Anonycaster: Simple, Efficient Anonymous Group Communication

Christopher C. D. Head
chead@cs.ubc.ca

ABSTRACT

The ability to communicate without revealing authorship of messages has, in the past, often been considered by means of two approaches. One approach has been for users to trust a particular known entity or small number of entities not to disclose their identities; this is the solution used in virtual private networks or organizationally hosted communication fora, for example. The other approach has been to rely on probabilistic anonymity; this is the solution used by such systems as The Onion Router (TOR), wherein the user does not explicitly trust any of the nodes on the network but assumes that anonymity can be achieved by routing messages through a large enough set of untrusted nodes that it is improbable that all nodes have been compromised.

In this paper, I introduce a system that exists between these two extremes. Rather than relying on a central authority to protect anonymity, cryptographic primitives are used to protect messages in transit such that the "network" (including any relay server) need not be trusted. However, rather than relying on a set of completely unknown-to-the-user relay nodes, trust is distributed among the participants in the conversation itself, who are far more likely to be known to the user. Finally, anonymity in this system is deterministic: the only information a compromised party can divulge is whether a particular message was written by that party or whether it was written by someone else in the group; no matter how many parties are compromised, it is impossible to differentiate authorship of messages between the remaining benign parties.

1. INTRODUCTION

Today's Internet is used by many people desiring a way to communicate without being tracked or identified. From human-rights activists to corporate whistle-blowers to police tip submitters to callers of complaint lines, the availability of anonymity is critical to those who fear for their safety or, for any other reason, wish to publish material without their names attached.

Existing systems that provide anonymity to users tend to fall into one of two categories. The first category is systems which trust a single, central server and rely on a secure connection to that server. The second category is systems which rely on a large network of untrusted relay nodes.

These approaches represent two ends of a spectrum: at one end, a single entity is trusted absolutely; at the other end, a massive number of entities are completely untrusted and users of the system simply assume that an attacker cannot compromise or replace the entire network.

Anonycaster takes an approach which lies in the middle of this spectrum: trust is spread among multiple entities, but those entities are the other parties to the conversation. By spreading trust among multiple entities, Anonycaster eliminates the single point of failure inherent to the centralized-server model. I assume that the user likely has an existing relationship with at least some of the other participants in any given conversation; Anonycaster takes advantage of this existing relationship to yield a more trustworthy anonymity system than one that simply uses a network of relays whose operators are unknown to the user.

2. RELATED WORK

A number of systems exist which implement some sort of multi-user anonymous communication.

2.1 Centralized Server

One way to provide anonymity is to use a secure connection mechanism, such as Transport Layer Security (TLS) or Internet Protocol Security (IPSec), to hide the content of a message being sent to a central server, then trust the server to publish the message while discarding the user's identity. Unfortunately, this naïve solution, while easy to understand and implement, suffers from a number of drawbacks. First, unless care is taken to randomly reorder or time-shift messages, this mechanism is vulnerable to traffic analysis: an attacker who observes an encrypted connection being made to the central server followed shortly by the appearance of a new message may conclude that the message was sent by the connection and may then take action against the traced originator of the connection. Second, the central server represents a single node which must be trusted absolutely and, thus, a single point of failure: an attacker need only compromise the server in order to instantly reveal the authorship of all messages; even if the server actually discards authorship rather than simply displaying it, the attacker can reveal the

authorship of any new messages delivered after the compromise. An example of this kind of system is Wikipedia [1]: while all edits are logged and associated with a user, a visitor who creates a named account has their IP address hidden when they edit content and these hidden IP addresses are, by policy, available only to a very limited set of trusted people.

2.2 Tor

Dingledine et al. describe a protocol [7] for setting up a network of relay stations which forward Transmission Control Protocol (TCP) connections anonymously. A sender negotiates, hop by hop, a virtual circuit through a random selection of nodes; connection control commands and payload data are sent in fixed-length cells which are repeatedly encrypted using keys shared with the relays on the path. As the cell passes each relay, the relay uses the shared key to decrypt the contents of the cell then passes the cell to the next hop selected during virtual circuit setup. The last relay in the route will decrypt the cell and obtain the plaintext data to send to the target server.

Tor's security relies on at least one of the relays in the chosen route being trustworthy. The expected deployment model of Tor is a very large network of relays, with each client selecting a random subset of relays for a route. Statistically, the client expects that the probability of an arbitrary relay being compromised in such a large network should be quite small and, thus, the probability of an entire randomly-selected route being compromised should be smaller yet. Unfortunately, the generality implied by anonymizing arbitrary TCP connections limits Tor's ability to rely on the presence of nodes whose operators are personally known to participants in the conversation. In Tor, the expectation is that the user does not personally know any of the relay operators, which could leave the user open to an attacker replacing the entire Tor network with a malicious "fake Tor universe". This would be especially problematic if the user did not have access to a trusted network with which to bootstrap knowledge of benign relays. Anonycaster, on the other hand, solves this bootstrapping problem by placing trust in exactly the parties participating in the conversation, who I assume met each other in some other medium, perhaps in person, and can exchange any necessary cryptographic fingerprints at that time.

2.3 Anonymous Remailers

Many authors [4, 5, 6] have applied Tor-like techniques to the specific problem of anonymous e-mail, typically by building mail transport agents that decrypt, anonymize, and reroute mail. These are essentially reimplementations of Tor at a different protocol layer and, as such, suffer from the same limitations.

2.4 Multi-party Off-the-Record Messaging

The Off-the-Record (OTR) protocol [2] provides a mechanism by which two parties may communicate over an encrypted channel while enjoying perfect forward secrecy and while also rendering it impossible for anyone, including a participant, to prove whether or not a given message occurred as part of the original conversation; it achieves this property by revealing a key at the end of the conversation that allows anyone in the world to forge messages. The Multi-party OTR protocol [8] extends original OTR to more than two participants.

MPOTR, however, yields a different set of properties than Anonycaster. MPOTR is sold as an Internet equivalent to an in-person, face-to-face meeting: the set of participants is known and mutually authenticated and the authorship of every message is strongly proven during the conversation, but messages are kept confidential to the group and both authorship and membership can be plausibly denied once the conversation is terminated. MPOTR is thus perfect for situations in which deniability of *evidence* is the goal: in a democratic country with a fair court system, MPOTR does its job of reducing conclusive evidence of authorship to witness testimony of the same. This is not, however, the same as the actual anonymity provided by Anonycaster. There are many situations in which deniability is of little value compared to actual anonymity. One such situation is a country under a totalitarian regime which can arrest and imprison dissenters without resorting to the due process expected in democratic countries and without procuring actual evidence; another is the far more informal case of a group, perhaps a group of employees and their manager, raising anonymous complaints or grievances without fear of being harassed for "not being a team player" or "making trouble". Both of these situations are characterized by a potentially harassing party who doesn't need formal proof of authorship in order to carry out the harassment; true anonymity, on the other hand, prevents the harassing party from knowing whom to target.

2.5 Dining Cryptographers Networks

The Dining Cryptographers Network (DC-Net) protocol, described by Chaum [3] albeit not with that name, is a proposed network protocol using simple cryptographic primitives to establish an anonymous group communication with almost exactly the same properties as those attributed to Anonycaster. In fact, Anonycaster is based on Chaum's DC-Net theories but nails down implementation details and improves bandwidth efficiency somewhat over Chaum's proposed protocol. Chaum's paper describes a general approach that requires $\mathrm{O}\left(n^2\right)$ messages to be sent over the network for each round of protocol execution (where n is the number of participants). Sardroud, Dousti, and Jalili reduce this to $\mathrm{O}\left(n\right)$ by using a more complex protocol involving linear equations [9]. Anonycaster achieves $\mathrm{O}\left(n\right)$ message transmissions per round, just like Sardroud et al.'s work, but does so using much simpler computations involving only existing cryptographic primitives and exclusive-OR computations.

3. THREAT MODEL

In constructing Anonycaster, I assume there are N group members attempting to converse. I take $N = b + u + m + 1$, where b is the number of benign conversation partners whose cryptographic fingerprints the user has verified (either in person or by checking with other parties), u is the number of conversation partners whose fingerprints neither the user nor any of the b benign nodes has verified (if the user were to omit checking node x's fingerprint, but a benign node y whose fingerprint the user had checked were to check x's fingerprint, the user could simply ask y for x's fingerprint; thus, u is actually the nodes that lie outside the transitive

closure of fingerprint checking by benign nodes rooted at the designated user), m is the number of conversation partners who have been compromised by the attacker, and the remaining node is the user himself or herself (every node is a user of the system, but I designate one user as "the user" for clarity). I consider two classes of attacker.

The first class of attacker controls the network but does not have the ability to control or replace any of the end nodes; for this threat model, $u = m = 0$ and thus $b = N - 1$. This class of attacker can perform a denial-of-service attack against the group, but this attack can be reliably detected by the group members and differentiated from a normal silent channel. The attacker cannot perform any other attacks in this scenario.

The second class of attacker controls both the network and some number of nodes. Control over these nodes could be obtained because the user has neglected to verify their fingerprints; in this case, $u \neq 0$ and the attacker can replace these nodes by redirecting network traffic. It could also be because the nodes themselves have been compromised; in this case, $m \neq 0$ and traffic redirection is unnecessary. An attacker could also compromise different nodes using diferent approaches. These cases are equivalent in terms of the abilities of the attacker; the important observation is that $b < N - 1$ and the remaining $m + u = N - b - 1$ nodes are controlled by the attacker while being accepted as legitimate by the user (and, presumably, the remaining b benign nodes; consider the transitive-closure argument from earlier in this section). In this environment, the attacker has four capabilities. First, the attacker can perform a denial of service attack; in this case, however, the denial of service attack is undetectable as such and the attacker has the choice to masquerade the attack as either a completely silent channel in which no nodes are attempting to transmit, or as a very busy channel in which nodes are transmitting but messages are being corrupted by collisions. Second, trivially, the attacker may read messages sent to the group. Third, also trivially, the attacker may inject messages into the group's conversation. Finally, the attacker may, given a message sent by the user, determine that the user belongs to the $b + 1$ benign group members and not to the $m + u$ malicious group members; however, the attacker is unable to reduce the size of the set of possible authors below $b + 1$.

4. IMPLEMENTATION

4.1 The Dining Cryptographers Protocol

Chaum's Dining Cryptographers protocol [3] describes a protocol by which n participants may, between then, compute the exclusive OR of a set of input bits, one input per participant, without any participant learning the value of any other participant's input (assuming $n > 2$, obviously; if $n = 2$, one's own input and the exclusive-OR output are sufficient to calculate one's partner's input). This is achieved by having every two participants generate a pairwise random bit unknown to each other participant; each participant exclusive-ORs all their observed random bits along with their input bit and publishes the result. The party then exclusive-ORs all the published bits; the random bits serve to mask out the individual messages in the published bits, but once all the published bits have been combined, each random bit will appear exactly twice (once from each person who observed it) and will thus cancel, leaving only the exclusive-OR of the input bits.

4.2 Naïve Group Communication

A naïve extension of this protocol to the field of anonymous group communication consists of treating the published bits as an error-prone bitstream transmission medium and implementing a reliable network protocol on top of that medium. For example, one might transform a message into a sequence of bits, begin transmitting the bits of the message in consecutive rounds of the DC protocol, and check for collisions (instances where the transmitted and received bits differ); in the case of a collision, one could stop transmitting, wait for a random period of time, and try again. A simple implementation of this protocol, however, is very inefficient: if every pair of nodes must communicate over the network in order to flip a virtual coin, then for each round, all n nodes will communicate with all $n-1$ other nodes, for a total of $\mathrm{O}\left(n^2\right)$ messages sent over the network; in a non-multicast-capable network (such as the Internet), the publishing stage also requires each of the n nodes to send its published combination of bits to all $n - 1$ other nodes, another $\mathrm{O}\left(n^2\right)$ operation. This inefficiency is made worse by the fact that as the number of nodes increases, assuming all nodes are transmitting regularly, the chance of collisions increases as well, meaning the frequency of rounds must increase to compensate. Anonycaster resolves this inefficiency by taking advantage of simple cryptographic primitives to reduce the amount of communication necessary to run the protocol.

4.3 Coin Flip Generation

The first stage of the round, the generation of pairwise-shared random bits, is easy to solve, and Chaum hints at a solution in the original DC paper [3]. Anonycaster solves the pairwise-shared–random–bit–generation problem by establishing a symmetric pairwise key between each pair of nodes (by means of Diffie-Hellman, ensuring neither node alone may seriously influence the generated key); the nodes run the AES cipher in counter mode in lockstep in order to generate a random bitstream of arbitrary length. This requires that the nodes exchange Diffie-Hellman keys during system setup, but requires no communication at all while the protocol is running.

4.4 Publishing

In the original version of the DC protocol, the publishing stage is not considered in depth; it is assumed that the final message (the exclusive OR of the inputs) is to be revealed publicly and that each node broadcasts its published bit to the other nodes. [3] In practice, these assumptions are generally not true: wide-area networks (particularly the Internet) provide only unicast (or anycast, which is uninteresting to this discussion) addressing, but not, for most Internet users, multicast; also, in many cases, it may occur that the final message should be kept private rather than revealed to the world. Anonycaster solves both of these problems.

Replacing broadcasts in the original, publicly-revealed message model is quite simple: each node can send its published bit to a central server, which combines the bits and returns the result, providing $\mathrm{O}\left(n\right)$ network traffic. Solving the public revelation model is also quite simple: each node sends its

published value to other nodes encrypted, either separately for each recipient using a pairwise key or identically for all recipients using a key shared among the entire group; the recipients then decrypt the published bits and combine them to extract the final output.

Unfortunately, these solutions are not obviously compatible with each other: the solution to the broadcast problem requires that the central server have access to the published bits in order to combine them, while the solution to the public revelation model requires that nobody other than the group members be able to decrypt the publish messages. Anonycaster achieves both goals: a group-shared key is established (by means of multi-party Diffie-Hellman) and is used with AES in counter mode. Each node maintains its own counter value (which will never overlap between nodes) which it uses for encryption, and the counters run in lockstep. Each node exclusive-ORs its published bit with one bit of its keystream thus generated before publishing. The server then exclusive-ORs these encrypted values together and publishes the result, which is the exclusive OR of all the published bits *and* all the keystream bits. Similarly to how the DC protocol itself uses cancellation to remove exclusive-ORed masking bits, each node reconstructs the keystreams of *all* nodes (possible since they share a key and differ only in counter values) and exclusive-ORs each keystream bit in turn with the received value; each exclusive OR operation removes one of the original masks, eventually leaving only the exclusive OR of the published bits, i.e., the output of the DC protocol.

4.5 Tamper Detection

In subsection 4.4, it is assumed that the server is trusted to accurately perform the exclusive-OR of the encrypted published bits. A server that fails to do this can tamper with messages being communicated: although the server cannot read the messages, it can invert arbitrary bits simply by inverting the corresponding bits in its replies to the nodes. To prevent this, Anonycaster establishes a key that is shared among the group (this key is separate from the cipher key described in subsection 4.4); this key is used to attach a MAC tag to each message before the message is injected into the DC medium. Because the key is shared among the entire group, this MAC does not reveal any information about authorship of the message, but it does guarantee that the message originated within the group. The MAC tag also allows nodes to avoid displaying garbage messages when collisions inevitably occur.

The fact that the medium is unreliable corrupt messages are a normal occurrence, due to collisions, opens the door to an undetectable denial-of-service attack by the server. The server need only sprinkle random inversions on the channel and no messages can be successfully delivered, but the nodes will believe the failure to be caused by collisions, not a malicious server. To close this hole, the nodes agree during setup on a *silence threshold* defining a percentage of rounds that must always produce an output of zero. The choice of exactly which rounds will be silent is made by a lockstep pseudorandom number generator shared between all nodes, preventing the server from knowing which rounds to avoid corrupting; because all clients exclude these rounds from message transmission and always provide inputs of zero, a nonzero output could only have been produced by a malicious server.

4.6 Key Setup

The above implementation details describe a number of keys that need to be established, for ciphers, MACs, and lockstep pseudorandom number generators. These keys can be divided into two categories: some keys are shared between exactly two nodes (pairwise keys), and other keys are known by all nodes in the group (group keys).

When joining a group, a node begins by publishing two freshly-generated Diffie-Hellman public keys. The first of these keys is used for establishing pairwise keys; these public keys are each distributed to all nodes, who then perform $n - 1$ pairwise Diffie-Hellman agreements producing a shared premaster secret with each other node. The second set of Diffie-Hellman public keys is rotated through the nodes by the server where the nodes perform a multi-party Diffie-Hellman agreement, ending with all n nodes agreeing on a group premaster secret that is known to nobody outside the group.

These premaster secrets are then each used to derive a cipher key, a MAC key, and pseudorandom number generator key, and a verifier string, by concatenating the premaster secret with a diversifier flag specific to the type of key and passing the result through a hash function. Finally, to authenticate its identity and participation, each client signs the complete list of usernames and Diffie-Hellman pairwise agreement keys along with the group verifier to attest that the key setup protocol has completed successfully; the signatures are distributed to all nodes for verification.

5. SECURITY

I here describe how the mechanisms used by Anonycaster constrain an attacker to only those capabilities described by the threat model in section 3. There are four basic attacks an attacker might carry out against Anonycaster: denial of service, deanonymization of authorship, reading of messages, and injection of messages. Where appropriate, I divide analysis into two cases that parallel the cases in section 3: cases where the attacker does not control any nodes (only the network and/or server), and cases where the attacker controls at least one participating node.

5.1 Group Premaster

If the attacker controls a node, it will obtain the group premaster as part of the setup phase. Assuming the attacker does not control a node, I describe why it is infeasible for the attacker to obtain the group premaster.

The group premaster is the output of the multi-party Diffie-Hellman exchange; obtaining the group premaster thus requires either compromising the Diffie-Hellman exchange algorithm, disrupting execution of the algorithm to convince one of the participants to reveal data it should not, or obtaining one of the Diffie-Hellman private keys. I assume that the algorithm itself is secure, and thus compromising the algorithm is infeasible as is recovering a private key from a public key (or from a node, since no nodes are compromised).

One possible avenue of attack would be for the attacker to attempt to recover the group premaster by disrupting the order in which intermediate keys are delivered to nodes during key setup. However, this is not possible. First, observe that all nodes must end with the same group premaster: any deviation from this requirement will be revealed when the verifier (derived from the group premaster by a hash function) is included in the signature comprising proof of participation in the key setup protocol, and the hash and signature algorithm used therein are assumed infeasible to compromise (as are the ECDSA private keys used to generate the signatures, since no node is compromised). Second, observe that, assuming node i's Diffie-Hellman private and public keys are x_i and g^{x_i} respectively, the group premaster can only be $g^{\prod_{i=1}^{n} x_i}$: each node i constructs its copy of the premaster by raising its last intermediate key to the power x_i, thus for all i, x_i must be a component of the premaster; the attacker may introduce additional exponents into the final product if he or she wishes but such additional exponents must either all end up equal (in which case the attacker has gained nothing, as the attacker must add those exponents *last* in order to recover the shared secret but it will never have the ability to carry out that calculation) or must end up unequal (in which case the verification stage will fail); and finally, it is not possible to include one or more of the private keys more than once in the premaster because doing so requires using up one of that node's key-processing cycles which must necessarily *remove* that node's private key from another intermediate key, resulting in unequal premasters when the protocol finishes. Third, observe that, given this definition of the premaster, a node can be convinced to publish the premaster by passing it what should be its final intermediate key earlier, but doing so will use up one of the node's key-processing cycles, again removing that node's private key from some other intermediate key which requires it and resulting in unequal premasters.

The group premaster itself is only ever used as input to a hash function in combination with a diversifier flag; thus, recovering the group premaster from its outputs requires compromising the hash function.

5.2 Pairwise Premaster

If the attacker controls one or more nodes, it will obtain all pairwise premasters that relate to one of those nodes as part of the setup phase. I describe why it is infeasible for the attacker to obtain any pairwise premaster owned by two nodes neither of which is compromised.

The pairwise premaster is the output of a Diffie-Hellman exchange; obtaining the pairwise premaster thus requires either compromising the Diffie-Hellman exchange algorithm, disrupting execution of the algorithm, or obtaining one of the Diffie-Hellman private keys. I assume that the algorithm itself is secure, and thus compromising the algorithm itself is infeasible as is recovering a private key from a public key (or from a node, since neither of the two involved nodes is compromised).

It is not possible for the attacker to disrupt the execution of the algorithm. Just two messages are sent over the network during pairwise key negotiation: each node sends its Diffie-Hellman public key to the other node. Should the attacker modify one or both of these keys, the modification will be detected because the keys are included in the signature comprising proof of participation. One of the signatures on this proof must come from the other node involved in the pairwise agreement; since that node is uncompromised, the attacker cannot forge is signature as he or she lacks the ECDSA private key.

5.3 Key Derivation

The keys used for the main protocol, as well as the group verifier used in the proof of participation, are derived from the relevant premasters by concatenating a diversifier flag (identifying the purpose of the derived key) and applying a hash function. An attack on the generation of these derived keys thus requires either compromising the hash function (assumed infeasible) or obtaining the input to the hash function (the premaster, which, as described in the prior two subsections, is infeasible for an attacker who has not compromised a node that would have legitimate access to said premaster).

5.4 Key Usage

The cipher and PRNG keys are used to key the AES block cipher algorithm running in counter mode to produce a pseudorandom bitstream. Any attack which allows the key or additional keystream to be revealed would constitute a known-plaintext attack on AES (since the "plaintext" is a lockstep counter which the attacker is assumed to know). Furthermore, for the encryption keys, care is taken that each counter value for a given key is only used once, preventing an attacker from using already-observed keystream to reveal other plaintext.

The MAC keys are used to key the HMAC-SHA512 algorithm. It is assumed that this algorithm does not permit recovery of the key from a set of its inputs and outputs.

5.5 Message Injection

Message injection is trivial if the attacker controls a node; all nodes in the group have the ability to send messages, so the attacker simply sends the message through the compromised node.

If the attacker controls only the network, he or she clearly has the physical capability to modify messages; the question, then, is whether messages can be injected *undetectably*. The protocol defends against message injection by attaching a MAC tag to each message, where the MAC is keyed by the group MAC key. The attacker can thus inject a message by forging a proper MAC tag, which requires either compromising the MAC or obtaining the group MAC key. As shown in the preceding sections, this is infeasible.

5.6 Denial of Service

The attacker can trivially cause denial of service because the attacker is assumed to control the network. The question is whether the attacker can cause denial of service that masquerades as normal activity.

If the attacker controls only the network and no nodes, this is impossible. Corrupting the protocol framework itself will obviously be detected. The only place where corruption is

expected by participating nodes is in the placing of messages onto the DC medium; here, corruption is expected because collisions can occur. The server could conceivably corrupt the medium to prevent messages from flowing; however, this will be detected as soon as the server corrupts a silent round. The choice of which round is silent is made by a lockstep PRNG keyed by the group PRNG key; for the server to compute the output of this PRNG would require it to compromise AES or obtain the group PRNG key, both of which are infeasible. Thus, the denial of service attempt will be detected with high probability by the nodes.

5.7 Message Revelation

Message revelation is trivial if the attacker controls a node; all nodes in the group have the ability to read messages, so the attacker simply extracts the message from the compromised node. I therefore assume the attacker only controls the network.

During normal operation, the attacker observes two kinds of messages flowing over the network: each node's masked and encrypted inputs, and the encrypted output (which the server generates). Ignoring the masking entirely, each node's inputs are encrypted with AES in counter mode using a unique substring of the keystream that is never used for any other message. Thus, removing the encryption and revealing the masked input requires computing this keystream, something which is infeasible because the server does not possess the group cipher key. The encrypted output is derived from the encrypted and masked inputs by a simple exclusive-OR and hence provides the attacker with no further information beyond what they could obtain from the encrypted and masked inputs alone.

5.8 Deanonymization

An attacker who controls a subset of the nodes is obviously able to determine whether or not a message came from within that subset. The interesting deanonymization question is whether such an attacker can differentiate between uncompromised nodes with respect to message authorship. Because the attacker may be a member of the group, the encryption of masked inputs is irrelevant; security against deanonymization is provided entirely by the mask. Furthermore, the attacker can clearly remove the masks generated by the pairwise PRNGs belonging to a compromised node; thus, in the following analysis, I consider the nodes' selected inputs masked only by those PRNGs that are shared between two uncompromised nodes.

First, the mask bits cannot be computed by the attacker. The mask bits are generated by AES in counter mode using a key shared only by the two involved nodes and are never reused except between those two nodes; thus, obtaining the mask bits would require either obtaining the key or else predicting the output of AES for an unknown key, both of which are infeasible.

Given a set of values published by uncompromised nodes, with each value being masked by the outputs of the pairwise PRNGs shared between the publishing node and all other *uncompromised* nodes, it is infeasible for the attacker to determine which of the uncompromised nodes published a particular value. The original message value is either zero or one. After applying the first pairwise PRNG, since the PRNG is assumed to output unbiased independent bits, the masked value has exactly equal probability of being the same as, or different from, the original message value; furthermore, the only nodes who can uncover the original message value are the publishing node and the other node in the pair. After applying the second PRNG, the same argument applies except that the "original message" in this case is actually the masked message from the first PRNG application; the PRNGs are independent, and the message can only be revealed by removing both masks, which requires cooperation from either the publishing node or else *both* paired nodes. Once all the masks have been applied, the original message can only be revealed by cooperation from all paired nodes.

6. CONCLUSION

Anonycaster sits at a point in a space of communication mechanisms, from cryptographic to trusted-authority, from anonymous to probabilistically anonymous to pseudonymous to identified, from private to group-readable to publicly readable. Anonycaster provides group-readable communication, full anonymity of message authorship, and authenticated group membership, based on cryptographic primitives with no trusted central authority. Anonycaster builds on existing work by optimizing communication overhead for the existing protocols while not compromising on anonymity. Anonycaster provides the kind of deterministic guarantees of security that many solutions in this space do not, and the point at which it sits is ideal for a number of applications, from such simple examples as "complaints box-style" anonymous feedback to such serious matters as corporate whistleblowing or human rights activism.

7. REFERENCES

[1] Wikipedia:CheckUser. `https://en.wikipedia.org/wiki/Wikipedia:Checkuser`.

[2] N. Borisov, I. Goldberg, and E. Brewer. Off-the-record communication, or, why not to use PGP. In *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, WPES '04, pages 77–84, New York, NY, USA, 2004. ACM.

[3] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1:65–75, 1988. 10.1007/BF00206326.

[4] D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, Feb. 1981.

[5] L. Cottrell. Mixmaster & remailer attacks. `http://web.archive.org/web/20040209042623/http://obscura.com/~loki/remailer/remailer-essay.html`.

[6] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a type III anonymous remailer protocol. In *Security and Privacy, 2003. Proceedings. 2003 Symposium on*, pages 2–15, may 2003.

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

[8] I. Goldberg, B. Ustaoğlu, M. D. Van Gundy, and

H. Chen. Multi-party off-the-record messaging. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 358–368, New York, NY, USA, 2009. ACM.

[9] A. A. Sardroud, M. S. Dousti, and R. Jalili. An efficient DC-net based anonymous message transmission protocol. In *Proceedings of the 6th International ISC Conference on Information Security and Cryptology*, ISCISC '09, 2009.