Scavenging for Anonymity with BlogDrop

Henry Corrigan-Gibbs and Bryan Ford

Yale University
New Haven, CT, USA

Abstract. Anonymous communication schemes that provide strong traffic analysis resistance (e.g., DC-nets and Mix-nets) are too slow for large-scale interactive use. Low-latency systems (e.g., Tor) provide users with alarmingly small anonymity set sizes, especially in heavily monitored networks. We present preliminary work on a new anonymity protocol that makes this anonymity/latency trade-off an explicit and user-adjustable parameter. We call our technique *anonymity scavenging*: the longer a user is willing to wait for a message to leave the system, the more anonymity that user will be able to scavenge from other users. Our protocol builds upon previous provable schemes, and we expect formal proofs of security to be a large component of future work.

1 Background

Existing anonymity systems offer dangerously small anonymity set sizes for users in heavily monitored networks. Although there are around 1,000 daily Tor users in China [12], many fewer than 1,000 users will be online at any moment. Our approximation, based on browsing behavior statistics [10], suggests that Tor has fewer than 90 active users in China in any given 5-minute interval.

Even though there are 1,000 daily users of Tor in China, and probably many more than 1,000 distinct users in a week, low-latency anonymity systems [6, 1, 8] do not allow latency-insensitive users to take advantage of these larger potential anonymity sets. The anonymity set size a user gets is only as large as the number of users online at the moment.

Consider Alice, a Tor user in China, who anonymously posts a sensitive article on an online bulletin board. If the content of Alice's article is of local interest (e.g., discussing a corrupt provincial official), government censors could guess, with some confidence, that that the author is inside of China. The censors could use the timestamp on Alice's post, in conjunction with usage data from local ISPs, to identify the 90 Tor users who were online when the article was posted. The geographical locations of these 90 Tor users might allow authorities to narrow their search further, until Alice is anonymous only within a handful of other Tor users.

We present our preliminary work on BlogDrop, an anonymity protocol that treats anonymity set size—and hence, latency—as a user-tunable parameter. Users who want lower-latency communication can settle for smaller anonymity set sizes (e.g., the 90 users who are online right now). Users who want more secure communication can request much larger anonymity set sizes (e.g., the 1,000 users who will be online in the next 24 hours). BlogDrop achieves these properties by combining a non-interactive DoS-resistant DC-net construction [9] with a client/server DC-net topology [13].

One attractive feature of our design is that, to post a BlogDrop message, clients need only to "drop off" a ciphertext at a server. Clients need not interact with the server (beyond this one-way bit drop-off) nor with other clients. We have not attempted a formal proof of security of the BlogDrop protocol, but we expect that this non-interactivity, and its basis on techniques with formal proof [9, 2] will greatly simplify BlogDrop's formal analysis.

Batching mix-net schemes [3, 5] can also achieve anonymity scavenging, but they require $O(n)$ storage for an anonymity set size of n; BlogDrop reduces the storage required to $O(1)$. BlogDrop's end-to-end number of serial communication rounds is constant in the number of servers, while mix-nets require $O(k)$ communication rounds for networks of k servers.

2 Design Overview

The BlogDrop protocol takes place between *posters* and *servers*. Each server maintains a "bin" for each blog and we assume that at least one of the servers is honest [13]. To contribute to a blog, a blog poster "drops" a ciphertext into the bin of a participating server. One of these blog posters is the secret author, whose ciphertext conceals

the plaintext blog message. The rest of the blog posters submit cover ciphertexts that are cryptographically indistinguishable from each other and from the author's ciphertext.

Once a server has enough valid ciphertexts (we explain what "enough" means later on) in its bin, the server shares the contents of its ciphertext bin with all of the other servers. When *every* server's bin contains enough ciphertexts, the servers take the union of their bins to determine the final ciphertext set. Finally, each server drops its *own* server ciphertext into the bin, along with ciphertexts from every other server. The server then combines this final set of client and server ciphertexts together to reveal the plaintext message. The servers then empty their bins and repeat the process.

To create the bins, we use a DC-net scheme that combines elements of Golle's non-interactive construction [9] and Wolinsky's client/server key-sharing graph [13]. When a poster drops off a ciphertext, the poster submits a non-interactive proof that the ciphertext is either a properly constructed cover ciphertext *or* that the poster knows the blog author's secret key. We implement this either/or proof using standard techniques [2] and we present a sketch of our ciphertext and proof construction in Appendix A.

Intuitively, the more ciphertexts there are in the bin, the larger potential anonymity set size the author can achieve. All participants know *some* poster submitted a plaintext message, but they do not know which one did. If the author is willing to wait until the bin has 1,000 distinct ciphertexts, then the author can hide among these 1,000 posters. In contrast, an impatient author who values speed over security might only require 10 ciphertexts to be in the bin before the servers collectively reveal the plaintext.

In fact, the blog author might specify more complicated bin *closure conditions* to prevent Sybil attacks or to encourage a more diverse anonymity set. For example, the blog author could request that servers close the bin only after 100 users with distinct VeriSign certificates *and* 15 users from Sweedish IP addresses *and* 200 other users have dropped ciphertexts in the bin *and* only when today's date is later than November 6, 2012. Since each server independently verifies that the closure condition is satisfied for each bin, and since we assume that at least one server is honest, the plaintext will remain indecipherable until the ciphertext bin satisfies the closure condition.

The BlogDrop protocol requires many posters to submit cover traffic but only one poster (the author) actually writes blog posts. Cover traffic posters could be readers of the blog, or they could be authors of other blogs hosted on the same set of servers. Servers could enforce a "pay-for-play" scheme, in which to read one blog, clients must post ciphertexts to a random set of other blogs. One area for further work is to investigate ways to incentivize non-authors to post ciphertexts and what the security consequences of these different incentives might be.

3 Blog Creation

To create a blog, the blog author creates a *policy document* for the blog and transmits it to the servers. The policy document defines: (1) the set of servers for the blog (identified by their public keys), (2) the pseudonymous author public key for the blog, and (3) the closure condition. The servers use information in the policy document to learn the closure condition for the blog and the clients use the policy document to construct their ciphertexts.

To maintain anonymity, the blog author must transmit the policy document *anonymously* to all of the servers. At first glance, it might appear that having to transmit the policy document anonymously leads to a "chicken-and-egg" problem: to blog anonymously, the author must anonymously transmit the policy document to the servers. Distributing the policy document is not as difficult as it sounds—primarily because the person who distributes the policy document never needs to participate in the BlogDrop protocol later on.

One potential policy document distribution scheme would use verifiable shuffles [11] or batching mix-nets [3]. Another distribution scheme would use offline social networks: the true author Alice could create a blog policy document and give it to her friend Bob, who then gives it to the servers. Since Alice does not sign the policy document with her public signature, the servers will not be able to cryptographically link the policy document back to Alice.

4 Conclusion

We have presented BlogDrop, an anonymity-scavenging communication protocol that allows users to adjust their anonymity set size to achieve their desired balance of security and latency. BlogDrop builds on formally analyzed DC-net and zero-knowledge proof techniques. We expect to apply these ideas and proof strategies to prove the security of BlogDrop in future work.

Acknowledgments

We wish to thank Michael J. Fischer, Ramki Gummadi, Ewa Syta, and David Wolinsky, for helpful discussion. This material is based upon work supported by the Defense Advanced Research Agency (DARPA) and SPAWAR Systems Center Pacific, Contract No. N66001-11-C-4018. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Agency (DARPA) and SPAWAR Systems Center Pacific.

References

1. Invisible internet project (i2p), January 2012. `http://www.i2p2.de/`.
2. Jan Camenisch and Markus Stadler. Proof systems for general statements about discrete logarithms. Technical Report 260, Dept. of Computer Science, ETH Zurich, March 1997.
3. David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2), February 1981.
4. Henry Corrigan-Gibbs and Bryan Ford. Dissent: accountable anonymous group messaging. In *CCS*, pages 340–350, October 2010.
5. George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a Type III anonymous remailer protocol. In *Security and Privacy*, May 2003.
6. Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: the second-generation onion router. In *13th USENIX Security Symposium*, Berkeley, CA, USA, 2004.
7. Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In George Blakley and David Chaum, editors, *Advances in Cryptology*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer Berlin / Heidelberg, 1985.
8. Sharad Goel, Mark Robson, Milo Polte, and Emin Gun Sirer. Herbivore: A Scalable and Efficient Protocol for Anonymous Communication. Technical Report 2003-1890, Cornell University, Ithaca, NY, February 2003.
9. Philippe Golle and Ari Juels. Dining cryptographers revisited. *Eurocrypt*, May 2004.
10. Ravi Kumar and Andrew Tomkins. A characterization of online browsing behavior. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 561–570, New York, NY, USA, 2010. ACM.
11. C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *8th CCS*, pages 116–125, November 2001.
12. Tor metrics portal. `http://metrics.torproject.org/`.
13. David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Scalable anonymous group communication in the anytrust model. In *5th EuroSec*, April 2012.

A Hybrid DC-Net

A.1 Description

We now briefly describe the communication primitive that the posters and servers use for the anonymous message exchanges.

We implement BlogDrop message "bins" using a hybrid DC-net scheme that combines design features from two prior constructions. Golle's DC-net [9] prevents DoS attacks by forcing participants to proactively prove the validity of their ciphertexts in zero knowledge. Wolinsky's DC-net [13] introduces a client/server key-sharing graph, and a few additional trust assumptions, to allow a DC-net exchange to continue even if many participating nodes fail. In particular, Wolinsky's construction requires that, to maintain security, *at least one* server is honest (though clients need not know which one is honest). We combine Golle's DoS-resistant DC-net construction with Wolinsky's client/server topology to create a DC-net that provides non-interactive DoS prevention and that allows for membership churn among the blog posters.

BlogDrop makes a few modification to this hybrid DC-net scheme to make it more suitable to anonymity scavenging. First, in a BlogDrop DC-net bin, there is only one transmitter (the blog author)—all non-author posters drop only cover ciphertext in the message bin. Since there is only one transmitter in the DC-net, there is no possibility for message collisions and there is no need to set up a transmission "schedule", as previous DC-net-derived schemes require [4, 13]. Having a single transmitter also means that the length of each message in each bin is constant in the number of participants. Blog posters can drop off ciphertexts without knowing how many other posters will drop off ciphertexts in the bin.

Second, there is no fixed or enumerated set of group members in the BlogDrop DC-net: the servers remain the same for a given blog, but the set of participating posters can change with every subsequent bin. Third, BlogDrop servers can incrementally collapse the contents of their bin into a constant-size combined ciphertext. (In contrast, mix servers must store n ciphertexts for a bin with n messages.) Having no fixed group membership means that the anonymity set size for a particular bin can become arbitrarily large and it can include participants who join after the blog's creation.

We have implemented the BlogDrop cryptographic primitives in C using Diffie-Hellman key exchanges in $\mathbb{Z}_p^*$ (as opposed to Golle's pairing-based instantiation [9]) using standard discrete-log proof techniques [2]. Generating 128 KB of poster ciphertext (including the zero-knowledge proofs) takes 5.2 seconds and generating 128 KB of server ciphertext takes 3.1 seconds on a modern workstation.

A.2 Ciphertext Construction

We outline the ciphertext construction technique for the hybrid DC-net without describing the specifics of the server-to-server communication protocol.

BlogDrop's ciphertext construction uses a finite cyclic group G and generator g (constant parameters). We denote the Diffie-Hellman public/private key pair (g^a, a) as (A, a). Each server j has a long-term well-known public/private key pair (B_j, b_j).[1] The policy document contains the author public key Y for the blog. Each poster i has a long-term well-known signing key pair that it uses to authenticate itself to the servers.

To create a ciphertext, message poster i generates a one-time-use key pair (A_i, a_i). The poster ciphertext is the product of the Diffie-Hellman shared secrets between poster i's one-time-use private key and the public key of every server:

$$C_{\text{poster},i} = \prod_{j \in \text{servers}} (B_j)^{a_i}$$

If the poster *is* the secret author, the poster multiplies the secret message m with the ciphertext, to produce the author ciphertext $mC_{\text{poster},i}$. The author then executes the rest of the ciphertext construction using $mC_{\text{poster},i}$ in place of $C_{\text{poster},i}$.

Using standard discrete logarithm proof-of-knowledge techniques [2], poster i proves that *either* $C_{\text{poster},i}$ is a correct cover *or* poster i is the blog author:

$$\text{PoK}\{a, y : \left(C_{\text{poster},i} = (\prod_{j \in \text{servers}} B_j)^a \wedge A_i = g^a \right) \vee Y = g^y\}$$

Poster i attaches the one-time-use public key, the proof of knowledge, the bin index t, and a hash of the blog policy document d to the final ciphertext and signs this tuple with the one-time-use key a_i:

$$\{A_i, C_{\text{poster},i}, \text{PoK}, t, \mathcal{H}(d)\}\text{SIG}_{a_i}$$

Poster i submits this ciphertext tuple to one of the servers. BlogDrop's use of a one-time-use public key to encrypt a ciphertext is similar to the technique used in ElGamal encryption [7], except that a BlogDrop ciphertext is encrypted with the public key of *every* server, not just with a single recipient's public key.

Once the servers have agreed upon a final poster ciphertext set, each server j uses the per-round public submitted by every poster to produce it's own server ciphertext:

$$C_{\text{server},j} = \prod_{i \in \text{posters}} (A_i)^{-b_j}$$

Server j proves the validity of $C_{\text{server},j}$ to the other servers by demonstrating that it used its long-term server private key b_j to generate its server ciphertext:

$$\text{PoK}\{b : C_{\text{server},j} = (\prod_{i \in \text{posters}} A_i)^{-b} \wedge B_j = g^b\}$$

The product of all poster and server ciphertext reveals the plaintext m.

[1] To prevent maliciously formed public keys, we require each server to non-interactively prove knowledge of their private key.