2011 31st International Conference on Distributed Computing Systems

Confidential Gossip

Chryssis Georgiou
University of Cyprus
chryssis@cs.ucy.ac.cy

Seth Gilbert
National University of Singapore
seth.gilbert@comp.nus.edu.sg

Dariusz R. Kowalski
University of Liverpool
D.Kowalski@liverpool.ac.uk

***Abstract*—Epidemic gossip has proven a reliable and efficient technique for sharing information in a distributed network. Much of the reliability and efficiency derives from processes collaborating, sharing the work of distributing information. As a result of this collaboration, processes may receive information that was not originally intended for them. For example, a process may act as an intermediary, aggregating and forwarding messages from some set of sources to some set of destinations.**

But what if rumors are *confidential*? In that case, only processes that were originally intended to receive a rumor should be allowed to learn the rumor. This blatantly contradicts the basic premise of epidemic gossip, which assumes that processes can collaborate. In fact, if only processes in a rumor's "destination set" participate in gossiping that rumor, we show that high message complexity is unavoidable.

In this paper, we propose a scheme in which each rumor is broken into multiple fragments using a very simple coding scheme: any given fragment provides no information about the rumor, while together, the fragments can be reassembled into the original rumor. The processes collaborate in disseminating the rumor fragments in such a way that no process outside of a rumor's destination set ever receives all the fragments of a rumor, while every process in the destination set eventually learns all the fragments. Notably, our solution operates in an environment where rumors are dynamically and continuously injected into the system and processes are subject to crashes and restarts. In addition, the scheme presented can tolerate a moderate amount of collusion among curious processes without too large an increase in cost.

***Keywords*-Confidentiality, Collusion, Randomized gossip, Fault-tolerance, Dynamic rumor injection, Message complexity.**

I. Introduction

Collaboration is as the heart of distributed computing: when a network of devices cooperates to solve a problem, the resulting computation is often more robust and more efficient than if each device had worked independently. A classic example of the benefits of collaboration can be found in the paradigm of epidemic gossip. Consider, for example, a set of n devices that want to share information. If each device communicates independently with the other devices in the network, then the message complexity for the protocol is $\Theta(n^2)$. By contrast, if the devices collaborate to share the information, each communicating with a small number of random devices in each round, then the message complexity for the protocol can be reduced to $O(n \log n)$.

Yet there are some drawbacks to collaboration. One significant cost is *privacy*: by collaborating with other devices to solve a problem, it is often the case that private information is divulged. Consider again n devices that want to share information—however the information is potentially *confidential* and should only be shared among specified groups of recipients. For example, a user may want to share an engineering blueprint with her colleagues, but not with her competitors. Or a psychiatrist may want to send an e-mail to a group of patients, but not to everyone. Unfortunately, standard distributed protocols for efficiently sharing information do not satisfy these requirements. If the users rely on epidemic gossip to distribute their information, then all confidentiality is lost: every device in the system may learn every piece of information.

We focus on *honest, but curious* processes. This concept has attracted considerable attention as a model of processes in distributed applications that need limited anonymity and privacy, c.f., [30], [14] (see more in related work below). It is not our protocol's intent to be secure against truly malicious parties: if data must be kept secure in all circumstances, then a more expensive solution is needed. For everyday transactions, however, where privacy is desired, we can ensure that no process ends up in possession of information that it is not intended to learn. Moreover, we can achieve this additional guarantee at a relatively limited cost (in terms of message complexity), even if a moderate number of the participants may be colluding (i.e., sharing information).

A. Results

The question we ask is whetherit is possible to achieve the benefits of collaboration—i.e., robustness and efficiency—without sacrificing confidentiality. We focus on the problem of *Continuous Gossip*, a long-lived version of information sharing (introduced in [13]) that has three notable properties: (1) any process can inject a *rumor* at any time; (2) each rumor specifies a set of recipients that should receive the rumor; and (3) each rumor has a deadline specifying by when it should be received. In this paper, we present a continuous gossip protocol that guarantees:

- ***Confidentiality:*** Only the specified recipients of a rumor learn the contents of the rumor, even if processes outside the specified recipients may collude.
- ***Timeliness:*** Every rumor is delivered by the deadline.
- ***Efficiency:*** The maximum *per-round message complexity*, with high probability, and in the absence of collu-

1063-6927/11 $26.00 © 2011 IEEE
DOI 10.1109/ICDCS.2011.71

sions, is $O((n^{1+48/\sqrt{dmin}} + n^{1+6/\sqrt[6]{dmin}})$ polylog $n)$, where $dmin$ is the shortest deadline of any *active* rumor (that is, a rumor whose deadline has not expired); note that for rumors with deadline of $\Omega(\log^6 n)$, this results in per-round message complexity of $O(n \text{ polylog } n)$. When up to τ processes may collude, the maximum per-round message complexity grows by a factor of τ^2.

- ***Robustness:*** Processes may crash and restart at any time; there is no bound on the number of crashed processes at any given time. Moreover, failures are ***adaptive***: they may depend on the execution and the random choices made by the individual processes.

The major challenge underlying confidential gossip is reconciling the need for collaboration to achieve efficiency, and the inherent loss of confidentiality created by collaboration. At first glance, it seems that only recipients of a rumor can help in its dissemination. Yet, if we limit all information regarding the rumor to its recipients, then it is impossible to achieve good message complexity. As we show in Theorem 1, if we limit messages regarding a rumor ρ to the *destination set* $\rho.D$, then the per-round message complexity is $\Omega(n^{(3/2)-\varepsilon}/dmax)$, for any $\varepsilon > 0$ ($dmax$ is the longest deadline of any active rumor). Thus it seems that confidentiality and efficiency are inherently at odds.

We circumvent this seeming impossibility via a simple insight: each rumor can be divided into multiple independent fragments; each fragment provides no information regarding the original rumor, and yet together they can be combined to re-assembled the original rumor. (This is the basic idea underlying *cryptographic secret sharing* [33], [35], though we require only the simplest instantiation of this idea.) All the processes in the system can now collaborate to distribute the rumor fragments, as long as we ensure that no process collects all the fragments. (In fact, we can rely on existing gossip protocols as a black box.) In this way, we gain the benefits of collaboration without sacrificing confidentiality.

A second challenge is that failures are not independent and history-oblivious. We assume that processes may crash and restart at any time, and we model failures as being caused by an *adaptive* and omniscient adversary that can fail processes based on the random choices made by the protocol. For example, every time a source sends a rumor (or rumor fragment) to another process, the adversary may choose to immediately crash that recipient, entirely preventing the dissemination of that rumor. We address this challenge by having processes collaborate, exchanging *metadata* that contains no information on rumors. This information allows processes to determine which other processes have failed recently. Using this metadata, processes can target their messages better, and processes can adjust the number of messages they are sending. By collaborating on metadata, rather than rumors, processes can still overcome an adaptive adversary without giving up confidentiality. (While some information is leaked via the metadata, we discuss in Section VI how to avoid this problem.)

B. Alternative approaches and other related work

There are several possible cryptographic approaches to the problem of confidential gossip, many of which exist under the rubric of *multicast security* (e.g., [5], [11], [25], [28], [31], [33]). If a system must be secure against truly malicious parties (i.e., not simply "honest-but-curious" processes), then these cryptographic solutions are the only method of achieving confidentiality. The basic idea, in many cases, is that each process holds some subset of the cryptographic keys; by encrypting the message with appropriate subsets of the keys, the sender can ensure that the message can only be decrypted by the intended recipients. The cost of such a solution depends on the number of keys needed to encrypt the message for a given destination set.

In general, cryptographic solutions will be more efficient when the groupings are stable. That is, when some processes want to communicate with a fixed or slowly changing set of destinations (e.g., [2], [27], [34], [36]), these cryptographic solutions can be made quite efficient by ensuring that the set of destinations share a single cryptographic key. However, we are not aware of any sub-quadratic, in terms of message complexity, cryptographic approach to guarantee confidential gossip when the groups are changing rapidly, or when there are no fixed groups, i.e., when each rumor has a different destination set. In many cases, the best solution appears to be encrypting the message individually for each process in the destination set, thereby significantly increasing the amount of data to be sent. Furthermore, there is the question on how efficient secret key maintenance would be in the presence of dynamic crashes and restarts, especially when restarted processes have no memory of the computation prior to restarting (as assumed in our model). As we show, our confidential gossip protocol is efficient even under such dynamic adverse conditions.

The gossip problem has frequently been considered in relation to random, epidemic communication (e.g., [10], [18], [19], [20]). Each process periodically sends its rumor—along with any new rumors it has learned—to another randomly selected process. This approach can lead to efficient rumor dissemination while tolerating benign failures ([18]).

The gossip problem has also been considered in a variety of fault-prone environments, ranging from crash failures to malicious/Byzantine ones (e.g., [6], [16], [22], [24], [26]). The survey by Pelc [32] together with the book by Hromkovic et al. [17] overview solutions for the gossip problem in fault-prone distributed networks.

Another related line of work is the problem of constructing scalable overlays of topic-based Publication/Subscribe systems (e.g., [1], [7], [8], [29]). The aim is to design an overlay for each pub/sub topic, so that for each topic, the subgraph induced by the nodes interested in the topic will be connected. New events for each topic can then be

routed from publishers to interested subscribers using such "topic-connected" overlays. If destination sets are viewed as topics, then a topic-connected overlay could provide a confidential way of distributing a rumor to its destination set. Unfortunately, we don't know how to maintain topic-connected overlays in a dynamic setting (i.e., for changing destination sets), and even the static case is NP-complete [8], [29]. Theorem 1 (Section III) effectively implies that topic-connected overlays cannot be used to support efficient confidential gossip.

The *honest-but-curious* model, also refered as the *semi-honest* model [4] is a standard cryptographic adversarial model [14]. This model has been widely considered in the problem of multi-party privacy-preserving computation of some function [37], [14]. Our solution to the confidential gossip problem can be viewed as a tool for multiparty computation when the privacy of inputs (in the form of rumors) could be kept within groups of processes. For example, a group of social networking websites, wishing to efficiently calculate aggregate statistics such as degrees of seperation and average number of acquaintances without compomising the in-group privacy, could use as a building block our confidential gossip algorithm.

II. Model and Definitions

We consider a distributed system consisting of n synchronous processes that can communicate via message-passing over a reliable network, where each process can communicate directly with each other process. Messages are not lost or corrupted in transit. Processes have unique ids from the set $[n] = \{1, \ldots, n\}$.

The computation proceeds in synchronous rounds. In each round, each process can: (i) send point-to-point messages to selected processes, (ii) receive a set of point-to-point messages sent in the current round, and (iii) perform some local computation. We assume that processes have access to a global clock, that is, rounds are globally numbered.

Processes may crash and restart dynamically as an execution proceeds. Each process is in one of two states: either `alive` or `crashed`. When a process is crashed, it does not perform any computation, nor does it send or receive any messages. We assume that processes have no durable storage, and thus when a process restarts, it is reset to a default initial state consisting only of the algorithm to execute and $[n]$. Each process can only crash or restart once per round. We denote by $crash(p, t)$ the event in which process p crashes in round t. The event $restart(p, t)$ is defined similarly. We say that a process p is ***continuously alive*** in the period $[t_a,\ t_b]$ if: (a) process p is `alive` at the beginning of round t_a and at the end of round t_b, and (b) for every $t \in [t_a,\ t_b]$, there are no $crash(p, t, \cdot)$ events.

When a process p crashes in round t, some of the messages sent by p in round t may be delivered and some may be lost. When p restarts in round t, some of the messages sent to p may be delivered and some may be lost.

Rumors are dynamically injected into the system as the execution proceeds. A rumor ρ consists of a triplet $\langle z,\ d,\ D\rangle$, where z is the data to be disseminated, $D \subseteq [n]$ is the set of processes to which z must be sent (destination set), and d is the deadline by which the rumor must be delivered. We denote by $Inj(\rho, t, p)$ the event in which rumor ρ is injected to process p in round t. We refer to p as the *source* of rumor ρ. We assume that at most one rumor is injected at each process per round.

We model crash/restarts and rumor injection via a ***Crash-and-Restart-Rumor-Injection adversary***, or $\boldsymbol{CRRI}$ adversary. In each round, the adversary determines which processes to fail, which processes to restart, and which rumors to inject. The adversary is ***adaptive*** in the sense that it can make decisions in a round t based on the events in all prior rounds $< t$, as well as the random choices being made in round t itself. We refer to an *adversarial pattern* $\mathcal{A} \in CRRI$ as a set of crash, restart and injection events caused by adversary $CRRI$.

Definition 1 (Quality of Delivery): A gossip protocol satisfies a *quality of delivery* guarantee if every rumor ρ injected in round t at a process p is delivered no later than round $t + \rho.d$ to every process in $\rho.D$ that is continuously alive for $[t,\ t + d]$, if p is also continuously alive for $[t,\ t + d]$.

Definition 2 (Confidentiality): A gossip protocol is *confidential* if every rumor ρ is delivered only to processes in $\rho.D$, in every execution of the protocol.

Definition 3 (Per-round Message Complexity): A *randomized* algorithm $Rand$ subject to adversary $CRRI$ has *per-round message complexity* at most $M(Rand)$, if for every round t, for every $\mathcal{A} \in CRRI$, with high probability: the number of messages sent $M_t(Rand, \mathcal{A})$ by $Rand$ in round t, is at most $M(Rand)$.

Note that for randomized algorithms we guarantee Quality of Delivery and confidentiality, while achieving a probabilistic bound on the per-round message complexity. More on the rationale of Quality of Delivery and in general on the continuous gossip problem can be found in [13].

III. The Limitations of Strong Confidentiality

A gossip protocol is *strongly confidential* if for every rumor, no message causally dependent on that rumor is ever sent to a process that is not in the destination set of the rumor. In this case, only the processes in the destination set of a rumor can collaborate on that rumor's dissemination. This collaboration incurs a high per-round message complexity, even against an *oblivious adversary*:

Theorem 1: For all $\varepsilon > 0$, every randomized strongly confidential gossip algorithm has a maximum per-round message-complexity of at least $\Omega(n^{(3/2)-\varepsilon}/dmax)$, with probability 1, even against an oblivious adversary, where $dmax$ is the longest deadline of the injected rumors.

Proof: (sketch) Let $x = n^{1/2-2/c}$ and let $c = \lceil 2/\varepsilon \rceil$. Suppose that only rumors with uniform deadlines $dmax$ are injected. We show a lower bound $\frac{nx}{2c \cdot dmax} = \Omega(nx/dmax) \supseteq \Omega(n^{(3/2)-\varepsilon}/dmax)$. Suppose that each process is initially injected with a single rumor with random set of destinations: each process belongs to each destination set with probability x/n. The crucial argument is that under this scenario, with probability at least $1 - x^{2c+2}/n^{c-1}$, no message can carry more than c rumors. Having this, observe that the number of pairs $(source\_process, destination\_process)$ is at least $nx/2$, with probability at least $1 - e^{-nx/8} \geq 1 - 1/e$, by a Chernoff bound. It follows that the total number of rumor copies carried by messages is at least $nx/2$. Therefore, the total number of messages delivering these rumors is at least $\frac{nx}{2c}$, with probability at least $1 - 1/e - x^{2c+2}/n^{c-1} \geq 1 - 1/e - n^{-2} \geq 1 - 2/e > 0$. By the probabilistic method, there is a configuration of destination sets where the above claims are satisfied, and so it takes at least $\frac{nx}{2c}$ messages to deliver all these rumors. This occurs over $dmax$ rounds, thus there must be a round with at least $\frac{nx}{2c \cdot dmax}$ messages. Since there are no crashes/restarts, the adversary is oblivious. ■

In view of the upper bound $O(n^{1+6\sqrt[3]{dmin}}$ polylog $n)$ on continuous gossip without confidentiality assumptions [13] (against an adaptive adversary), we obtain a polynomial, in n, *price of strong confidentiality*, in terms of per-round message complexity (for minimum deadline $dmin > 24$). This motivates the study of the weaker version of confidentiality.

IV. Gossiping Continuously and Confidentially

In this section we present and analyze a continuous gossip algorithm, called ConGos, that guarantees that the content of rumors remains confidential under adversary $CRRI$. For simplicity, here we assume no collusion.

A. Algorithm ConGos

When a rumor is injected at a process p_i, the algorithm repeats the following procedure $\log n$ times concurrently: *Step 1:* Process p_i splits the rumor into two fragments such that only a process with both fragments can reconstruct the rumor. Processes are partitioned (deterministically) into two equal-sized groups. *Step 2:* Since process p_i itself belongs to one of the two groups, it uses a black-box continuous gossip service to share one of the half rumors with its own group. It uses a Proxy Service to distribute the other half rumor to the other group, with which it cannot gossip directly. At the end of the second step, each non-failed process has received one of the two half rumors. *Step 3:* The rumor fragments are sent to their appropriate final destinations using the GroupDistribution service: the fragments for rumor ρ are sent to processes in the destination set $\rho.D$. No process outside a rumor's destination set gets both fragments, while all processes in the rumor's destination set (for which the rumor is admissible) deliver the rumor by the specified deadline. (A pseudocode-based description of the algorithm can be found in [12].) We now proceed to present the technical details of the above outline.

1) Preliminaries: For the purposes of the description, fix a deadline $dline$ and focus on rumors with deadlines in the range $[dline/2, dline]$. We execute $\Theta(\log \log n)$ instances of the protocol, each for a specified range of deadlines. When a rumor has a deadline $> \Theta(\log^6 n)$, we truncate the deadline and handle it accordingly, as there is no benefit to deadlines longer than $\Theta(\log^6 n)$.)

We present ConGos as a set of composed distributed services. We will leverage existing distributed protocols as black box services, without delving into the underlying implementation details. We assume that the system consists of the following three services:

- Network: a communication network with ports *send* and *receive* at each process.
- GroupGossip$[\ell]$: a *Continuous Gossip service*, albeit, one that does not guarantee confidentiality. We assume per-round message complexity by $O(n^{1+6\sqrt[3]{dmin}}$ polylog $n)$, with high probability, where $dmin$ is the shortest deadline of any active rumor (see [13]).

We assume there are $\log n$ instantiations of this continuous gossip service, GroupGossip$[\ell]$ for $\ell \in \{1, \ldots, \log n\}$. The instance GroupGossip$[\ell]$ is associate with partition ℓ of the network, which we define shortly. Every message sent by GroupGossip$[\ell]$ is *filtered* before being sent over the network: if a process p_i is a member of some group P' in partition ℓ, then every message sent by GroupGossip$[\ell]$ at process p_i to a process *not* in P' is dropped; every message sent by GroupGossip$[\ell]$ at process p_i to a process in P' is relayed to the Network and sent. From the perspective of the instance GroupGossip$[\ell]$, the processes that cannot be reached due to the filter are effectively failed.

- AllGossip: a non-filtered *Continuous Gossip service*.

ConGos is composed of three further services: ConfidentialGossip, Proxy, and GroupDistribution.

2) ConfidentialGossip Service: Rumors are injected in the ConfidentialGossip service, which acts as the main control unit of the protocol. To ensure confidentiality, each rumor ρ is divided into two fragments ρ_0 and ρ_1. Both fragments maintain certain *metadata*, such as the rumor's destination set, but each fragment on its own provides no information as to the original rumor datum $\rho.z$; together, they allow the original rumor to be reconstructed. There are a variety of simple schemes for accomplishing this: for example, let $\rho_0.z$ be a random binary string, and let $\rho_1.z = (\rho.z$ **xor** $\rho_0.z)$. We have thus reduced the problem of confidentiality to ensuring that no process, except those in the destination set, learn both fragments. All the processes in the system are partitioned into two components, and one rumor fragment is distributed to each half.

It is not sufficient, however, to carry out this splitting-

ConfidentialGossip:

- Do in parallel for each $\ell = 1, \ldots, \log n$:
  1) Split rumor ρ into a pair $\langle \rho_{0,\ell}, \rho_{1,\ell} \rangle$.
  2) If p_i is in group b of partition ℓ, inject $\rho_{b,\ell}$ into GroupGossip$[\ell]$, and inject $\rho_{1-b,\ell}$ into Proxy$[\ell]$. Together, these two services ensure that each rumor fragment is delivered to every non-failed process in the appropriate group of the partition.
  3) For each rumor fragment received from GroupGossip$[\ell]$ or Proxy$[\ell]$, inject the fragment into GroupDistribution$[\ell]$.
  4) Save every fragment received from GroupDistribution$[\ell]$, and reassemble and deliver rumors as fragments become available.
- Whenever a message from AllGossip confirms that, for some partition ℓ, both fragments of a rumor ρ, initiated at p_i, have been sent to every destination in $\rho.D$, confirm that ρ has been delivered.
- Whenever a deadline is about to expire for some rumor ρ initiated at p_i, and there is no confirmation that ρ has been delivered, send ρ directly to every process in $\rho.D$.

Figure 1. Outline of ConfidentialGossip service at p_i.

and-partitioning process only once: the adversary, being adaptive, may kill all the processes in one of the groups in the partition. We thus define, a priori, $\log n$ different partitions. Each partition is based on a specified bit in the binary representation of a process's identifier. Let $p_j[\ell]$ be the ℓ^{th} bit in p_j's binary representation. Then partition ℓ is defined by the two sets $P_{0,\ell} = \{p_j : p_j[\ell] = 0\}$ and $P_{1,\ell} = \{p_j : p_j[\ell] = 1\}$. For each partition, rumor ρ is divided into two fragments $\rho_{0,\ell}$ and $\rho_{1,\ell}$.

The ConfidentialGossip service, running at each process p_i, "spawns" ℓ instances of the other services. The notation ServiceName$[\ell]$ denotes the instance of a service for partition ℓ. (The Network service and the AllGossip service run only one instance for all partitions.) Figure 1 has a high-level outline of the ConfidentialGossip at a process p_i. (Detailed pseudocode can be found in [12].) We now summarize.

Time is divided into blocks of $dline/4$ rounds. A rumor injected during some block B is split into fragments during block B (step 1); the fragments are distributed to their respective groups during block $B+1$ (step 2); the fragments are reassembled in block $B+2$ (step 3 and 4); and the source verifies that its rumor has been delivered by the end of block $B+3$. If it cannot verify that its rumor has been delivered in time, it sends the rumor directly.

A notable aspect is that a process cannot directly distribute both fragments. If a process p_i is in group $P_{0,\ell}$, it cannot directly participate in gossip with group $P_{1,\ell}$; if it did, it might risk learning rumor fragments from the other group. The Proxy service is used to circumvent this problem.

Another notable aspect occurs at the end of the protocol, when a process attempts to confirm that its rumors have been delivered. Each process, as part of the GroupDistribution service, initiates a gossip (via AllGossip) indicating which rumor fragments have been distributed to which processes. Of course, a process cannot divulge the *contents* of the rumor that have been distributed; however, it can safely indicate a unique identifier that was appended by the source, when the rumor was split. In this way, the source can ensure that, for at least one partition, both rumor fragments were successfully delivered. (It would not be sufficient for recipients to send an acknowledgment, as the source does not know which processes have remained alive.)

3) Proxy Service: The proxy service delivers rumor fragments safely across group boundaries. The proxy service for partition ℓ at process p_i repeatedly samples processes from the other group (i.e., the group that p_i does not belong to), requesting that these processes act as *proxies* for p_i in distributing its rumor fragments. The proxies then participate in GroupGossip$[\ell]$ to distribute the rumor fragments, as requested. If they succeed, they send an acknowledgment to p_i. Otherwise, process p_i needs to try again.

The challenge, here, is that the adversary may (adaptively) crash processes as soon as they receive proxy requests. Even worse, at any given time, most members of a group may be failed, requiring p_i to send a very large number of queries to find a non-failed proxy. To avoid this problem, the processes in one group collaborate on finding proxies in the other. During this process, p_i does not share any information on the fragments it is attempting to distribute. The Proxy service for partition ℓ proceeds as in Figure 2 (detailed pseudocode can be found in [12]).

Proxy$[\ell]$:

- Time is divided into blocks of length $dline/4$.
- At the beginning of a block, collect all the fragments that have been injected since the last block began, and set status to active.
- Each block is divided into *iterations* of $\sqrt{dline} + 2$ rounds. In each iteration, we maintain a set $collaborators$ of the active processes in the same group as p_i. We also keep track of $failed\text{-}proxies$, i.e., those that we have already learned (in previous iterations) have failed in this block. For each iteration:
  - Round 1: send every rumor fragment associated with the other group to $n^{1+48/\sqrt{dline}} \log n / |collaborators|$ processes chosen uniformly at random from the other group, excluding processes in $failed\text{-}proxies$. (Notice that as long as the set $collaborators$ is a good estimate of the set of collaborators, this ensures a good bound on the message complexity of this step.) Every process that receives a request to be a proxy for the other group caches the received rumor fragments.
  - Rounds $2, \ldots, \sqrt{dline} + 1$: initiate a GroupGossip$[\ell]$ in which processes in the same group as p_i share the set of $failed\text{-}proxies$, as well as choose a new set of $collaborators$, i.e., members of the group that are still active. Processes also share all the rumor fragments received from the other group. (The deadline for rumors in GroupGossip$[\ell]$ here is $\sqrt{dline}$.)
  - Round $\sqrt{dline} + 2$: Any process that was asked to be a proxy for the other group sends an acknowledgment that proxying was successful. Any process that sent a request, and does not receive an acknowledgment, adds the non-acknowledging processes to the set of $failed\text{-}proxies$.

Figure 2. Outline of Proxy$[\ell]$ at p_i.

4) GroupDistribution Service: The GroupDistribution$[\ell]$ service forwards rumor fragments to their final destination. To this point, for a partition ℓ, the rumor fragment $\rho_{0,\ell}$ has been distributed to processes in $P_{0,\ell}$, and the rumor fragment $\rho_{1,\ell}$ has been distributed to processes in $P_{1,\ell}$. Now, group

GroupDistribution[ℓ]:

- Time is divided into blocks of length $dline/4$.
- At the beginning of the second round of a block, collect all the fragments that have been injected since the first round of the block, and set status to active. (The first round of the block is spent waiting for rumor fragments from the previous block.)
- Each block is divided into *iterations* of $\sqrt{dline}+2$ rounds. In each iteration, we maintain a set $collaborators$ of the active processes in the same group as p_i. We also keep track of a set $hitSet$ of processes that have been sent a message in this block. Each process in this set was sent all the rumor fragments for this block. For each iteration:
  - Round 1: wait for rumor fragments to be injected.
  - Round 2: send every "appropriate" rumor fragment to $n^{1+48/\sqrt{dline}}\log n/|collaborators|$ processes chosen uniformly at random from the other group, excluding processes in $hitSet$. By appropriate we mean that if p_j is a process chosen randomly by p_i, then p_i sends to p_j only the rumor fragments in which p_j is in the destination set. (Recall that each partial rumor contains the target destination set as part of the metadata.) Every process that receives rumor fragments can now reconstruct the rumor and return it to its user (via the ConfidentialGossip service).
  - Rounds $3,\ldots,\sqrt{dline}+2$ rounds: initiate an instance of GroupGossip[ℓ] (with deadline $\sqrt{dline}$) in which processes in the same group as p_i share their $hitSet$s, as well as count how many members of the group are still active.
- In the last round of the block, initiate an instance of AllGossip (with deadline $dline/4-1$). Each process p_i gossips the information in its $hitSet$, but without including the rumor fragments themselves. That is, if the $hitSet$ of process p_i indicates that some rumor fragment $\rho_{0,\ell}$ was sent to some process p_j, and if $\rho_{0,\ell}$ has identifier r, then p_i gossips that the fragment 0 for partition ℓ of the rumor associated with identifier r was sent to p_j. This provides sufficient information for the source to determine whether the rumor was delivered, without revealing the contents of the rumor. (See the description of the ConfidentialGossip service, above, for how this information is used.)

Figure 3. Outline of GroupDistribution[ℓ] at p_i.

$P_{0,\ell}$ collaborates to send the fragment $\rho_{0,\ell}$ to $\rho_{0,\ell}.D$, while $P_{1,\ell}$ does the same for fragment $\rho_{1,\ell}$. Of course there may be many different fragments active in each group, each with a different destination set.

The basic operation of the GroupDistibution is similar to that of the Proxy Service. Each process chooses a set of recipients at random, and sends each of them a message carefully composed of only appropriate rumor fragments. The processes then gossip within their group (via GroupGossip[ℓ]), sharing information on which processes have already been notified, and which remain to be notified. At the same time, processes calculate the number of processes active in a group, which allows them to determine the appropriate number of messages to send. We give an outline of the GroupDistribution service for partition ℓ in Figure 3 (detailed pseudocode can be found in [12]).

B. Algorithm Analysis

We begin with the correctness of CONGOS. A formal proof is deferred to [12].

Theorem 2: Algorithm CONGOS solves the Confidential Continuous Gossip problem under adversary $CRRI$.

We now state important properties needed for analyzing the message complexity of the algorithm.

Lemma 3: Given rumor ρ, injected at time t: if there are at least 2 processes that remain alive throughout the interval $[t, t+\rho.d]$, then for some partition ℓ, there is at least one process in $P_{0,\ell}$ and one process in $P_{1,\ell}$ that remain alive throughout the interval $[t, t+\rho.d]$.

Proof: Let p_i and p_j be the two processes hypothesized to remain alive throughout the specified interval. Since identifiers are unique, let ℓ be some bit where the identifier of p_i and p_j differ. The claim follows for partition ℓ. ■

Lemma 4: In each block, the Proxy[ℓ] service and the GroupDistribution[ℓ] service execute at least $\sqrt{dline}/8$ iterations, if $dline > 4$.

Proof: Each block is of length $dline/4$, and each interval is of length at most $\sqrt{dline}+2 \le 2\sqrt{dline}$. ■

Lemma 5: In each round, the Proxy[ℓ] service and the GroupDistribution[ℓ] service send at most $O(n^{1+48/\sqrt{dline}}\log n)$ messages.

Proof: (sketch) In Proxy[ℓ] and GroupDistribution[ℓ], each process sends $\frac{n^{1+48/\sqrt{dline}}\log n}{|collaborators|}$ messages in, respectively, the first and second round of an iteration. The bound on $collaborators$ implies the desired result. In Proxy[ℓ], each process that received a proxy request sends a response at the end of an iteration. Each response is the result of an earlier request in the first round of the iteration, and we have already bounded the message complexity of the first round of an iteration, leading here too to a bound of $O(n^{1+48/\sqrt{dline}}\log n)$. ■

The next lemma would be straightforward if the adversary were oblivious. However, since the adversary is adaptive and can schedule according to the random choices, we have to show that the requisite properties hold despite all possible adversarial choices.

Lemma 6: Given rumor ρ, injected at time t at process p_i: if at least one process in $P_{0,\ell}$ and one process in $P_{1,\ell}$ remain alive throughout the interval $[t, t+\rho.d]$, then every process in $P_{0,\ell}$ that remains alive throughout the interval $[t, t+\rho.d]$ receives $\rho_{0,\ell}$, and every process in $P_{1,\ell}$ that remains alive throughout the interval $[t, t+\rho.d]$ receives $\rho_{1,\ell}$ by time $t + 2dline/4 - (t \mod dline)$, with high probability.

Proof: (sketch) Fix ℓ to be the partition identified in Lemma 3. Assume w.l.o.g. that $p_i \in P_{0,\ell}$. (The alternate case is symmetric.) Since p_i injects rumor fragment $\rho_{0,\ell}$ into the GroupGossip[ℓ] service with deadline $\sqrt{dline}$, it is guaranteed to reach every process in $P_{0,\ell}$ that remains alive throughout the interval. It remains to show that each process in $P_{0,\ell}$ succeeds in finding a proxy in $P_{1,\ell}$, while executing Proxy[ℓ] during the first complete block after rumor ρ is injected beginning at time $t + dline/4 - (t \mod dline)$. Let $Z = n^{48/\sqrt{dline}}$. The crucial argument is that in every pair of iterations, one of the three following events occurs: (i) At least a $(1-1/Z)$ fraction of processes in $P_{0,\ell}$ that were alive at the beginning of the first iteration

fail by the end of the second iteration; (ii) At least a $(1-1/Z)$ fraction of processes in $P_{1,\ell}$ that were alive at the beginning of the first iteration fail by the end of the second iteration; (iii) In the second iteration, at least a $(1-1/Z)$ fraction of processes in $P_{0,\ell}$ succeed in finding a proxy. From this claim and by the lemma assumptions, we can conclude that by the end of $3\log_Z(n)$ pairs of iterations, every process in $P_{0,\ell}$ has succeeded in finding a proxy. Note that since $\log_Z(n) = \sqrt{dline}/48$, this process finishes within $6\log_Z(n) \leq \sqrt{dline}/8$ iterations, as required. Once a process has succeeded in finding a proxy, it follows from the guarantees of GroupGossip that its rumor fragment is distributed to every non-failed process in the other group. ■

Lemma 7: Given rumor ρ, injected at time t at process p_i: if at least one process in $P_{0,\ell}$ and one process in $P_{1,\ell}$ remain alive throughout the interval $[t, t+\rho.d]$, then every process $p_j \in \rho.D$ receives fragment $\rho_{0,\ell}$ and fragment $\rho_{1,\ell}$ by time $t + 3dline/4 - (t \mod dline)$, with high probability.

Proof: (sketch) Due to Lemma 6, it remains to show that during the following block of rounds, for every process $p_j \in \rho.D$, at least one process from each group sends its rumor fragment to p_j. W.l.o.g. we focus on group $P_{0,\ell}$. Let $Z = n^{48/\sqrt{dline}}$. The crucial observation is that in each pair of iterations of the GroupDistribution$[\ell]$ service, one of the following two events occurs: (i) At least a $(1-1/Z)$ fraction of processes in $P_{0,\ell}$ that were alive at the beginning of the first iteration fail by the end of the second iteration; (ii) For every process p_k active throughout both iterations, the set of processes $[n] \setminus hitProcs_k$ decreases by a factor of Z by the end of the second iteration. (The set $hitProcs_k = \{p_q \in [n] : \langle p_q, \cdot\rangle \in hitSet_k\}$.) From this claim, we conclude that within $2\log_Z n$ pairs of iterations, either every process in $P_{0,\ell}$ fails, or every process has been added to $hitProcs$. Hence we conclude that by the end of $4\log_Z n \leq \sqrt{dline}/8$ iterations, every process has been sent all of its rumor fragments. ■

Lemma 8: Given rumor ρ, injected at time t at process p_i: if p_i does not fail by time $t+\rho.d$, then by round $t+\rho.d-1$, process p_i receives confirmation that rumor ρ was delivered, with high probability.

Proof: By Lemma 3, we know that if rumor ρ has even one admissible destination $\neq p_i$, then there is some partition ℓ where there is at least one process in $P_{0,\ell}$ and one process in $P_{1,\ell}$ that does not fail in $[t, t+\rho.d]$. By Lemma 7, we know that by time $t+3dline/4-(t \mod dline)$, with high probability, rumor ρ has been delivered to every destination in $\rho.D$ by the GroupDistribution$[\ell]$ service. Moreover, since at least one process p_0 in $P_{0,\ell}$ and one process p_1 in $P_{1,\ell}$ does not fail during $[t, t+\rho.d]$, we conclude that p_0 and p_1 complete the block in which the rumor fragments for ρ are delivered to their destinations. At the end of the last round of the block, processes p_0 and p_1 inject sanitized versions of the $hitSets$ as rumors into the AllGossip service with deadline $dline/4-1$, thus ensuring that process p_i receives this information no later than round $t+\rho.d-1$. Process p_i then marks rumor ρ confirmed. ■

Theorem 9: The per-round message complexity of algorithm ConGos is:

$$O\left((n^{1+48/\sqrt{dline}} + n^{1+6/\sqrt[6]{dline}})\,\text{polylog } n\right) .$$

Proof: From Lemma 8 we have that for a given rumor ρ injected at process p_i, with high probability the rumor is confirmed prior to the deadline expiring. Since each process is injected at most one rumor per round (hence there can be $O(n\,\text{polylog } n)$ active rumors in the system at any given time), with high probability, no source process sends any messages directly to the destinations. From Lemma 5, the per-round message complexity for each instance of Proxy$[\ell]$ and GroupDistribution$[\ell]$ is $O(n^{1+48/\sqrt{dline}}\log n)$, leading to a per-round message complexity of $O(n^{1+48/\sqrt{dline}}\log^2 n)$. Each instance of continuous gossip, invoked with rumors at least $\sqrt{dline}$, has message complexity $O(n^{1+6/\sqrt[6]{dline}}\,\text{polylog } n)$. There are $\log n + 1$ such instances of continuous gossip. ■

V. Gossiping in the Presence of Collusion

In this section we extend our investigation of the confidential gossip problem by additionally assuming that processes outside of a rumor's destination set may collude in an attempt to learn the rumor. We assume throughout that the processes are *honest-but-curious*, i.e., they may collude to learn information, but they will continue to follow the protocol.

More formally, given a rumor ρ injected at a process p_i, we denote by C_ρ the collusion set of ρ. C_ρ may contain any process $q \notin \rho.D \cup \{p_i\}$. Adversary $CRRI$ selects the colluding processes in an adaptive way during the execution. $CRRI(\tau)$ denotes the adversarial patterns of $CRRI$ for which $|C_\rho| \leq \tau$ for all rumors ρ system. Finally, an algorithm is *τ-collusion-tolerant* if it solves confidential gossip under adversary $CRRI(\tau)$.

A. Lower Bound

We show a lower bound for any algorithm generlizing ConGos. A gossip algorithm is *partition-based* if it allows only two operations tampering with the content of the rumors: splitting, which splits a rumor into fragments, and merging, which merges fragments of the *same* rumor[1]. Otherwise, the protocol must treat the rumor (and its fragments) as nonmalleable tokens. We show that in this case, the effect of collusion is quite significant, even against an *oblivious adversary*.

Theorem 10: For all $\varepsilon > 0$, every randomized, τ-collusion-tolerant, partition-based algorithm solving confidential gossip has a maximum per-round message-complexity of at least $\Omega(\min\{n\tau, n^{(3/2)-\varepsilon}\}/dmax)$, with

[1]Notice that this does not allow other algebraic manipulation of the rumor, as in "network coding" techniques.

probability 1, against an oblivious adversary, where $dmax$ is the longest deadline of the injected rumors.

Proof: (sketch) We assume the same initial setting of parameters and rumors, including their destination sets and deadlines, as in the proof of Theorem 1. We assume $n > 8$, and let c be a constant and x be a parameter, to be specified, depending on ε. Each process is initially injected with one rumor, all of which have deadline $dmax$. The destination set of each rumor is as in the proof of Theorem 1.

Consider an execution. Let a *rumor interval* be a set of rumor fragments such that any set of fragments sufficient to reconstruct the rumor includes one fragment from the rumor interval. Two cases are possible:

Case 1: More than half of the rumors satisfy the following property: there is a rumor interval such that none of its contained fragments is ever transmitted to a process outside the destination set. It follows that each destination process receives some rumor fragment in this rumor interval directly from the rumor's source or relayed entirely through the processes in the destination set. Therefore, the messages carrying rumor fragments in this rumor interval altogether suffer from the same constraints as it would an original rumor propagated within its destination sets only, and by Theorem 1, the number of such messages is proportional to the size of the destination set. Since there are more than $n/2$ such rumors, we get the lower bound $\Omega(n^{(3/2)-\varepsilon})$ on the total number of such messages in the considered period of length $dmax$. Hence, the per round message complexity in this case is $\Omega(n^{(3/2)-\varepsilon}/dmax)$.

Case 2: At least half of the rumors satisfy the following property each: fragments of the rumor transmitted outside the destination set cover the whole original rumor.

In this case for each such rumor there are at least $\tau + 1$ processes outside its destination set that receive a fragment of the rumor directly from some processes in the destination set or the rumor's source (the process that the rumor was injected at); otherwise at most τ such outside processes could collude and get fragments covering the whole rumor, thus violating the definition of confidentiality (which must hold for every execution). We call such at least $\tau + 1$ fragments *border fragments*. Therefore there are at least $\tau+1$ point-to-point messages sent from some processes in the destination set or the rumor's source to the considered at least $\tau + 1$ outside processes. Call these messages *border messages*. It follows that there are at least $(\tau + 1)n/2$ copies of border fragments sent via border messages. Recall the property of the considered configuration of destination sets as proved in Theorem 1: each process is in at most c destination sets, where c is a constant. It follows that a process sends at most c border fragments per border message, which gives at least $\frac{(\tau+1)n/2}{c} = \Omega(n\tau)$ border messages. Hence the per-round message complexity in this case is $\Omega(n\tau/dmax)$. ■

B. Algorithm

We modify algorithm ConGos in the following way. Instead of $\log n$ partitions, we utilize $c\tau \log n$ different partitions of the processes, for an appropriate choice of constant c. Each partition contains $\tau + 1$ groups, instead of 2 groups. Rumors are now divided into $\tau + 1$ fragments.

1) Partitions: The set of $c\tau \log n$ partitions needs to satisfy the following properties, for appropriate choice of constants c and c': (i) In each partition, each group contains at least one process. (ii) For every set S of at least $2c'\tau \log n$ processes, there exists a partition such that every group in the partition contains at least one process in S. The first property ensures well-formedness, i.e., that the partition is a proper division of the processes into non-empty groups. The second property ensures good performance: as long as there are $\Omega(\tau \log n)$ processes alive, then one partition has live processes in every group and hence can be used to distribute the rumor fragments. By randomly selecting the groups for each partition, observe that with positive probability both properties are satisfied. Thus, by the probabilistic method, we can show (see [12]) that there exists a good set of partitions that meets the requirements:

Lemma 11: If $\tau < n/\log^2 n$, then there is a set of $c\tau \log n$ partitions satisfying the above conditions, for some constants $c, c' > 0$.

We leave the polynomial time construction of partitions satisfying the required conditions as future work.

2) Overview of collusion-tolerant ConGos*:* The modified version of algorithm ConGos operates much like ConGos, just relying on more groups and more partitions. Consider a newly injected rumor ρ at a process p_i. For each partition ℓ, the procedure ConfidentialGossip divides the rumor into the fragments $\rho_{0,\ell}, \ldots, \rho_{\tau,\ell}$ such that all fragments (from the same partition) are needed in order for ρ to be re-assembled. (A way to do this is as follows: Let $\rho_{0,\ell}, \ldots, \rho_{\tau-1,\ell}$ be different random binary strings and set $\rho_{\tau,\ell} = (\rho$ **xor** $\rho_{0,\ell}$ **xor** $\ldots$ **xor** $\rho_{\tau-1,\ell})$. Then ρ can be computed when all $\tau + 1$ fragments are received. Note that this scheme makes the algorithm partioned-based.)

Say that in partition ℓ, process p_i belongs in group x. Then it injects fragment $\rho_{x,\ell}$ in GroupGossip$[\ell]$ and all other fragments into Proxy$[\ell]$. Via procedure GroupGossip$[\ell]$, the fragment $\rho_{x,\ell}$ is gossiped in the members of group x and via Proxy$[\ell]$ each other fragment is gossiped into every other corresponding group (such that every other group learns a different fragment of the rumor). Then procedure GroupDistribution$[\ell]$ is called so that the processes in each group collaborate in sending their corresponding fragment of the rumor only to the processes of the rumor's destination set. These processes receive all fragments and hence can reassemble the rumor. Lemma 11 assures the existence of at least one partition ℓ in which all admissible rumors are received by the live processes of the rumor's destination set.

Detailed outlines of the procedures can be found in [12]. We now give the main result of this section.

Theorem 12: The modified version of algorithm CONGOS solves the confidential gossip problem under adversary $CRRI(\tau)$ with per-round message complexity of

$$O\left((n^{1+48/\sqrt{dline}} + n^{1+6/\sqrt[6]{dline}})\tau^2 \operatorname{polylog} n\right) .$$

Proof: (sketch) The correctness follows by similar arguments as for algorithm CONGOS, since the partitions used for the modified algorithm (with the properties proved in Lemma 11) satisfy the same conditions explored in the analysis as the partitions used in the original algorithm CONGOS.

The message complexity increases by a factor τ^2, compared to the complexity of the original algorithm CONGOS, because of the following two observations.

First, for all rounds but the last one of the considered deadline period the amount of inter-group communication (that is, the last round in procedure GroupDistribution) increases by a factor of at most $\tau + 1$, as the number of groups is now $\tau + 1$ instead of 2. The communication is increased by another factor $\Theta(\tau)$ due to the fact that now there are $\Theta(\tau \log n)$ partitions instead of $\log n$. Thus each message sent in the original algorithm CONGOS is multiplied by at most $\Theta(\tau^2)$ different copies.

Second, in the last round of ConfidentialGossip (shooting directly to processes in the destination set) there is no communication if the number of processes alive in the whole period is at least $2c'\tau \log n$ (that is, all rumors have been confirmed to be delivered); this follows by the second property of the set of partitions and by the same argument as in the analysis of the original algorithm CONGOS. In the case where the number of alive processes is smaller than $2c'\tau \log n$, the number of point-to-point messages sent is bounded by $2c'\tau n \log n$ due to the fact that only these processes that remain alive throughout may send messages in the last round (each sending at most n messages). This completes the proof. ∎

Observe from Theorem 12 that when $dline = \Theta(\log^6 n)$ the per-round message complexity is $O(n\tau^2 \operatorname{polylog} n)$. When contrasted with Theorem 10 it follows that for $\tau < n^{1/4}$, the per-round message complexity is within a factor of $\tau \operatorname{polylog} n$ of the lower bound. (For $\tau = O(\operatorname{polylog} n)$ the algorithm is optimal within log factors.)

VI. DISCUSSION

In this paper we have considered the problem of *confidential* gossip, where each rumor is learned only by processes in the rumor's specified destination set. Assuming an adaptive and omniscient adversary that dynamically and continuously injects rumors into the system and causes process crashes and restarts, we have designed an efficient (w.r.t. per-round message complexity) algorithm which we call algorithm CONGOS. As an alternative to cryptographic schemes, which can be expensive in such a dynamic environment, the algorithm deploys a simple rumor splitting technique that enables an efficient "all-process" collaboration while guaranteeing confidentiality. For this purpose, the algorithm combines, in a non-trivial way, a black-box efficient non-confidential continuous gossip service with other auxiliary services (namely, Filter, Proxy, GroupDistribution). While we have focused on continuous gossip, we believe that the same techniques apply to other gossip variants (e.g., single-instance gossip, etc.).

We have also discussed the problem of collusion, and shown how to tolerate a moderate amount of collusion at a limited cost. An interesting open question is whether we can tolerate higher levels of collusion if the adversary is oblivious, or if we allow some small probabilistic violation of confidentiality.

In addition, as currently presented, the algorithm guarantees the confidentiality of rumors, but various other metadata is released. For example, processes learn of the existence of rumors, roughly how many rumors are active, the source of each rumor, a sequence number of each rumor, and the set of destinations for each rumor. Some of this information can be readily hidden. For example, the sequence number can be replaced with a pseudorandom identifier. Other information appears more difficult to hide, for example, the proxies learn precisely who is requesting that they act as a proxy, and this seems, to some extent, unavoidable.

The destination set associated with each rumor can be hidden, without increasing the overall message complexity, but at the cost of increasing the message size (significantly). When a rumor ρ is injected at process p_i, the source creates n new rumors, each with a single process in its destination set. For every process in $\rho.D$, the new rumor contains a copy of the injected rumor's content. For the remaining new rumors, the contents of the new rumor are chosen at random. The source then proceeds to distribute this entire collection of rumors. Only the processes in the destination set can determine whether a rumor contains real content or simply a random string, and hence processes cannot determine the real destination set.

Similarly, the very existence of rumors can be hidden by the continual injection of fake content-free rumors, at the cost of wasted messages. In this way, a process cannot determine how many real rumors are currently active.

Finally, an interesting open question is whether we can tolerate truly malicious processes, i.e., those that do not follow the protocol. In fact, we believe that the approach for tolerating collusion may be extended to deal with malicious processes, if the adversary is oblivious. In that case, we can tolerate some groups misbehaving and failing to deliver their message fragments.

REFERENCES

[1] S. Baehni, P.T. Eugster, and R. Guerraoui. Data-Aware Multicast. *In DSN 2004,* pages 233–242.

[2] A.J. Ballardie. *A New Approach to Multicast Communication in a Datagram Network*, Ph.D. Thesis, University College London, 1995.

[3] A. Beimel, K. Nissim, and E. Omri. Distributed Private Data Analysis. In *CRYPTO 2008*, pages 451–468.

[4] J. Brickell and V. Shmatikov. Privacy-preserving Graph Algorithms in the Semi-honest Model. In *ASIACRYPT 2005*, pages 236–252.

[5] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast Security: A Taxonomy and Some Efficient Constructions. *In INFOCOM 1999,* pages 708–716.

[6] B.S. Chlebus and D.R. Kowalski. Time and Communication Efficient Consensus for Crash Failures. *In DISC 2006,* pages 314–328.

[7] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg. SpiderCast: A Scalable Interest-Aware Overlay for Topic-Based Pub/Sub Communication. *In DEBS 2007,* pages 14–25.

[8] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg. Constructing Scalable Overlays for Pub/Sub with Many Topics. *In PODC 2007,* pages 109–118.

[9] G. Delposte-Gallet, H. Fauconnier, R. Guerraoui, and E. Ruppert. Secretive Birds: Privacy in Population Protocols. In *OPODIS 2007*, pages 329–342.

[10] B. Doerr, T. Friedrich, and T. Sauerwald. Quasirandom Rumor Spreading: Expanders, Push vs Pull, and Robustness. *In ICALP 2009,* pages 366–377.

[11] A. Fiat and M. Naor. Broadcast Encryption. *In CRYPTO 1993,* pages 480–491.

[12] Ch. Georgiou, S. Gilbert, and D.R. Kowalski. Confidential Gossip. *Technical Report*, available at http://www.cs.ucy.ac.cy/~chryssis/congosTR.pdf.

[13] Ch. Georgiou, S. Gilbert, and D.R. Kowalski. Meeting the Deadline: On the Complexity of Fault-Tolerant Continuous Gossip. *In PODC 2010,* pages 247–256.

[14] O. Goldreich. *Foundations of Cryptography: Volume II (Basic Applications)*. Cambridge University Press, 2004.

[15] I. Gupta, A.M. Kermarrec, and A.J. Ganesh. Efficient Epidemic-style Protocols for Reliable and Scalable Multicast. *In SRDS 2002,* pages 180–189.

[16] H. D. Johansen, A. Allavena, and R. van Renesse. Fireflies: Scalable Support for Intrusion-tolerant Network Overlays. *In EuroSys 2006,* pages 3–13.

[17] J. Hromkovic, R. Klasing, A. Pelc, P. Ruzika, and W. Unger. *Dissemination of Information in Communication Networks: Broadcasting, Gossiping, Leader Election, and Fault-Tolerance,* Springer-Verlag, 2005.

[18] R. Karp, C. Schindelhauer, S. Shenker, B. Vocking. Randomized Rumor Spreading. *In FOCS 2000,* pages 565–574.

[19] D. Kempe, J. Kleinberg, and A. Demers. Spatial Gossip and Resource Location Protocols. *Journal of the ACM,* 51:943–967, 2004.

[20] A. Kermarrec, L. Massoulie, A. Ganesh. Probabilistic Reliable Dissemination in Large-scale Systems. *IEEE Transactions on Parallel and Distributed Systems,* 14(3):248–258, 2003.

[21] L. Kissner and D. Song. Privacy-preserving Set Operations. In *CRYPTO 2005*, pages 241–257.

[22] D. R. Kowalski and M. Strojnowski. On the Communication Surplus Incurred by Faulty Processors. *In DISC 2007,* pages 328–342.

[23] Y. Lindell and B. Pinkas. Privacy Preserving Data Mining. *J. Cryptology*, 15(3):177–206, 2002.

[24] D. Malkhi, Y. Mansour, and M.K. Reiter. Diffusion Without False Rumors: On Propagating Updates in a Byzantine Environment. *Theoretical Computer Science,* 299:289–306, 2003.

[25] D. Micciancio and S. Panjwani. Corrupting One Vs. Corrupting Many: The Case of Broadcast and Multicast Encryption. *In ICALP 2006,* pages 70–82.

[26] Y.M. Minsky and F.B. Schneider. Tolerating Malicious Gossip. *Distributed Computing,* 16:49–68, 2003.

[27] S. Mittra. Iolus: A Framework for Scalable Secure Multicasting. *SIGCOMM Comput. Commun. Rev.,* 27(4):277–288, 1997.

[28] Multicast Security. http://datatracker.ietf.org/wg/msec/

[29] M. Onus and A.W. Richa. Minimum Maximum Degree Pub/Sub Overlay Network Design. *In INFOCOM 2009,* pages 882–890.

[30] J. Pang and C. Zhang. How to Work with Honest but Curious Judges? *In Proc. 7th International Workshop on Security Issues in Concurrency*, pages 31–45, 2009.

[31] S. Panjwani. Tackling Adaptive Corruptions in Multicast Encryption Protocols. *In TCC 2007*, pages 21–40.

[32] A. Pelc. Fault-tolerant Broadcasting and Gossiping in Communication Networks. *Networks,* 28: 143–156, 1996.

[33] A. Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, 1979.

[34] A.T. Sherman and D.A. McGrew. Key Establishment in Large Dynamic Groups Using One-Way Function Trees. *IEEE Transactions on Software Engineering,* 29(5):444–458, 2003.

[35] D.R. Stinson. *Cryptography: Theory and Practice,* CRC Press, 3rd edition, 2005.

[36] C. K. Wong, M. Gouda, and S. Lam. Secure Group Communications Using Key Graphs. *IEEE/ACM Transactions on Networking,* 8(1):16–30, 2000.

[37] A.C. Yao. Protocols for Secure Computations. *In FOCS 1982*, pages 160–164.