Bureau has long had policies to enable meaningful extraction of demographic data from the decennial census while still maintain a level of privacy for people in the queried data sets. As of last year, different microdata queries with the Census Bureau were limited to return data for population groups of at least 10,000 and, for another dataset, 100,000 individuals. Thus we are comforted that ISPs can easily advertise 1,000,000 address IP prefixes within which users can hide.

4 A Cryptographic Approach to ISP Crowds

AHP's design is realized in two parts, one at the ISP gateway, and an *optional* component on the client. An ISP can unilaterally deploy AHP-capable gateways, thereby enabling its clients to immediately benefit from deployment. Importantly, with AHP, ISPs can provide the benefit of client-side privacy to their users even in the absence of any explicit client support for it. The client-side component of AHP is required only to support peer-to-peer and server applications; it does not affect application-level protocols, and thus supports both legacy clients and servers.

We wish to protect users from having their applications inadvertently reveal their identity. Thus, AHP must be transparent to ordinary client-server applications and must maintain privacy. Some user applications, such as peer-to-peer programs, require the ability to support both outgoing and incoming connections. Because incoming connections generally require an externally routable IP address, AHP allows applications to request a temporary, but fixed inbound identifier at which external hosts can contact them. We denote one-time only addresses as hidden addresses and denote sticky addresses to be those that can be reached by many parties from the outside. Internal to the ISP, we assign each host two addresses, a default hidden one with which to communicate with full privacy, and a sticky one that provides a stable external identifier that can be contacted by multiple hosts via multiple flows. In our design, applications must explicitly request use of the sticky address.

4.1 Design Overview

While there are numerous challenges that we faced in the design of AHP—such as the need to handle multiple ingress and egress points and the need to minimize the amount of data stored for forensic purposes—the high-level design of the AHP gateway is both simple and efficient. Each outgoing packet's IP address

Table 1. A summary of AHP design components and mechanisms

Goal	Mechanism	Description	§
Secure default	hidden and sticky	Each host is assigned two addresses; the default is strictly-hidden (hidden), the other partially-exposed (sticky)	4.2
Address hiding	Tweakable block cipher	Efficient permutation of local IP/port into public IP/port given private key and public destination IP/port	4.3
Long-lived flow support	Collision detection	Prevents two flows of different epochs from mapping to the same public IP/port	4.3
Birthday attack prevention	Key rotation	AHP gateway changes its block cipher key per epoch to prevent repeat flow transformations	4.3
Forensic support	Time-based keys	Epoch keys selected during key rotation are derived from a master key based on the time, and can be regenerated later	4.3
Inbound flow support	sticky mode	Enables hosts behind an AHP gateway to request semi-permanent public addresses to accept inbound connections	4.4
Backward compatibility	expose wrapper	Wraps unmodified applications to enable their use of sticky mode to allow inbound connections from remote hosts	4.4

(host portion only)/port is encrypted. Encryption on short values, such as 16 bits of an address, is non-trivial; these and other challenges lead us to selecting a short-domain tweakable block cipher. The key used in this process is rotated over time, mitigating birthday attacks and ensuring that the permutation can be reproduced at a later time—this is crucial not only for forensic support, but to support multiple ingress and egress routers within an ISP.

Most large ISPs have many routers through which packets can enter and leave the network. Due to asymmetry of routes, a flow's outbound packets may traverse a different router than its inbound packets. A non-keyed, NAT-like solution would require constant, real-time replication of flow-table state between all participating routers in the ISP—clearly an onerous process. While we omit a full concept of multi-ingress/egress support using AHP, our approach is straightforward—all participating routers simply use the same keyed permutation and exchange small amounts of information every few hours to keep their state in sync.

When presented with packets from a hidden address, the gateway performs a full hiding operation, which includes transforming the IP/port into a different public IP/port for every distinct destination. However, when presented with packets from a sticky address, the gateway performs the transformation solely based upon the internal IP/port pair and not the destination IP/port pair, so as to maintain a consistent public IP/port to which remote hosts can connect and communicate[7]. Next we delve deeper into the details of our implementation, both abstractly and as it pertains to our software prototype. Table 1 provides a summary of several AHP design components and mechanisms described below, and hints at some of the challenges that our design overcomes.

[7] We must note, however, that providing sticky addresses is not without consequences. Any system that provides pseudo-permanent identifiers like our sticky addresses may inadvertently reveal at the client-side the destination of packets through correlated inbound flows. Most modern NATs enable hosts to register ports to be forwarded, but a NAT's goal is not to ensure identity privacy. In our context, to fully understand the impact sticky addresses and peer to peer applications when used with AHP, we hope to study a real deployment of AHP within a small ISP.

4.2 Address Partitioning

To preserve privilege separation, all hosts within the local routing domain receive two addresses for routing within the ISP: a hidden address and a sticky address. A hidden address can be converted into a sticky address and vice versa by flipping the high order bit of a /8 address. hidden addresses reside in 10.128.0.0/9 and sticky addresses reside in 10.0.0.0/9[8]. The hidden address is assigned to the default network interface on the user's host, thereby ensuring that network communication is private by default. The ISP will scramble hidden and sticky addresses when communicating with hosts outside of the ISP.

4.3 Handling Traditional Client Applications: hidden Mode

In hidden mode, when using client-server applications, there are no perceptible changes required by the client. Since each host's hidden address is its default, all programs that do not explicitly specify an interface will bind to it and use it for outgoing TCP connection requests. sticky addresses are not even needed for such applications, so no changes need to be made on the host. Thus AHP requires no modifications to web applications like Firefox and Safari or even Tor clients.

Address hiding. Similar to a NAT, we aim to translate addresses. However, our fundamental design constraint is that we provide forensic support. Given the large storage requirements to store a NAT's flow tables over a long history, the challenge we face is to translate hidden addresses deterministically by permuting the hidden address and the TCP source port. To improve privacy and ensure that no two flows from the same source can be correlated, we must base the permutation not just on a private key, but on the destination IP and port.

We find that the primitive that meets this challenge is the tweakable block cipher [17]. Given a public "tweak" t and a private key k (known only to the gateway router), we can instantiate a tweakable block cipher $E_{k,t}(M) = F_k(M \oplus H(t)) \oplus H(t)$, where F is an underlying block cipher and H is a cryptographic hash function. In our prototype, we instantiate the block cipher F using 20-round RC5 with a 16-bit word length (producing a 32-bit block length) and instantiate H with SHA-1[9]. In our discussion, we restrict the space of hidden addresses to 10.128.0.0/16; we also discuss address spaces and hiding sets in Appendix B. Used in this manner, $E(\cdot)$ yields a secure PRP keyed on the private key k that is held by the AHP gateway and by the public data t that—in the manner of tweakable block ciphers—selects the particular PRP family that the key operates with. The operation amounts to a single-block encryption, and

[8] In our implementation, we assume that all local addresses are assigned from within the 10.0.0.0/8 address block, which is officially reserved as a block of private-network IP addresses. Note that this assumption holds even when the externally advertised prefix is small, e.g., a /24 or smaller.

[9] Cryptanalysis has proved effective against lower-round variants of RC5; in addition, any block cipher with only a 32-bit block width is potentially vulnerable to birthday attacks. We mitigate these potential issues via key rotation, as we discuss later.

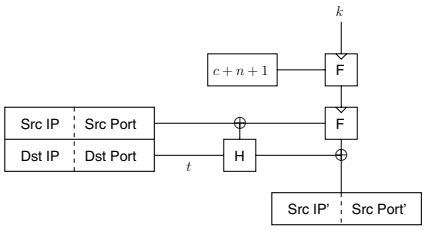


Fig. 3. The concrete instantiation of AHP. The final result, (Src IP', Src Port') replaces the original source IP and Port pair.

Table 2. Summary of findings from trace-based simulation of flow collisions

Quantity	Value
Duration	30000s
Total number of flows	8960585
Largest epoch current flowset	227489
Largest epoch old flowset	3062
Total number of collisions	120511
Total number of old-flow collisions	582

requires no authentication (say, with an appended MAC) because tampering of the packet header will necessarily cause misrouting of the packet by the routing system. We apply the tweakable block cipher $E(\cdot)$ as shown in Figure 3: we refer to this as the Hide operation. We store the host portion of the hidden address in the high-order two bytes of a four-byte block b, and store the TCP source port in the low-order two bytes. We compute the tweak t using the same approach, except that we use the entire destination address and port, yielding a six-byte tweak. We apply $E_{k,t}(b)$ and replace host portion of the hidden address with the two high-order bytes of the result and similarly replace the port with the two low-order bytes. Finally, we replace the network portion of the hidden address with that of the public IP prefix, and forward the packet.

While our description suffices to explain how ordinary data packets are sent, there are several issues that arise with this basic design. First, to ensure long-term security from birthday attacks, the gateway must rotate keys. Second, since each permutation (created by $E(\cdot)$) is independent, collisions in the public IP/port space will occur between flows hidden with different keys. As we describe next, these two issues must be resolved simultaneously. A separate issue is that we must be able to handle multiple ingress and egress points; handling such ingress and egress points is technically straightforward given our solutions to the above two challenges, which is not surpising since we iterated on this goal in combination with the first two issues mentioned above. We do not detail our somewhat involved design for handling multiple ingress/egress points, however it may prove to be a useful extension for a real deployment in a large ISP.

Overcoming challenges of long-lived flows. Key rotation presents a fundamental challenge: since no bits introduced by the AHP gateway can persist solely in packet state over the lifetime of a flow, there is no way to tag packets within a flow with their time of birth (which would indicate which key to use for the flow). Instead, we must associate a flow with a key without packet state or per-flow state at the gateway. We achieve this by maintaining sets that associate packets with the keys that are used for their translation—each key

corresponds to its *epoch*. We maintain a set of sets, $(S_1, \ldots, S_n)$ each associated with a counter designating its epoch number, $(c, \ldots, c+n-1)$ for some value c[10]. The last element of each of these sets is $(S_n, c+n-1)$. When a TCP SYN packet arrives, our goal is to ensure that the resulting public IP/port pair are not already in use by another flow. To this end, we perform the Hide operation on the outgoing packet and search for the (Src IP', Src Port', Dst IP, Dst Port) in the sets beginning with S_1 proceeding to S_n. We use the corresponding key to translate the flow and stop the search at the first matching set. If there is no match, we insert it into the "current" set. If there is a match, we send a TCP RST to the client, forcing it to attempt to reconnect. While this solution is somewhat complex, most network applications are designed to be robust to temporary outages; we evaluate the frequency of matches in the next section. When a TCP RST or TCP FIN packet arrives, we perform the same search for the matching set, but perform a set remove operation of the flow. Maintaining sets of each key's associated flows requires flow state within each epoch; in Section 5 we study the memory requirements of these flow sets.

Avoiding birthday attacks. To ensure that two flows with the same flow ID do not translate the same twice, we must rotate keys at the AHP gateway over a fixed time period—an *epoch*. The lifetime of each key is determined based on the privacy guarantees we wish to provide. For each existing set S_i, we create a parallel set S'_i and, for some constant time window (say, five minutes), we insert any active flows that match S_i into S'_i. After the time window expires, we shift all counters and sets down by one, and replace them each with their parallel sets that only contain currently active flows. We add all flows present in the oldest set S_1 into a per-flow table of very long-lived flows, indexed by their destination IP/port. Finally, we clear the newest set and increment its counter, thereby changing the current key.

Epoch key selection. By deriving epoch keys from a master key and the current epoch counter, we avoid having to store all epoch keys. Thus, if needed for forensics, we can easily regenerate the key for any particular epoch. The current key, k_{c+n-1}, is derived from the master key k using AES as a PRF, $k_{c+n-1} \leftarrow F_k(c+n-1)$. ($c$ increments every epoch; n is the number of sets used for collision detection.)

Multiple ingress and egress points. AHP's design extension to support multiple ingress/egress points ensures that large ISPs that have asymmetric routes can still use AHP. We can ensure that each router increments its epoch counter at the same time via ordinary `ntp` time synchronization. At the end of each epoch, all routers responsible for a given IP prefix merge their parallel sets S'_i. This ensures that the flow state at these routers is synchronized, thereby avoiding collisions upon flow arrival.

[10] Here we present the conceptual model; in practice, set of sets is circular, and only requires the low value, c, rather than the entire set of counters.

4.4 Handling Peer-to-Peer Applications: sticky Mode

Moving beyond client-only applications poses a new challenge—enabling applications to reserve public IP/port pairs to receive incoming connections as is necessary to support P2P applications like BitTorrent. Since the available address and port space must be shared with that of hidden mode connections, we must change our mechanism to ensure that the two modes operate harmoniously.

As we have discussed, when in sticky mode, two fundamental changes in behavior occur. First, since sticky mode entails a publicly addressable IP/port pair, the external address for a host does not change based upon the remote IP or port. Second, for the gateway to know to reserve an IP/port pair, the user application must make an explicit request; we provide a wrapper that makes these requests on behalf of unmodified client applications. Since an explicit mapping must be made, the gateway behaves like a NAT. However, to avoid having to store the mappings for forensics, we select the mappings in a manner similar to in hidden mode.

sticky wrapper library. Unlike ordinary client applications, peer-to-peer applications require inbound connection support. Thus, we provide a wrapper script, `expose`, that uses library interposition in Unix (via `LD_PRELOAD`) to intercept specific system calls that require special handling in sticky mode[11]. Changing the gateway to support reservation of sticky addresses is straightforward. The primary cause for concern is that sticky addresses will collide with hidden addresses. To avoid this, we add a smaller, parallel group of flow sets to pre-test incoming packets: any that match the sticky sets are translated in the same manner as hidden addresses, except that the gateway uses a constant (0) as the tweak value, and in doing so ensures that the sticky address maps to the same sticky address for all incoming flows, regardless of origin IP or port.

[11] Specifically, `bind()` and `getsockname()` both require modification. With `bind()`, our main task is to explicitly select the sticky interface on the host in the `sin_addr` field. However, before returning to the application from the library, but after the local `bind()` call, we make a request via a single UDP packet to the gateway to reserve a sticky IP/port pair. In the UDP request is the local IP and port assigned by the real `bind()` call, as retrieved via the real `getsockname()` call. If the library does not receive an affirmative response containing the socket's sticky address and port within a timeout (currently 500ms), the library returns `-1` and sets `errno` to `EADDRINUSE`. If the library receives an affirmative response from the gateway, it returns `0`. To ensure that peer to peer applications that wish to announce their presence can do so, we intercept `getsockname()`. If the gateway allowed the allocation of a sticky address, then we possess the socket's externally-visible IP address/port combination, and return it to the application. When the application calls `listen()`, we acknowledge the setup of the sticky address in a second UDP packet to the gateway; the packet serves a similar purpose to a DHCP lease, and must be renewed periodically (we do not implement periodic renewal; how often to renew is a matter of policy) to maintain the public address. In addition, we add the socket to a table of active server sockets. Finally, when the application calls `accept()`, we return the result of the real `accept()` call, but also add the newly returned socket to the list of sockets with the given sticky address.

5 Analysis

We have already analyzed several aspects of AHP's security in-line in Section 4. In addition we study specific aspects of AHP's design—including collision likelihood and forwarding performance—that warrant further exploration. We consider the length of the key rotation interval in Appendix C.

5.1 Forwarding Performance

Any in-network system such as AHP must be capable of high line rates. While we expect that a real deployment would involve a hardware implementation of the AHP gateway, we measured our software prototype as an indication of the efficiency of the design itself. Our prototype of AHP runs as a daemon process under Linux 2.6 and handles all AHP gateway functionality. The prototype captures packets using superuser-specified IPTables rules that divert packets to the daemon via userspace queueing. Packets are then translated appropriately and transmitted via a raw socket. The prototype supports configuration of the size of the IP prefix in question, of its memory usage, and of its key rotation interval.

We forwarded data via an ordinary TCP socket from a host to itself over the system's loopback network interface, to test the performance of the core of the algorithm. We perform the test on a 1-Ghz Pentium M laptop running Linux 2.6.22. We find, not surprisingly, that other packet handling costs are greater than the cost of AHP's processing, and thus, as the packet size increases, so does the forwarding rate. At its peak, the system forwards at 408 Mbps. We find that although our AHP gateway is somewhat slower than native Linux forwarding, which peaks at 506 Mbps on the same hardware, the precise forwarding rates themselves are not of primary importance, since the overhead of AHP processing is a fixed per-packet cost that is small relative to other overheads.

5.2 Collisions

In the operation of an AHP gateway, "collisions" can occur wherein two distinct local IP/port tuples going to different IP/port tuples are mapped to the same external IP/port tuple. Since tweakable block ciphers represent a family of permutations parametrized by both the tweak and the key, key rotation yields a different, specific permutation that may collide with mappings under past or future keys. Here, the collision probability is governed not by the birthday paradox, as arriving flows with the same key cannot collide with one another. We compute the likelihood of collision as follows: the collision of a newly arriving flow is related only to the number of flows that exist in the flow sets for old keys. Thus, by selecting an appropriate key rotation interval—one that balances privacy and collisions—the problem can be mitigated. A recent study by Lee and Brownlee [14] indicates that only about 10% of flows last over ten seconds, and the fraction that lasts for 1,000 seconds is vanishingly small—less than 0.1%. Assuming a key rotation period of 1,000 seconds, which is well under that which is needed for maintaining key rotation privacy (indeed, with a 1,000-second

key-rotation period, the probability of a birthday collision of the local port on a host is less than 0.003), less than 0.1% of flows will remain in flow sets for old keys. Assuming these flows all remain for a long time (conservatively, one day), and the gateway services ten million flows per day, that yields, conservatively, 10,000 old flows. Given a /16 IP prefix, the space of possible address/port pairs is roughly 2^{32}, and thus the probability that any single flow arrival will collide with an old flow is 2.3×10^{-6}, which we believe to be small enough for practical purposes—about one out of every half-million flows will collide.

To rest our analysis upon firmer ground, we perform a trace-based simulation of collision rates. Our goal is to better understand the collision probability that governs the rate of TCP RSTs being sent to hosts. We use a real day-long cross-Pacific trace from March 3, 2006 of a 100-Mbps backbone link provided by the Japanese WIDE project [33]. A summary of our results is shown in Table 2. While the sources and destinations of the packets in the trace are not all from one particular ISP or IP prefix, we use the trace to get a better understanding of not only how many flows occur over the one-day period, but also how many persist long enough that they would have caused collisions if they had been hidden by an AHP gateway. We begin our trace-based simulation by preprocessing the input trace: since we have 32-bits of IP address and 16-bits of port for each packet source by hashing them together to produce a single 32-bit identifier for them. (This preprocessing step may cause extra collisions that would not occur in practice, which will mean our results are on the safe-side.) After this step, we apply our ordinary AHP address hiding to the resulting packet headers and store them appropriately in their flow sets. We rotate keys every 300 seconds and use a 30-second grace period before rotation; we did not carefully choose these durations, but found the results were not sensitive to them. We use only two sets—a current flow set and a recent flow set. We keep an exact old-flow set for those flows that last more than two rotations (600 seconds).

What we find, as shown in Table 2, is that since the vast majority of flows are short-lived, the tables for older flows do not gather many flows, and remain small over time. As a result, the collision rate (and thus, the RST rate) is low. However, we were initially puzzled that the collision rate was even as high as it was. Upon careful examination of the packet traces, we found that the vast majority of the collisions in our count appear to be due to TCP SYN flood attacks—if such a packet is unlucky enough to collide once, then each duplicate SYN in the flood similarly collides[12]. Legitimate senders, on the other hand, do not send many duplicate SYNs for each flow.

6 Related Work

AHP belongs to a different class of privacy-preserving network protocols, but resembles prior protocols designed to provide anonymity. We note some of the

[12] We did not exactly quantify the impact of SYN floods on the collision rate, since there were several cases in which it was ambiguous whether a series of packets constituted an attack.

key works here, but defer to Danezis and Diaz's detailed survey of the extensive research in anonymity [4]. Prior work can be broadly classified as belonging either to *anonymity* research or *new Internet architectures*. Since Chaum's seminal work, many researchers have developed re-routing based anonymity systems, including Crowds [26], Freedom [34], Tor [7], Tarzan [10], GAP/GNUnet [2], Herbivore [30], P5 [28], Hordes [16], Slicing [13], and JAP [12]. Over time, researchers have developed attacks of ever-increasing sophistication, involving techniques such as timing analysis [15,27,29] and broad spectrum traffic analysis [22,25], and have found weaknesses in systems designed to enable forensic support [5]. From a design perspective, AHP bears closest resemblance to CPP, a system that hierarchically encrypts IPv6 addresses to obtain privacy [32], and Anonymizer [1] and Proxify [24], which provide commercial application-level anonymization proxying. However, the design goals of AHP are different; critically, AHP is easily composable with other anonymity systems (including Anonymizer.com and Proxify) and operates at the network layer. As a result of our different goals, the architecture AHP differs significantly from that of either of these commercial services. While not designed explicitly with anonymity or address hiding in mind, a separate thread of research in the networking community lends itself to address hiding. In particular, four projects—IPnl [8], ROFL [3], HIP [21], and *i*3 [31]—describe fundamentally new Internet architectures that could accommodate additions to provide AHP-like functionality.

7 Conclusions

Today's Internet does not adequately protect the privacy of users. Indeed, even with strong end-to-end cryptographic mechanisms like SSL and the emergence of privacy controls within applications, such as the private browsing modes of Safari and Firefox, the Internet architecture—almost by definition—violates the privacy of Internet users by assigning unique identifiers (IP addresses) to users' machines. Dingledine and Mathewson's [6] observation—that a little bit of anonymity for everyone by default is valuable—captures the essence of the goal in this research, while also striving for a system that is is both incrementally deployable *and* appropriately incentivized for those responsible for deployment: ISPs. If our approach were adopted, the privacy of network communications would not be decreased from that to today's, even *if* ISPs chose to invoke AHP's forensic capabilities—and indeed users could layer Tor or other stronger mechanisms on top of AHP. In addition, we argue that the average Internet user's privacy would be improved by AHP. Our results show that today's ISPs can be treated as a crowd to provide users identity privacy. Our prototype shows that AHP induces negligible overhead. In addition to benchmarks, we ran ordinary Internet applications as case studies—Firefox, BitTorrent, and Tor—on top of AHP and anecdotally observed no negative impact. We believe that AHP could be integrated into existing ISP infrastructures, and is appropriately incentivized with forensic capabilities for regulatory and policy compliance, high performance, and the ability to provide address hiding as a service to one's customers.

References

1. Anonymizer, http://www.anonymizer.com/
2. Bennett, K., Grothoff, C.: Gap – practical anonymous networking. In: Proceedings of Workshop on Privacy Enhancing Technologies (2003)
3. Caesar, M., Condie, T., Kannan, J., Lakshminarayanan, K., Stoica, I.: ROFL: routing on flat labels. In: Proceedings of ACM SIGCOMM (2006)
4. Danezis, G., Diaz, C.: A survey of anonymous communication channels. Technical Report MSR-TR-2008-35, Microsoft Research (January 2008)
5. Danezis, G., Sassaman, L.: How to bypass two anonymity revocation schemes. In: Proceedings of the Privacy Enhancing Technologies Symposium (2008)
6. Dingledine, R., Mathewson, N.: Anonymity loves company: Usability and the network effect. In: Proceedings of WEIS (2006)
7. Dingledine, R., Mathewson, N., Syverson, P.: Tor: The second-generation onion router. In: Proceedings of the USENIX Security Symposium (2004)
8. Francis, P., Gummadi, R.: IPNL: A nat-extended internet architecture. In: Proceedings of ACM SIGCOMM (2001)
9. Freedman, M.J., Lakshminarayanan, K., Mazières, D.: OASIS: Anycast for any service. In: Proceedings of USENIX/ACM NSDI (2006)
10. Freedman, M.J., Morris, R.: Tarzan: a peer-to-peer anonymizing network layer. In: Proceedings of ACM CCS (2002)
11. Granboulan, L., Pornin, T.: Perfect block ciphers with small blocks. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 452–465. Springer, Heidelberg (2007)
12. Java Anon Proxy, http://anon.inf.tu-dresden.de/
13. Katti, S., Cohen, J., Katabi, D.: Information slicing: Anonymity using unreliable overlays. In: Proceedings of USENIX NSDI (2007)
14. Lee, D., Brownlee, N.: Passive measurement of one-way and two-way flow lifetimes. SIGCOMM Comput. Commun. Rev. 37(3) (2007)
15. Levine, B.N., Reiter, M.K., Wang, C., Wright, M.K.: Timing attacks in low-latency mix-based systems. In: Juels, A. (ed.) FC 2004. LNCS, vol. 3110, pp. 251–265. Springer, Heidelberg (2004)
16. Levine, B.N., Shields, C.: Hordes — A Multicast Based Protocol for Anonymity. Journal of Computer Security 10(3) (2002)
17. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable block ciphers. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, p. 31. Springer, Heidelberg (2002)
18. McCoy, D., Bauer, K., Grunwald, D., Kohno, T., Sicker, D.: Shining light in dark places: Understanding the Tor network. In: Privacy Enhancing Technologies Symposium (July 2008)
19. Meyer, D.: Route Views Project. http://antc.uoregon.edu/route-views
20. Moore, D., Periakaruppan, R., Donohoe, J., Claffy, K.: Where in the world is netgeo. caida.org? In: Proceedings of INET (2000)
21. Moskowitz, R.: Host identity payload. Internet Draft, IETF (Feburary 2001), draft-moskowitz-hip-arch-02.txt (expired)
22. Murdoch, S.J., Danezis, G.: Low-cost traffic analysis of tor. In: Proceedings of the IEEE Symposium on Security and Privacy, pp. 183–195 (2005)
23. Privoxy, http://www.privoxy.org/
24. Proxify, http://proxify.com/
25. Raymond, J.-F.: Traffic analysis: Protocols, attacks, design issues and open problems. In: Federrath, H. (ed.) Designing Privacy Enhancing Technologies. LNCS, vol. 2009, pp. 10–29. Springer, Heidelberg (2001)

26. Reiter, M.K., Rubin, A.D.: Anonymous web transactions with crowds. Commun. ACM 42(2), 32–48 (1999)
27. Serjantov, A., Sewell, P.: Passive attack analysis for connection-based anonymity systems. In: Snekkenes, E., Gollmann, D. (eds.) ESORICS 2003. LNCS, vol. 2808, pp. 116–131. Springer, Heidelberg (2003)
28. Sherwood, R., Bhattacharjee, B.: P5: A protocol for scalable anonymous communication. In: Proceedings of IEEE Symposium on Security and Privacy (2002)
29. Shmatikov, V., Wang, M.-H.: Timing analysis in low-latency mix networks: Attacks and defenses. In: Gollmann, D., Meier, J., Sabelfeld, A. (eds.) ESORICS 2006. LNCS, vol. 4189, pp. 18–33. Springer, Heidelberg (2006)
30. Sirer, E.G., Goel, S., Robson, M., Engin, D.: Eluding carnivores: file sharing with strong anonymity. In: Proceedings of the ACM SIGOPS European workshop (2004)
31. Stoica, I., Adkins, D., Zhuang, S., Shenker, S., Surana, S.: Internet indirection infrastructure. In: Proceedings of ACM SIGCOMM (2002)
32. Trostle, J., Way, B., Matsuoka, H., Tariq, M.M.B., Kempf, J., Kawahara, T., Jain, R.: Cryptographically protected prefixes for location privacy in ipv6. In: Proceedings of the Privacy Enhancing Technologies Symposium (2004)
33. WIDE Project, `http://www.wide.ad.jp/`
34. Zero Knowledge Systems Freedom Network, `http://www.zks.net/`

A Case Studies

In this section, we briefly discuss the use of AHP with common applications, and in post-hoc forensic analysis. To provide anecdotal evidence that use of AHP presents no user-perceived changes in application behavior or performance degradation, one of the authors of this paper spent an afternoon using several ordinary applications that were on a network segment behind our prototype AHP gateway. We used several typical network applications—Mozilla Firefox, BitTorrent, SSH, and XChat—and an existing anonymity system—Tor[13].

Firefox and XChat. Firefox ran normally, as did XChat—both only open ordinary HTTP connections and perform DNS queries, which are translated properly by AHP in hidden mode. There was no noticeable slowdown in browsing performance. While this is only anecdotal, we found no cases in which a Web page failed to load as usual.

Tor. Though Tor opens a listening socket on the local machine (which is the port on which it accepts SOCKS connections), since all connections are local, AHP does not interfere. As recommended, we used Tor with Privoxy [23] which presents an HTTP proxy interface to Tor and performs application-level privacy

[13] To accommodate for the fact that when connecting to the Internet, our broadband Internet service only has a single IP address assigned to it, we set up a small local network within which to perform address hiding before routing to the Internet. We connected the user's machine directly (over Ethernet) to a second machine which served as the AHP gateway. Given the constraint of only one public IP address, we performed address hiding within the local subnet and then performed NAT (which was needed since the public IP address space is much smaller than the private address space in this scenario) before actually sending packets out to the public Internet.

filtering. We were able to browse the Web as usual via Tor with no interference from AHP. As a result, those Internet users who desire additional privacy beyond the capabilities provided by AHP will still be able to enjoy the benefits of Tor and similar overlay anonymity services.

BitTorrent. BitTorrent performs best when it can make outgoing connections to other peers *and* accept incoming connections. To enable inbound connections, we ran the official BitTorrent client in sticky mode using `expose`. While this did require one additional step—adding `expose` to the command line—we believe this step is not onerous, and furthermore, we believe the changes required here are even less than those to support transition to NATs (for which many applications had to be updated). *No* code needed to be changed in the BitTorrent application itself. Regardless, our main observation is that Internet users can continue to use peer-to-peer applications if their ISPs offer AHP to them as a service. One concern is that sticky mode incurs overhead at the AHP gateway, as the gateway has to process and store requests on behalf of applications that wish to accept inbound connections. To discover whether BitTorrent requires repeated or burdensome communication with the AHP gateway when in sticky mode, we logged its system calls while downloading the top ranked torrent from a popular BitTorrent website. We left all application settings at their defaults. Over the course of the approximately 30 minute download, BitTorrent accepted 2,034 inbound TCP connections, made 3,980 outbound TCP connections, and yet only needed to `bind()` a listening TCP socket exactly once. (That one call to `bind()` initialized a sticky IP/port pair in the AHP gateway.)

Forensic recovery. Recall that one of the principal goals of AHP is to enlist ISPs to help improve users' Internet privacy while also still allowing ISPs to be consistent with existing or emerging government legislation and internal corporate policies. AHP thus, for example, enables ISPs to easily respond to requests. We built a simple forensic tool that performs the same operation as an AHP gateway to Unhide the address and port of a given packet—the packet is read in from a user-specified file. The only remaining information—the timestamp and which side initiated the flow—must be provided to the tool so it can select the key for that timestamp. Though collisions can occur, they only occur across different remote IP/port tuples. For a given (remote) destination IP address and port, there exists only one permutation at a given gateway, and thus a unique mapping for each hidden address that is communicating with that IP/port pair.

B Flexible Hiding Sets

A fundamental property of in-network, directed-routed anonymity or address hiding systems is that their hiding sets are firmly tied to route advertisements. Thus, if an ISP only advertises small IP prefixes, and cannot or does not aggregate them with adjacent prefixes, then the address space within which a user hides is small, leaving the user open to a host of de-anonymization attacks. At the other extreme, a large ISP with a large IP prefix (such as a /8) can leverage

the expansive IP space to hide all its customers even across continents. This tension between presenting larger hiding sets for users and enabling fine grained route control is not a new one—network engineers in ISPs try to optimize their routing advertisements to maximize routing flexibility (by advertising small prefixes) while also considering route stability, convergence, and update overhead (by advertising large prefixes). Thus the consideration of selecting the appropriate size IP prefix to advertise to aid in user address hiding is not a new, undue burden upon network administrators and engineers.

B.1 Variable Prefix Sizes

Due to the lack of variable block-size block ciphers to use as the underlying PRP, the span of IP prefix sizes we can support using the exact techniques we describe above directly using the block cipher are limited. In our prototype implementation, we first built support for 16 bit externally visible IP prefixes, which, along with a 16-bit port field, are appropriate for a cipher with a 32-bit block width.

However, to provide ISPs greater flexibility, a deployment implementation would need to support a variety of IP prefix lengths. The shuffle-based random permutation design of Granboulan and Pornin [11] lets us select arbitrary size prefixes, though their algorithm is computationally expensive. In hardware, a table-based permutation is appropriate for small IP prefixes.

We extended our prototype to cope both with IP prefixes larger than and smaller than a /16 prefix. For those smaller, we were required to perform a slight layer violation, and include the high-order bits of the TCP timestamp field in the block permuted by the cipher (to pad the IP/port pair up to the 32-bit mark). For those prefixes larger than /16, we restrict ourselves to even bitlength IP prefixes, and build a Luby-Rackoff cipher using a larger block-size primitive—in our case AES—as a PRF; such a design comes with the corresponding loss of security due to the PRF to PRP transformation.

B.2 Disjoint Prefix Aggregation

A fundamental problem with operating strictly on IP prefixes is that an ISP may not use strictly neighboring address spaces, and thus, may be forced to advertise them separately. Instead, we suggest that with a slightly modified version of AHP, we can aggregate multiple smaller, disjoint prefixes into one IP address space over which AHP can anonymize clients. Our primary approach is to revisit the notion of permuting addresses as opposed to encrypting them. By permuting addresses, we eliminate the need for strict, IP prefix-based partitioning.

The following simple technique would allow the aggregation of disjoin prefixes: given n prefixes $(p_1, \ldots, p_n)$, we map the addresses via a bijection to $\mathbb{Z}_k^+$ where $k = |p_1| + \cdots + |p_n|$. We then apply a permutation such as that of Granboulan and Pornin [11] to $\mathbb{Z}_k^+$ and remap the result back to the original prefixes.

As a result of this approach, anonymity sets can be made as large an ISP's entire address space. We leave a thorough study of disjoint prefix aggregation to future work.

C Key Rotation

In AHP we must rotate keys if we are to protect the unlinkability of flows originating from the same true source address to the same destination address over time. How often should key rotation occur? The epoch length, which we define to be the period of key rotation, constrains the maximum length of a flow. Too short an epoch will unduly constrain flow durations—too long and it may allow port reuse at end hosts and thus privacy loss.

Suppose a client creates a new socket to the same port on a server repeatedly. Though most server applications typically set the `SO_REUSEADDR` socket option, thereby allowing port reuse even before TCP fully flushes its state for a particular port, most client applications allow the operating system to select a random port. Under that assumption, and with the additional constraint that client applications do not have the necessary rights to use the first 1024 port numbers, $k = 64512$ ports are available. Allowing the full 2 minutes for each port to become available again after use, we can compute the time required to reuse a port with probability 0.5 using the Taylor series approximation of the birthday paradox: $p(n) = 1 - e^{-\frac{n(n-1)}{2k}}$. Substituting, we compute n to be 299 attempts, which indicates that we should expect a port collision to occur after 35886 seconds, or roughly 10 hours. Naturally, then, we would like for the counter value that we use to increment in 10 hours or less. We discuss selection of a specific key rotation interval later in the context of collision analysis.