Xor-Trees for Efficient Anonymous Multicast and Reception

SHLOMI DOLEV
Ben-Gurion University of the Negev
and
RAFAIL OSTROVSKY
Telcordia Technologies

We examine the problem of efficient anonymous multicast and reception in general communication networks. We present algorithms that achieve anonymous communication, are protected against traffic analysis, and require $O(1)$ amortized communication complexity on each link and low computational complexity. The algorithms support sender anonymity, receiver(s) anonymity, or sender-receiver anonymity.

Categories and Subject Descriptors: C.2.0 [**Computer-Communication Networks**]: General—*Security and protection* (e.g., firewalls); F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Routing and layout*

General Terms: Algorithms, Security

Additional Key Words and Phrases: Anonymous communication, anonymous multicast

1. INTRODUCTION

One of the primary objectives of an adversary is to locate and to destroy command-and-control centers—that is, sites that send commands and data to various stations and agents. Hence, one of the crucial ingredients in almost any network with command centers is to conceal from (and confuse) the adversary as to which stations issue the commands. This paper shows how to use standard off-the-shelf cyptographic tools in a novel way in order

An extended abstract of this paper appears in the *Proceedings of the 17th Annual IACR Crypto Conference, CRYPTO 1997*.
This work was done while Shlomi Dolev was visiting Bellcore with the support of DIMACS, and was also supported in part by the Israeli Ministry of Science and Arts grant 6756195.
Authors' addresses: S. Dolev, Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, 84105, Israel; email: dolev@cs.bgu.ac.il; R. Ostrovsky, Telcordia Technologies, Morristown, NJ 07960–6438; email: rafail@research.telcordia.com.
Permission to make digital/hard copy of part or all of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.
© 2000 ACM 1094-9224/00/0500–0063 $5.00

to conceal the command-and-control centers, while still assuring easy communication between the centers and the recipients.

Specifically, we show efficient solutions that hide the identities of the sender and the receiver (or both) of the message/directive in a variety of anonymity requirements (e.g., the receiver knows the identity of the sender, or the receiver does not know the identity of the sender, etc.). The proposed solutions are efficient in terms of communication overhead (i.e., how much additional information must be transmitted in order to confuse the adversary), namely $O(1)$ bits per link. The solutions are also efficient in terms of computation (i.e., how much computation must be performed for concealment), namely, $O(k)$ pseudorandom sequences are produced to cope with *k-listening internal dynamic adversary*. A k-listening internal dynamic adversary can monitor all the communication lines between sites and *also* monitor (the internal contents of) up to $k < \lfloor n/2 - 1 \rfloor$ sites of the network. We note that a preprocessing seeds transmission procedure is executed before the actual transmission starts. The communication overhead of this preprocessing procedure is amortized in a long sequence transmission. The preprocessing stage requires $O(g \cdot (kn)^2)$ bits transmission per a link, where g is a security parameter that is the seed size of the pseudorandom generator and n is the number of sites in the network.

1.1 The Problem

Modern cryptographic techniques are extremely good at hiding all the *contents* of the data by encrypting the messages. However, hiding the contents of a message does not hide the fact that *some* message was sent from or received by a particular site; in other words, it does not protect against traffic analysis. Thus, if some location (or network node) is sending and/or receiving a lot of messages and if an adversary can monitor this fact, then even if an adversary does not understand what the messages are, just the fact that there are a lot of outgoing (or incoming) messages reveals that this site (or a network node) is sufficiently active to make it a likely target. The objective of this paper is to address this problem—that is, how to hide, in an efficient manner, the site (i.e., command-and-control center) that transmits (or receives) a lot of data to (or from, respectively) other sites in the network. This problem has been addressed previously in the literature [Chaum 1981; 1988; Rackoff and Simon 1993]. We show an amortized solution for a general graph which, after a fixed preprocessing stage, can transmit an arbitrary polynomial-size message in an anonymous fashion using only $O(1)$ bits over each link (of a spanning tree) for every data bit transmission across a link.

1.2 General Setting and Threat Model

We consider a network of processors/stations where each processor/station has a list of other stations with which it can communicate (we do not restrict the means of communication here, i.e., the means could be computer

networks, radio/satellite connections, etc.) Moreover, we do not restrict the topology of the network—our general methodology will work for an arbitrary network topology. One (or several) of the network nodes is a command-and-control center that wishes to send commands (i.e., messages) to other nodes in the network. The question we address in this paper is: How can we hide the site that is broadcasting (or multicasting) data to (a subset of) other processors in the network? Before we explore this question further, we must specify what kind of attack we are defending against.

A simple attack to defend against is that of a restricted adversary (called an outside adversary) who is allowed to monitor communication channels only, but is *not* allowed to infiltrate/monitor the internal contents of any processor in the network. (Note that such a weak attack is very easy to defend against: all processors simply transmit either noise or encrypted messages on each communication channel. If noise is indistinguishable from encrypted traffic, the pattern of communication is completely hidden.) Of course, a more realistic adversary (and the one that we consider in this paper) is the (internal) adversary that can monitor all the communication between stations and *in addition* is also trying to infiltrate the internal nodes of the network.

That is, we consider the adversary that may mount a more sophisticated attack, one in which he or she manages to compromise the security of one or several *internal* nodes of the network, whereby the adversary is now not only able to monitor the external traffic pattern, but is also able to examine every message and all the data that passes through (or is stored at) the infiltrated node. Thus, we define an *internal k-listening adversary* as one that can monitor all the communication lines between sites and *also* manages to monitor (the internal contents of) up to k sites of the network. (This, and similar definitions, were considered before in the literature; see, for example, Rackoff and Simon [1993] and Canetti et al. [1997], and references therein.) In this paper, however, we restrict out attention to a listening adversary only, one that monitors traffic but does not try to sabotage it, similar to that of Franklin et al. [1993] and Kushilevitz et al. [1994], but with different objectives.

1.3 Comparison with Previous Work

One of the first (if not the first) attempts to consider the problem of hiding the communication pattern in the network is that of Chaum [1981], in which the concept of a *mix* was introduced. A single processor in the network, called a mix, serves as a relay. A processor P that wants to send a message m to a processor Q encrypts m using Q's public key to obtain m'. Then P encrypts the pair (m', q) using the public key of the mix. The double encrypted message is sent to the mix. The mix decrypts the message (to get the pair (m', q)) and forwards m' to q. Further work appears in Pfitzmann [1985]; Pfitzmann et al. [1991]; and Syverson et al. [1997].

The single mix processor is not secure when this *single* processor is cooperating with the (outside) adversary. If the processor that serves as a

mix is compromised, it can inform the adversary as to where the messages are forwarded. Hence, as Chaum points out, a sequence of "mixes" must be employed at the price of additional communication and computation. Moreover, the single mix scheme operates under some statistical assumptions on the pattern of communication. In case a single message is sent to the mix, an adversary that monitors the communication channels can observe the sender and the receiver of the particular message.

An extension of the mix scheme is presented by Rackoff and Simon [1993] who embed an n-element sorting network of polynomial depth in $\log(n)$ that mixes incoming messages and requires only polynomially many (in $\log(n)$) synchronous steps. In each such step every message is sent from one site of the network to another site of the network. Thus, the message delay may be proportional to $\log(n)$ times the diameter of the network. The statistical assumptions on the patterns of communication are somewhat relaxed in Rackoff and Simon [1993] by the introduction of dummy communication: every processor sends a message simultaneously. However, the number of (real and dummy) messages arriving at each destination is available to the traffic analyzer. This has two drawbacks. The first is the possibility of revealing the identity of the receiver when all (real) messages are sent to a particular node, the second is the $O(n)$ messages that a particular node (the receiver) has to process at a time. Rackoff and Simon [1993] also presented a scheme that copes with passive internal adversaries by using randomly chosen committees and multiparty computation (e.g., Goldreich et al. [1987], Ben-Or et al. [1988], Chaum et al. [1988], and Canetti et al. [1996; 1997].)

More generally, secure multiparty computation can be used to hide the communication pattern in the network (see, for example, Goldreich et al. [1987], Chaum [1988], Waidner and Pfitzmann [1989], Ben-Or et al. [1988], Chaum et al. [1988], and Canetti et al. [1996; 1997]) via secure function valuation. However, anonymous communication is a very restricted form of hiding participants' input, and hence may benefit from less sophisticated and more efficient algorithms. Moreover, the solutions above are complexity-theoretically secure, while there are approaches that may lead to information-theoretically secure solutions, such as the one described in Chaum [1988]. Chaum [1988] suggests using the dc-net approach in order to achieve anonymous communication. Our approach is similar to the dining cryptographers solution in Chaum [1988], where a graph characterization of the random bits distribution is given. Our contributions in relation to Chaum [1988] are as follows: (1) we provide a method (an efficient instance that satisfies Chaum's graph characterization) for selecting (a small number of) keys for each processor; (2) we give a new procedure to securely distribute the keys, one that fits global communication networks (unlike the ring assumed in Chaum [1988]); (3) we show that $O(1)$ amortized communication complexity on each link can be achieved for a long enough input (and specify the length); (4) our algorithm is proven correct by a new argument proving that each bit communicated has an equal probability of

being 0 and being 1 for a particular adversary; and (5) while the case of an anonymous sender is considered in Chaum [1988], in the present work we also suggest schemes for the cases in which the receiver(s) is (are) anonymous, and in which both the sender and the receiver are anonymous to each other.

In Chaum [1988], it is assumed that the underlying communication network is a ring and that a back-off mechanism is repeatedly used to send data. In this work we consider the problem of anonymous communication on a spanning tree of a general graph communication network. We note that solutions for star and tree networks are briefly mentioned in Pfitzmann [1985] and Pfitzmann and Waidner [1987], with no details for the way communication starts and terminates for this specific network. Our contribution in relation to Pfitzmann [1985] and Pfitzmann and Waidner [1987] is a detailed design for a (spanning) tree communication network.

Note that in the present work we do not concern ourselves with an active adversary (also called a Byzantine adversary) that can corrupt the program or forge messages on the links, as assumed in Waidner [1989]. The scheme presented in Waidner [1989] is an extension of Chaum [1988] in the following sense: the extension suggested in Waidner [1989] is a design of a fail-stop broadcast instead of assuming reliable broadcast. The transmission stops once the broadcast gives an indication of the arrival of a different input to some of the processors (thus a single Byzantine fault may stop the transmission). We note that for a system with a general communication graph, an active adversary may in fact partition the system into two or more isolated connected components. In such a partition, processors of one connected component cannot communicate with processors of any other connected component. Since we consider systems with a general communication graph, we do not to address this case here.

Contributions: We specify the initialization (including seed distribution), the communication, and the termination procedures that preserve anonymity for the (spanning) tree communication network. We use an extra random sequence (produced by a pseudorandom generator) shared by the sender (and the receiver(s)) to encrypt (decrypt, respectively) the message, avoiding the use of an additional different scheme for encryption and decryption during the transmission of (long) messages. This new approach is appropriate for the transmission of a very long sequence of bits, such as video information to several recipients. Hence it can be used for anonymous multicast such as multicast by cable TV.

Our initialization scheme is designed to cope with the problem of information revealed by the back-off mechanism (see Bos and Boer [1989]) by using a predefined order of transmission. Generally speaking, the back-off mechanism is used to cope with the situation in which several processors simultaneously try to transmit information. When processors identify such a situation, the processors stop transmitting and each tries again following a randomly chosen waiting period. Thus, an outside observer can conclude that (a subset of) the processors that try to transmit would like to communicate with other processors. We solve this problem by giving each

processor a time slot in which it can request communication with other processor(s), all the (encrypted) requests arrive (in a mixed order) at a single processor that can count the number of requests, without gaining any information about the identities of the requesting processors.

In a network of n processors, our algorithm (after a preprocessing stage) sends $O(1)$ bits on each tree link in order to transmit a clear-text bit of data, and each processor computes $O(k)$ pseudorandom bits for the transmission of a clear-text bit. Multiple anonymous transmission is possible by executing in parallel several instances of our algorithm. Each instance uses part of the bandwidth of the communication links. Our algorithm is secure for both *outside adversary* and a *k-listening internal dynamic adversary*.

The rest of this paper is organized as follows. The next section uses a simple example to demonstrate the problem and possible solutions. The problem statement appears in Section 3. The anonymous communication (our Xor-Tree algorithm), which is the heart of our scheme, appears in Section 4. Sections 5 and 6 sketch the anonymous seeds transmission and the initialization and termination schemes, respectively. Extensions and concluding remarks appear in Section 7.

2. A SIMPLE EXAMPLE

In this section we examine a very simple special case in order to illustrate the issues and to offer a solution to this special case. We stress that we develop a general framework that works for the general case (e.g., the case of a general communication graph, unknown receiver, etc.) as well.

Suppose we are dealing with a network with 9 nodes,

$$P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4 \rightarrow P_5 \rightarrow P_6 \rightarrow P_7 \rightarrow P_8 \rightarrow R,$$

where R is the "receiver" node and one of the P_i is the command-and-control center that must broadcast commands to R. The other P_j's for $j \neq i$ are "decoys" used for transmission from P_i to R, and also to "hide" which particular P_i is the real command-and-control center. That is, in this simplified example, we only wish to hide from an adversary the P_i that is the real command-and-control center that sends messages to R. Before we present our solution, we want to examine several inefficient, but natural, simple strategies and then explain their drawbacks.

Communication-inefficient solution: One simple (but inefficient!) way to hide the P_i that is the command-and-control center is for every P_i to broadcast an (encrypted) stream of messages to R. Thus, R receives 8 different streams of messages, ignores all the messages except those from the real command-and-control center, and decrypts that one. Every processor P_i forwards messages of all the smaller-numbered processors, and in addition sends its own message. Clearly, an adversary that is monitoring all the communication channels and that can also monitor the internal memory of one of the P_i's (which is not the actual command-and-control

center) does not know which P_j is broadcasting the actual message. *Drawback:* Notice that instead of one incoming message, R must receive 8 messages, thus the amount of information the real command-and-control center can send to R is only $1/8$ of the total capacity! As the network becomes larger, this solution becomes even more costly. Note that this solution enables the receiver to identify the sender.

Computation-inefficient solution: In the previous example, the drawback is that the messages from decoy command-and-control nodes take up the bandwidth of the channel. In the following solution, we show how this difficulty can be avoided. In order to explain this solution, we use pseudorandom generators[1] [Blum and Micali 1984; Hastad et al. 1999]. We first pick 8 seeds $s_1, \ldots, s_8$ for the pseudorandom generator and give seed s_i to processor P_i. Processor P_1 stretches its seed s_1 into a long pseudorandom sequence and, at each time step, sends the next bit of its sequence to processor P_2. Processor P_2 takes the bit it got from processor P_1 and "xors" it with its own next bit from its pseudorandom sequence $G(s_2)$ and sends it to P_3, and so forth. The processor P_j that is the real command-and-control center additionally "xors" into each bit it sends a bit of the actual message m_i. Processor R is given all the 8 seeds $s_1, \ldots s_8$, so it can take the incoming message, (which is the message from command-and-control center "xored" with 8 different pseudorandom sequences). Hence, R can compute all the 8 pseudorandom sequences, subtract (i.e., xor) the incoming message with all the 8 pseudorandom sequences, and get the original command-and-control message m. The advantage of this solution is that the entire bandwidth of the channel between command-and-control processor and the receiver is used to send the messages from the center to the receiver. Note that any P_j that is not a command-and-control center still cannot deduce which other processor is the real center. *Drawback:* The receiver must compute 8 different pseudorandom sequences in order to recover the actual message. As the network size grows, this becomes prohibitively expensive in the amount of computation the receiver needs to do in order to compute the actual message m.

Our solution: Here, we present a solution that is *both computation-efficient and communication-efficient*, and is secure against an adversary that can monitor all the communication lines, and can also learn the internal memory content of any one of the intermediate processors. The seed distribution (for a particular communication session) is as follows:

—Pick 9 random seeds for pseudorandom generator $s_0, s_1, \ldots, s_8$.

—Give seed s_0 to the real command-and-control processor.

[1]Pseudorandom generator $G(s) = r_1, r_2, \ldots$ takes a small initial "seed" of truly random bits and deterministically expands it into a long sequence of pseudorandom bits. There are many such commercially available pseudorandom generators, and any "off-the-shelf" generator that is sufficiently secure and efficient will suffice.

—Additionally, give seed $\{s_1, s_2\}$ to processor P_1 ; give two seeds $\{s_2, s_3\}$ to processor P_2 ; give two seeds $\{s_3, s_4\}$ to processor P_3, and so on. That is, we give the seeds $\{s_i, s_{i+1}\}$ to each processor P_i for $i \geq 1$.

—Give one seed s_0 to receiver R.

Suppose processor P_4 is the real command-and-control center. Then the distribution of seeds is as follows:

$$P_1(s_1, s_2) \rightarrow P_2(s_2, s_3) \rightarrow P_3(s_3, s_4) \rightarrow P_4(s_4, s_5, s_0) \rightarrow P_5(s_5, s_6) \rightarrow$$

$$\rightarrow P_6(s_6, s_7) \rightarrow P_7(s_7, s_8) \rightarrow P_8(s_8, s_1) \rightarrow R(s_0)$$

Now the transmission of the message is performed in the same fashion as in the previous solution—that is, each processor receives a bit-stream from its predecessor, "xors" a single bit from each pseudorandom sequence that it has, and sends it to the next processor. The command-and-control center "xors" bits of the message into each bit that it sends out.

Note that adjacent processors "cancel" one of the pseudorandom sequences by xoring it twice, but introduce a new sequence. For example, processor P_2 cancels s_2, but "introduces" s_3. Moreover, each processor must now only compute the output of at most three seeds. Yet it can be easily verified that if the adversary monitors all the communication lines and in addition can learn the seeds of any single processor P_i that is not a command-and-control center, then it cannot gain any information as to which P_i is the real command-and-control center, even after learning the two seeds that belong to processor P_i.

Of course, the simplified example that we present only works when the adversary cannot monitor both the actual command-and-control center and the memory contents of the receiver. (We note that these and other restrictions can be resolved; we will address this further in the paper.) Moreover, it should be stressed that the restricted solution presented above does not work if the adversary is allowed to monitor more than one decoy processor. In our example, P_3 and P_5 know the seeds s_4 and s_5 and therefore can observe that P_4 is using an additional seed, the special seed of the sender. Note that our solution requires that the command-and-control and the receiver have a special common seed s_0; one obvious extension is to ensure that every two processors have a distinct additional seed that is used for communication between themselves. We should point out that in the rest of the paper we show how the above scheme can be extended to one that is robust against adversaries that can monitor up to k stations, where in our solution every processor is required to compute the number of different pseudorandom sequences proportional to k only (in particular, at most $2k + 1$). Moreover, we also show how to generalize the method to arbitrary topology networks/infrastructures. Additionally, we show how the initial distribution of seeds can be done without revealing the

command-and-control center. Finally, we show how communication from stations back to the command-and-control center can be achieved without the stations knowing the node of the network at which the center is located, and how totally anonymous communication can be achieved.

2.1 Secret-Key Solutions vs. Public-Key Solutions

The above simple solution is a secret-key solution; that is, we assume that before the protocol begins, a set of seeds for pseudorandom function must be distributed in a private and anonymous manner. Thus, we combine this solution with a preprocessing stage in which we distribute these seeds using a public-key solution; that is, a solution where we assume that all users/nodes only have corresponding public and private keys and do not share any information a priori. Thus, our overall solution is a public-key solution, where we do not assume that users share any private data before communication begins. As usual in many such cryptographic settings, our overall efficiency comes from the fact that we *switch* from public-key to secret key solutions and then show how to (1) make an efficient private-key implementation , and (2) how to set up secret-keys in a preprocessing stage by using public keys in an anonymous and private manner.

3. PROBLEM STATEMENT

A communication network is described by a communication graph $G = (V, E)$. The nodes $V = \{1, \cdots, n\}$ represent the processors of the network. The edges of the graph represent bidirectional communication channels between the processors. Let us first define the assumptions and requirements, starting with the adversary models. The adversary is a *passive* listening adversary that does not intervene in the computation. In particular, it neither forges messages on the links nor corrupts the program of the processors.

—An *outside adversary* is an adversary that can monitor all the communication links, but *not* the contents of the processor's memory.

—An *internal dynamic k-listening adversary* (*inside adversary*, for short) is an adversary that can choose to "bug" (i.e., listen to) the memory of up to k processors. The targeted processors are called *corrupted*, *compromised*, or *colluding* processors. Corrupted processors reveal all the information they know to the adversary, but they still behave according to the protocol. The adversary does not have to choose the k faulty processors in advance. While the adversary corrupts fewer than k processors, the adversary can choose the next processor to be corrupted using the information the adversary gained thus far from the processors that are already corrupted.

The following assumptions are used in the first phase of our algorithm, which is responsible for *seed distribution*. Each of the n processors has a

public-key/private-key pair. The public key of a processor P is known to all the processors, while the private key of P is known only to P.

The anonymity of the communicating parties can be categorized into four cases:

—*Anonymous to the nonparticipating processors:* A processor P wishes to send a message to processor Q without revealing to the rest of the processors and to the inside and outside adversary the fact that P is communicating with Q.

—*Anonymous to the sender and the nonparticipating processors:* P wishes to receive a message from Q without revealing its identity to any processor including Q as well as to an inside and outside adversary.

—*Anonymous to the receiver(s) and the nonparticipating processors:* P wishes to send (or multicast) a message without revealing its identity to any processor, as well as to an inside and an outside adversary.

—*Anonymous to the sender, to the receiver, and the nonparticipating processors:* A processor P wishes to communicate with some other processor, without knowing the identity of the processor and without revealing its identity to any processor, including the one it is communicating with, as well as to an inside and outside adversary. (This is similar to the "chat room" world-wide-web applications, where two processors wish to communicate with one another totally anonymously, without revealing their identity to each other or anybody else.)

In the sequel, we consider the first anonymity category. Then, in Section 7 we show how to extend the solution to the other three anonymity categories.

The *efficiency* of a solution is measured by the *communication overhead*, which is the number of bits sent over each link in order to send a bit of clear-text data.[2] The efficiency is also measured by the *computation overhead*, which is the maximal number of *computation steps* performed by each processor in order to transfer a bit of clear-text data.

The algorithm is a combination of anonymous seeds transmission, initialization, communication, and termination. In the anonymous seeds transmission phase, processors that would like to transmit anonymously send seeds for pseudorandom sequence generators to the rest of the processors. The anonymous seeds transmission phase also resolves conflicts of multiple requests for transmission by an anonymous back-off mechanism. Once the seeds are distributed, the communication can be started. Careful communication,

[2]Note that in a system that has a chain communication graph, like the one presented in Section 2, $\omega(1)$ bits on each link (or a total of $\omega(n)$ bits) are required to transmit a bit from a sender to a receiver when no information concerning the command-and-control can be revealed.

Seeds Distribution —

—Assign (anonymously) a distinct seed s_i to each processor P_i.

—Assign to the sender all the seeds $s_1, s_2, \ldots, s_n$ of all the processors and an additional seed s_0.

—Assign the receiver(s) with an additional seed, the seed s_0.

Upwards Communication :

P **is the sender —**

—Let d_i be the i'th bit of data.

—Let $b_1, b_2, \cdots, b_l$ be the i'th bits received from the children (if any) of P.

—Let b'_0 be the i'th bit of the pseudo random sequence obtained from the additional seed s_0 of P_j.

—let $b'_1, b'_2, \cdots, b'_n$ be the i'th bits of the pseudo random sequence obtained from the seeds $s_1, s_2, \cdots, s_n$.

—The i'th bit P_j sends to its parent (if any) is $d_i \oplus b'_0 \oplus b_1 \oplus b_2 \oplus \cdots b_l \oplus b'_1 \oplus b'_2 \cdots \oplus b'_n$.

P **is not the sender —**

—Let $b_1, b_2, \cdots, b_l$ be the i'th bits received from the children (if any) of P_j.

—Let b'_j be the i'th bit of the pseudo random sequence obtained from the seed s_j.

—The i'th bit that P communicates to its parent (if any) is $b_1 \oplus b_2 \oplus \cdots b_l \oplus b'_j$.

Downwards Communication —

—The root processor calculates an output as if it has a parent and sends the result to every of its children.

—Every processor which is not the root, sends to its children every bit received from its parent.

—The receiver(s) decrypts the downward communication by xoring the i'th bit that arrives from the parent with the i'th bit in the pseudo random sequence obtained from s_0.

Fig. 1. $O(n)$ Computation steps algorithm, for a processor P_j.

initialization, and termination procedures that hide the identity of the sender must be performed.

We first describe the core of our algorithm, which is the communication phase. During the communication phase, seeds are used for the production of pseudorandom sequences. The anonymous distribution of seeds is presented following the description of the anonymous communication phase.

4. ANONYMOUS COMMUNICATION

4.1 The Computation-Inefficient $O(n)$ Solution

The communication algorithm is designed for a spanning tree $\mathcal{T}$ of a general communication graph, where the relation *parent-child* is naturally defined by the election of a root. We start with a simple but inefficient algorithm, which requires $O(n)$ computation steps of a processor. (This algorithm is similar to the computation-inefficient solution presented in Section 1, but for the general-topology graph.) We then show how to make it computation-efficient as well. In this (computation-inefficient) solution the sender chooses a distinct seed for each processor. The sender can then encrypt each bit of information using the seeds of all the processors including its own seeds. Each such seed is used for producing a pseudorandom sequence. The details of the algorithm appear in Figure 1. The symbol $\oplus$ is used to denote the binary xor operation.

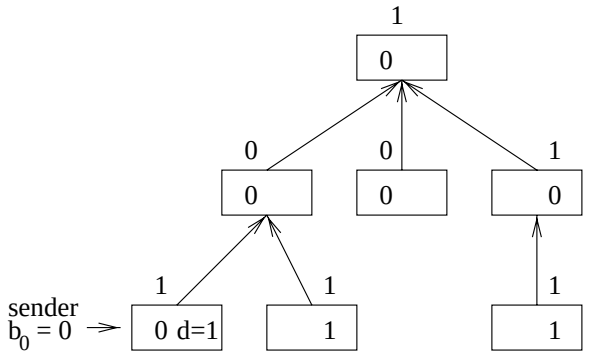

Fig. 2. Upwards communication.

Figures 2 and 3 schematically depict the upwards and downwards communication. Rectangles represent processors and arrows represent the direction of communication on the tree. The number written in each rectangle in Figure 2 is the result of the xor of the ith bits of the pseudorandom sequence of the processor (the number the sender has is the result of xoring all these bits). The output of the processor is the xor of the outputs of its children, xored with the value obtained from its pseudorandom sequence (the number written in the rectangles). In this example, the ith bit of the additional sequence obtained from the the sender's special seed is 0 and the ith bit of data is 1. Figure 3 demonstrates the fact that the root broadcasts the result to every processor in the tree and the receiver uses the special sequence (shared with the sender) to obtain the clear-text. We note that Figures 2 and 3 depict only the way a single bit is handled. In fact, every time unit, a new data bit is handled (in a pipeline fashion) in a similar way.

Note that the ith bit produced by the root is a result of xoring every ith bit of the pseudorandom sequences twice (except the sender's sequence): once by the sender and then, during communication, upwards. Each encrypted bit of data will be xored by the receiver(s) using the senders' seed to reveal the clear-text. Note that the scheme is resilient to any number of colluding processors as long as the sender and the receiver(s) are nonfaulty. This simple scheme requires a single node (the sender) to compute $O(n)$ pseudorandom bits for each bit of data. (We remark that, in contrast, our Xor-Tree algorithm requires the computation of only $O(k)$ pseudorandom bits to cope with an outside adversary and an internal dynamic k-listening adversary.) The next lemma states the communication and computation complexities of the algorithm presented in Figure 1.

LEMMA 4.1 *The next two assertions hold for every bit of data to be transmitted over each edge of the spanning tree:*

—*The communication overhead of the algorithm is* $O(1)$ *per edge.*

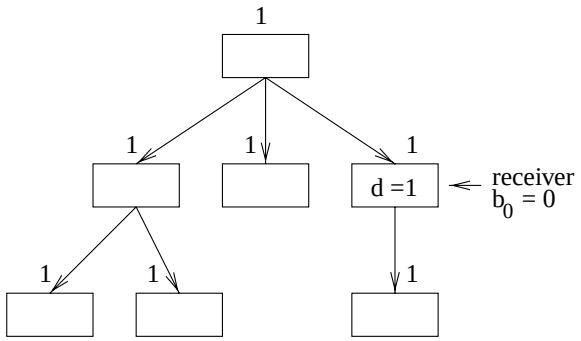

Fig. 3. Downwards communication.

—*The computation overhead of our algorithm is $O(n)$ pseudorandom bits to be computed by each processor per each bit of data.*

PROOF. In each time unit two bits are sent in each link: one upwards and the other downwards. Since a bit of data is sent every time unit (possibly except the first and last h time units, where $h < n$ is the depth of the tree), the number of bits sent over a link to transmit a bit of data is $O(1)$. The second assertion follows from the fact that the sender computes the greatest number of pseudorandom bits in every time unit, namely $O(n)$ pseudorandom bits in every time unit. □

4.2 Towards Our $O(k)$ Solution: The Choice of Seeds

For the realization of the communication phase of our $O(k)$ solution, we use $n(k + 1)$ distinct seeds where k is less than $\lfloor n/2 - 1 \rfloor$. Each processor receives $2(k + 1)$ seeds. To describe the sender's seed distribution decisions, we use $k + 1$ *levels,* each consisting of two *layers* of seeds. We order the processors by their (arbitrary assigned) indices $P_1, P_2, \cdots, P_n$, we use the relation *follows* in a straightforward manner.

The seed distribution procedure appears in Figure 4. An example for the choice of seeds for the processors appears in Figure 5.

The choice of seed made by the sender has the following properties:

—Each seed is shared by exactly two processors (here we use the fact that $k < \lfloor n/2 - 1 \rfloor$).

—Every two processors share at most one seed.

—For every processor P, P shares a (distinct) seed with every processor of the $k+1$ processors that immediately follow P. If there are fewer than $k+1$ processors following P, then P shares a (distinct) seed with every one of the processors that immediately follow P up to and including P_n.

4.3 The Xor-Tree Algorithm

Here we present our main algorithm, the Xor-Tree algorithm, which appears in Figure 6. Note that the algorithm in Figure 6 is different from

The first level — Let $\mathcal{L}^1 = s_1^1, s_2^1, s_3^1, \cdots, s_n^1$ be the seeds that the sender (randomly) chooses for the first level. The sender uses the sequence of seeds $\mathcal{L}_1^1 = s_1^1, s_2^1, s_3^1, \cdots, s_n^1$ for the first layer of the first level and $\mathcal{L}_2^1 = s_2^1, s_3^1, \cdots, s_n^1, s_1^1$ for the second layer. Note that $\mathcal{L}_1^1 = \mathcal{L}^1$ and that $\mathcal{L}_2^1$ is obtained by rotating $\mathcal{L}^1$ once. P_i, $1 < i \leq n$, receives the seeds s_i^1 and s_{i-1}^1. P_1 receives s_1^1 and s_n^1.

The l'th level — Similarly, for the l'th level $1 \leq l \leq k+1$ the sender (randomly) chooses n distinct seeds for this level $\mathcal{L}^l = s_1^l, s_2^l, \cdots, s_n^l$ to be the seeds of the l'th level and uses two sequences $\mathcal{L}_1^l = \mathcal{L}^l$ and $\mathcal{L}_2^l = s_{n-l+1}^l, s_{n-l+2}^l, \cdots, s_n^l, s_1^l, \cdots, s_{n-l}^l$; $\mathcal{L}_2^l$ is obtained by rotating $\mathcal{L}^l$ l times. P_i, $1 \leq i \leq l$, receives the seeds s_i^l and s_{n-l+i}^l and P_j, $l < j \leq n$, receives the seeds s_j^l and s_{i-l}^l.

Thus, at the end of this procedure every processor is assigned by $2k+2$ distinct seeds.

Fig. 4. The choice of seeds.

Seeds of	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9
Level 1	s_1 s_9	s_2 s_1	s_3 s_2	s_4 s_3	s_5 s_4	s_6 s_5	s_7 s_6	s_8 s_7	s_9 s_8
Level 2	s_1' s_8'	s_2' s_9'	s_3' s_1'	s_4' s_2'	s_5' s_3'	s_6' s_4'	s_7' s_5'	s_8' s_6'	s_9' s_7'
Level 3	s_1'' s_7''	s_2'' s_8''	s_3'' s_9''	s_4'' s_1''	s_5'' s_2''	s_6'' s_3''	s_7'' s_4''	s_8'' s_5''	s_9'' s_6''

Fig. 5. An example for the distribution of seeds, where $n = 9$ and $k = 2$.

the algorithm in Figure 1 in the way pseudorandom bits cancel each other in the upwards communication phase. The downwards communication in Figure 6 is only encoded by bits produced from the additional seed s_0.

4.4 An Abstract Game

In this section we describe an abstract game (see Figure 7) that will serve us in analyzing and proving the correctness of the Xor-Tree algorithm in the previous section.

The adversary gets to see the outputs of all the players. The adversary can pick k out of the players and see their seeds. We claim, and later prove, that when the adversary does not pick the sender then every one of the remaining $(n - k)$ processors that are not picked by the adversary is equally likely to be the sender for any *poly-bounded adversary*. A *poly-bounded adversary* is an adversary with computation power bounded by computations of polynomial time. We show that, if the poly-bounded adversary can predict the sender in polynomial time, then we can use this adversary to break a pseudorandom generator in polynomial time (see, e.g., Hastad et al. [1999]).

We proceed by showing that the above assignment of seeds yields a *special seed* ds_P for each processor P. We choose ds_P out of the seeds assigned to each nonfaulty processor P. We order the processors by their index in a cyclic fashion such that the processor that follows the ith processor, $i \neq n$, is the processor with the index $i + 1$, and the processor that follows the nth processor is the first processor. Then we assign a new index for each processor such that the sender has the index one, the

Seeds Distribution —

—Assign seeds to the processors as described in Figure 4.

—Assign the sender with one additional seed, s_0.

—Assign the receiver(s) with an additional seed, the seed of the sender s_0.

Upwards Communication :

P_j **is the sender —**

—Let d_i be the i'th bit of data.

—Let $b_1, b_2, \cdots, b_l$ be the i'th bits received from the children (if any) of P_j.

—Let $b'_1, b'_2, \cdots, b'_{2k+2}$ be the i'th bits of the pseudo-random sequences obtained from the seeds of P_j.

—Let b'_0 be the i'th bit of the pseudo-random sequence obtained from the additional seed s_0 of P_j.

—The i'th bit P_j sends to its parent (if any) is $d_i \oplus b_1 \oplus b_2 \oplus \cdots b_l \oplus b'_1 \oplus b'_2 \cdots \oplus b'_{2k+2} \oplus b'_0$.

P_j **is not the sender —**

—The i'th bit that P_j communicates to its parent (if any) is $b_1 \oplus b_2 \oplus \cdots b_l \oplus b'_1 \oplus b'_2 \cdots \oplus b'_{2k+2}$.

Downwards Communication —

—The root processor calculates an output as if it has a parent and sends the result to every of its children.

—Every non root processor send to its children every bit received from its parent.

—The receiver(s) decrypts the downward communication by xoring the i'th bit that arrives from the parent with the i'th bit in the pseudo random sequence obtained from s_0.

Fig. 6. The Xor-tree algorithm, for a processor P_j.

Seeds Assignment — Assign seeds to the processors as described in Figure 4. Assign the sender with one additional seed.

Computation — Each processor, P, uses its seeds to compute pseudo-random sequences. At the i'th time unit the sender S computes the i'th bit of every of its pseudo-random sequences, xors these bits and the i'th bit of data and outputs the result. At the same time unit every other processor P computes the i'th bit of every of its pseudo-random sequences xors these bits and outputs the result.

Fig. 7. The abstract game.

processor that follows the sender has the index two, and so on. These new indices are used for the interpretation of next, follows, prior, and last in the description of the choice of special seeds that appears in Figure 8. Recall that every two processors share at most one seed.

Note that by our special seed selection, described in Figure 8, the special seeds are not known to the k faulty processors.

THEOREM 4.2 *In the abstract game, any of the* $(n - k)$ *nonfaulty processors is equally likely to be the sender for any poly-bounded internal* k*-listening adversary.*

PROOF. We prove that the ith bit produced by any nonfaulty processors is equally likely to be 0 or 1 (for any poly-bounded adversary). Let P be the first nonfaulty processor that follows the sender (P is among the first $k + 1$ processors that follow the sender). Let ds_{P_1} be the special seed of the sender that is shared only with (the nonfaulty processor) P. The ith bit that the sender outputs is a result of a xor operation with the ith bit of the

The sender P_1 — Each of the $k+1$ processors that immediately follows the sender shares exactly one seed with the sender. Since there are at most k colluding processors, one of these $k+1$ processors must be non-faulty. Pick, P, the first such non-faulty processor. Assign ds_{P_1}, the special seed of the sender, to be the seed that the sender shares with P.

A processor P that is not among the $k+1$ last processors — A processor P has $k+1$ seeds that belong to the first layers (and additional $k+1$ seeds that belong to the second layers) of the $k+1$ levels. Define the $k+1$ seeds of P that belong to the first layer as the *new* seeds of P — seeds that do not belong to any processor that proceeds P. Since there are at most k colluding processors, one of the next $k+1$ processors is non-faulty. Let Q be the first such non-faulty processor and assign ds_P by the new seed that P shares with Q. Repeat the procedure until you reach a non-faulty processor that is among the last $k+1$ processors.

A processor Q that is among the $k+1$ last processors — Note that Q does not introduce $k+1$ new seeds since some of its seeds are assigned to the first processors (at least a seed of the $k+1$'th level). Fortunately, Q shares a single new seed with every of the last processors. This fact allows us to continue the special seed selection procedure, by choosing the new seed shared with the next non-faulty processor.

Fig. 8. Choice of special seeds.

pseudorandom sequence (among other pseudorandom sequences) obtained from ds_{P_1}. Since only P (that is a nonfaulty processor) shares ds_{P_1} with the sender, it holds that the ith bit output by the sender is equally likely to be 0 or 1 (for any poly-bounded internal i-listening adversary). A similar argument hold for the output of P, since there exists a special seed shared with the next nonfaulty processor Q. In general, it holds for the output of every nonfaulty processor. The same argument holds if *any* of the $n-k$ nonfaulty processors is the sender. Thus, for any polynomially-bounded k-internal and external adversary, the distribution of the output is not correlated to the identity of the sender. □

The fact that the adversary can be a dynamic adversary is implied by Corollary 4.4. The proof of the corollary is similar to the proof of Theorem 4.2.

COROLLARY 4.3 *For any $k' \leq k$ after the adversary chooses k' faulty processors, any of the $(n-k')$ nonfaulty processors is equally likely to be the sender for any poly-bounded internal k'-listening adversary.*

4.5 Reduction to the Abstract Game

In this section we prove that if there is an algorithm that reveals information about the identity of the sender in the tree, then there exists an algorithm that reveals information about the identity of the sender in the abstract game. The above reduction together with Theorem 4.2 yields the proof of correctness for the Xor-Tree algorithm.

Roughly speaking, the reduction starts assuming that there exists an adversary $\mathcal{A}$ that reveals information about the sender in a tree $\mathcal{T}$ of n processors. Then we show that an adversary $\mathcal{A}'$ that reveals information from the abstract game exists. This last adversary uses the output of the abstract game to simulate the output of the tree. Then $\mathcal{A}'$ runs $\mathcal{A}$ to reveal information on the sender in the tree. In more detail, an abstract game of n nodes is mapped to the tree, as described in Figure 9.

(1) Each processor of the abstract game is assigned to a node of the tree $\mathcal{T}$.

(2) The output of every processor to its parent is computed as follows: Let the *height* of a processor P in $\mathcal{T}$ be the number of edges in the longest path $\mathcal{P}$ from P to a leaf such that $\mathcal{P}$ does not traverse the root. We start with the processors that are in height 0 i.e. the leaves. The output of the processors that were assigned to the leaves of the tree is not changed i.e. it is identical to their output in the abstract game. Once we computed the output of processors in height h we use these computed outputs to compute the outputs of processors in height $h+1$. Let Q be a processor in height $h+1$, let $b_1, b_2, \cdots, b_l$ be the i'th computed bits that are output by the children of Q, and let b_Q be the original i'th output bit of Q in the abstract game. The computed output of Q is $b_1 \oplus b_2 \oplus \cdots b_l \oplus b_Q$.

Fig. 9. The reduction.

THEOREM 4.4 *In the Xor-Tree algorithm, any of the $(n - k)$ nonfaulty processors is equally likely to be the sender for any poly-bounded internal k-listening adversary.*

PROOF. If there exists an adversary $\mathcal{A}$ that reveals information about the identity of the sender in a tree $\mathcal{T}$, then there exists an abstract game with the same number of processors and the same seed distribution, such that the application of the reduction in Figure 9 yields the communication pattern of $\mathcal{T}$ and reveals information about the sender's identity in the abstract game. This contradicts Theorem 4.2, and thus contradicts the existence of $\mathcal{A}$. □

The next Lemma states the communication and computation overheads of the anonymous communication algorithm.

LEMMA 4.5 *The next two assertions hold for every bit of data to be transmitted over each edge of the spanning tree:*

—*The communication overhead of the algorithm is $O(1)$ per edge.*

—*The computation overhead of our algorithm is $O(k)$ pseudorandom bits to be computed by each processor per each bit of data.*

PROOF. In each time unit two bits are sent in each link: one upwards and the other downwards. Since a bit of data is sent every time unit (possibly except the first and last h time units, where $h < n$ is the depth of the tree), the number of bits sent over each link to transmit a bit of data is $O(1)$. The second assertion follows from the fact that in each time unit each processor generates at most $2k + 3$ pseudorandom bits. □

5. ANONYMOUS SEED TRANSMISSION

We first outline the main ideas in the seed transmission scheme and then give full details. Every processor has a public-key that is known to all other processors. A virtual ring defined by the Euler tour[3] on the tree is used for seed transmission. Note that the indices of the processor in this description

[3]Called a Eulerian circuit [Knuth 1997, p. 374].

are related to their location on the virtual ring. First, all processors send messages to P_1 over the (virtual) ring. Those processors that wish to broadcast send a collection of seeds, and those processors that do not wish to broadcast, send dummy messages of equal length. To do so in an anonymous fashion (so that P_1 does not know which message is from which processor), $k + 1$ of Chaum's *mix*es [Chaum 1981] are used, where $k + 1$ (real) processors just before P_1 in the Euler tour are used as mixes. Hence, P_1 can identify the number of nondummy arriving messages, but not their origin. In case more than one nondummy message reaches P_1, a standard back-off algorithm is initiated by P_1. Namely, P_1 notifies the processors that more than one processor sent a message, and the processors use randomization to decide whether to resend a message during the new seed collection procedure that P_1 starts. Once exactly one message (containing a collection of seeds) arrives at P_1, the seed distribution procedure described above (for sending a collection of seeds to P_1) is used to send the seeds to P_2 and so on. (At this point processors know that only one processor wishes to broadcast.) This procedure is repeated n times in order to allow the anonymous sender to transmit a collection of seeds to every processor.

The details follow. The seed transmission procedure uses a virtual ring $\mathcal{R}$ defined by a Euler tour of the tree $\mathcal{T}$. Note that each edge of $\mathcal{T}$ appears exactly twice in $\mathcal{R}$, and therefore the number of edges and nodes in $\mathcal{R}$ is $2n - 2$. The seed transmission procedure starts with the transmission of seeds to the first processor P_1. Let $\mathcal{L}_1 = P_1, P_2, \cdots, P_{2n-2}$ be the list of processors in $\mathcal{R}$ in clockwise order starting with P_1; the indices 2 to $2n - 2$ are implied by the Euler tour and not by the indices of the processors in $\mathcal{T}$. Note that a single processor of $\mathcal{T}$ may appear more than once in $\mathcal{L}_1$. We use the term *instance* for each such appearance. Define the reduced list $\mathcal{RL}_1$ to be a list of processors that is obtained from $\mathcal{L}_1$ by removing all but the first instance of each processor. Thus, in $\mathcal{RL}_1$ every processor of $\mathcal{T}$ appears exactly once. The communication of seeds uses the anticlockwise direction. Define *the last l real processors* to be the first l processors in $\mathcal{RL}_1$. When transmitting seeds to P_i, $\mathcal{L}_i$, $\mathcal{RL}_i$ and the last l processors, are defined analogously.

In the first stage, every processor that wants to communicate with another processor sends an encrypted message with the seeds to be used by P_1. Note that P_1 can be a faulty processor, thus a careful transmission must be carried on. Let $\overline{\mathcal{L}}_1 = P_{2n-2}, P_{2n-3}, \cdots, P_1$ be the list of processors in $\mathcal{R}$ in anticlockwise order i.e., $\mathcal{L}_1$ in reverse order. Again, $\overline{\mathcal{L}}_1$ includes more than one instance of each processor P of $\mathcal{T}$. Define the *active instance* of a processor P of $\mathcal{T}$ in $\overline{\mathcal{L}}_1$ to be the last appearance of P in $\overline{\mathcal{L}}_1$. Define an *active message* to be a message that arrives to an active instance of a processor. The details of the anonymous seed transmission to P_1 appears in Figure 10.

As we prove in the sequel, no information concerning the identity of the requesting processors is revealed during the anonymous seed transmission

P_n **starts** — The first processor to send a message m_n to P_1 is P_n. If P_n wants to transmit data then m_n contains seeds to be used by P_1, otherwise m_n is an empty message i.e. a message that can be identified by P_1 as a null message. P_n uses the public keys $pu_1, pu_2, \cdots, pu_{k+1}$ of the last $k+1$ real processors $P_1, P_2, \cdots, P_{k+1}$ to encrypt m_n in a nested fashion. First encrypting m_n with pu_1 then encrypting the resulting message with pu_2 and so on. P_n sends the $k+1$-nested encrypted message m_n^{k+1} to the processor that is next to the active instance of P_n in $\overline{\mathcal{L}}_1$.

Non active message — When a processor P_i receives a (non active) message it forwards the message to the next processor according to $\overline{\mathcal{L}}_1$.

Active message — We now proceed by describing the actions taken by a processor upon the arrival of an active message.

—We first describe the action taken by a processor P_i that is not among the last $k+1$ real processors. When an active message with $\{m_n^{k+1}, m_{n-1}^{k+1}, \cdots, m_{i+1}^{k+1}\}$ arrives (to the active instance of) P_i, P_i adds m_i^{k+1} its own $k+1$-encrypted message (again, containing seeds to be used by P_1 or null message) to the message received and sends the message to the next processor according to $\overline{\mathcal{L}}_1$.

—We now turn to consider a processor P_i that is among the last $k+1$ real processors. When an active message with $\{m_n^i, m_{n-1}^i, \cdots, m_{i+1}^i\}$ arrives to P_i then P_i decrypts every m^i in $\{m_n^i, m_{n-1}^i, \cdots, m_{i+1}^i\}$ using its private key to obtain $\{m_n^{i-1}, m_{n-1}^{i-1}, \cdots, m_{i+1}^{i-1}\}$. P_i encrypts m_i its message to P_1 by the public keys of the last $i-1$ real processors. P_i randomly orders the set $\{m_n^{i-1}, m_{n-1}^{i-1}, \cdots, m_{i+1}^{i-1}, m_i^{i-1}\}$ and sends the reordered set to the processor that is next according to $\overline{\mathcal{L}}_1$ (Note that following the first such reordering the j'th index of m_j^{i-1} is not necessarily the index of the sender of m_j^{i-1}).

Arrival to P_1 — When P_1 receives an active message with $\{m_n^1, m_{n-1}^1, \cdots, m_2^1\}$, P_1 decrypts every message and finds out the number n_t of the processors that would like to transmit. If $n_t \neq 1$ then P_1 sends a message with the value of n_t that traverses the virtual ring. Upon receiving such a message each processor that wants to transmit, randomly chooses a waiting time in the range of say, 1 to $2n_t$. The procedure of sending seeds to P_1 is repeated until $n_t = 1$.

Fig. 10. Anonymous seed transmission to P_1.

to P_1 except the information that can be drawn from the value of n_t—the number of processors that would like to transmit. Once $n_t = 1$, the processors start sending messages to P_2 in a fashion similar to the one used to send seeds to P_1. Then processors send seeds to P_3 and so on, until the processors send messages to P_n. Note that when $n_t = 1$ there is exactly one sender for the next communication session, and at the end of the seed distribution procedure every processor holds the seeds distributed by the sender.

LEMMA 5.1 *A coalition of k colluding processors cannot reveal the identity of the seed distributors.*

PROOF. We prove the lemma for the transmission of seeds from the sender to P_1. Note that one of the last $k+1$ real processors must be nonfaulty. If P_1 is nonfaulty, then no information concerning the identity of the seed distributors is revealed to the adversary. Otherwise, when P_1 is faulty, let P_i be the nonfaulty processor that is the last to reorder the set $\{m_n^{i-1}, m_{n-1}^{i-1}, \cdots m_i^{i-1}\}$ upon the arrival of $\{m_n^i, m_{n-1}^i, \cdots m_{i+1}^i\}$. Since every arriving m^i is encrypted with P_i's public key, no set of k-faulty processors

can decrypt m^i (unless m^i was originated by a faulty processor). P_i randomly orders the set $\{m_n^{i-1}, m_{n-1}^{i-1}, \cdots m_i^{i-1}\}$, thus it holds that a coalition of k processors cannot reveal the identity of the sender of any m^{i-1} in $\{m_n^{i-1}, m_{n-1}^{i-1}, \cdots m_i^{i-1}\}$. □

The next lemma states the communication overhead of the seed transmission procedure.

LEMMA 5.2 *The expected number of bits sent during the seed transmission procedure is* $O(g(kn)^2)$.

PROOF. Each (of the n) processors sends $O(k)$ encrypted seeds of size g to P_1. A seed that is encrypted k times consists of $O(kg)$ bits. Hence, the communication to P_1 consists of $nk^2g\log n$ bits (where the $\log n$ factor is related to the expected number of back-offs). The communication to the rest of the n processors consists of n^2k^2g bits, which implies $O(g(kn)^2)$ bits per an edge. Thus, as long as the message size p to be broadcast is greater than $O(g(kn)^2)$, we achieve $O(1)$ overall amortized cost per edge, and otherwise we get $O(g(kn)^2/p)$ amortized cost. □

6. INITIALIZATION AND TERMINATION

When the seed distribution procedure is over, the transmission of data may start. P_n broadcasts a signal on the tree that notifies the leaves that they can start transmitting data. The leaves start sending data in a way that ensures that every nonleaf processor receives the ith bit from its children simultaneously. Thus, the delay in starting transmission of a particular leaf l is proportional to the difference between the longest path from a leaf to the root and the distance of l from the root. Each nonleaf processor waits to receive the ith bit from each of its children, uses these bits and its seeds to compute its own ith bit, and sends the output to its parent. Note that buffers can be used in case the processors are not completely synchronized. A processor P that receives the ith bit from one of its children will buffer this bit until P receives the ith bit from every one of its children, then computes its output and sends it to its parent.

The above procedure implies a latency of $O(h)$ time units (where h is the depth of the tree) for each bit to arrive at the root of the tree and then to its destination(s). Note however that these $O(h)$ time units can be viewed as part of preprocessing. Once preprocessing is over, a new bit is sent (and received) in every time unit.

The sender can terminate the session by sending a termination message that is not encrypted by its additional seed. This message will be decrypted by the root that will broadcast it to the rest of the processors to notify the beginning of a new anonymous seed transmission.

7. EXTENSIONS AND CONCLUDING REMARKS

Our treatment so far considered the anonymous sender case, which is also anonymous to the nonparticipating processors.

Anonymous receiver: A simple modification of the algorithm can support the anonymous receiver case: The receiver plays the role of a sender of the previous solution in order to communicate, in an anonymous fashion, an additional seed to the sender. Then the sender uses the same scheme for the anonymous sender case with the seed the sender got from the receiver.

Anonymous sender and receiver: To achieve anonymous communication in which both the sender and the receiver are anonymous, we propose the following procedure: The two participants, P and Q, that would like to communicate (each) send anonymously distinct seeds to $P_1, \ldots, P_{k+1}$. It is possible that more than two participants will send anonymously distinct seeds to P_1. In such a case, P_1 will broadcast to the processors that more than two processors tried to chat anonymously, and a back-off mechanism will be used until exactly two participants, P and Q, send seeds to P_1. Then, P_1 will encrypt and broadcast the two seeds it got, each seed encrypted (using distinct intervals of the pseudorandom expansions of the two seeds) by the other seed. Hence, each of the two processors will use its seed to reveal the seed of the other processor. At this stage, P and Q will continue and anonymously send seeds to $P_2, \ldots, P_{k+1}$. The same procedure continues for $P_2, P_3, P_4 \cdots P_{k+1}$. Now P has a set of $k + 1$ seeds that are used for encryption of messages sent to Q, and Q has $k + 1$ seeds used for encryption messages sent to P. They both act as senders, using the bit resulting from xoring the bits produced by the set of the $k + 1$ seeds as the bit of special seed known to the receiver in our anonymous sender scheme. The back-off mechanism ensures that one of P and Q starts the communication and then the other can replay (when the first allows him to, i.e., stop transmitting data). We remark that it is possible to have more than two participants by a similar scheme.

The security of the above algorithm is derived from the fact that there must be a nonfaulty processor among the processor $P_1, P_2, \ldots, P_{k+1}$, and therefore the adversary does not know at least one key used to encrypt and decrypt messages by the sender and the receiver.

ACKNOWLEDGMENTS

We thank Oded Goldreich, Ron Rivest, and the anonymous referees for helpful remarks.

REFERENCES

BEN-OR, M., GOLDWASSER, S., AND WIGDERSON, A. 1988. Completeness theorems for noncryptographic fault-tolerant distributed computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing* (STOC '88, Chicago, IL, May 2-4), J. Simon, Chair. ACM Press, New York, NY, 1–10.

BLUM, M. AND MICALI, S. 1984. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput. 13*, 4 (Nov.), 850–864.

BOS, J. AND DEN BOER, B. 1990. Detection of disrupters in the DC protocol. In *Proceedings of the Workshop on Advances in Cryptology* (EUROCRYPT '89, Houthalen, Belgium, Apr. 10–13), J.-J. Quisquater and J. Vandewalle, Eds. Springer Lecture Notes in Computer Science. Springer-Verlag, New York, NY, 320–327.

CANETTI, R., FEIGE, U., GOLDREICH, O., AND NAOR, M. 1996. Adaptively secure multi-party computation. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing* (STOC '96, Philadelphia, PA, May 22–24), G. L. Miller, Chair. ACM Press, New York, NY, 639–648.

CANETTI, R., KUSHILEVITZ, E., OSTROVSKY, R., AND ROSÉN, A. 1997. Randomness vs. fault-tolerance. In *Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing* (PODC '97, Santa Barbara, CA, Aug. 21–24), J. E. Burns and H. Attiya, Chairs. ACM Press, New York, NY, 35–44.

CHAUM, D. 1981. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM 24*, 2 (Feb.), 84–88.

CHAUM, D. 1988. The dining cryptographers problem: Unconditional sender and recipient untraceability. *J. Cryptology 1*, 1 (), 65–75.

CHAUM, D. AND DAMGARD, I. 1988. Multiparty unconditionally secure protocols. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing* (STOC '88, Chicago, IL, May 2-4), J. Simon, Chair. ACM Press, New York, NY, 11–19.

FRANKLIN, M., GALIL, Z., AND YUNG, M. 1993. Eavesdropping games: A graph-theoretic approach to privacy in distributed systems. In *Proceedings of the IEEE Symposium on Foundations of Computer Science* (FOCS '93). IEEE Computer Society Press, Los Alamitos, CA.

GOLDREICH, O., MICALI, S., AND WIGDERSON, A. 1987. How to play ANY mental game. In *Proceedings of the 19th Annual ACM Conference on Theory of Computing* (STOC '87, New York, NY, May 25-27), A. V. Aho, Ed. ACM Press, New York, NY, 218–229.

HASTAD, J., IMPAGLIAZZO, R., LEVIN, L. A., AND LUBY, M. 1999. Construction of a pseudorandom generator from any one-way function. *SIAM J. Comput. 28*, 4, 1364–1396.

KNUTH, D. E. 1997. *The Art of Computer Programming*. 3rd ed. Addison-Wesley, Reading, MA.

KUSHILEVITZ, E., MICALI, S., AND OSTROVSKY, R. 1994. Reducibility and completeness in multi-party private computations. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science* (Nov.). IEEE Computer Society Press, Los Alamitos, CA.

PFITZMANN, A. 1985. How to implement ISDNs without user observability: some remarks. Tech Rep. TR 14/85. University of Karlsruhe, Germany.

PFITZMANN, A. AND WAIDNER, M. 1987. Networks without user observability. *Comput. Secur. 6*, 2 (Apr.), 158–166.

PFITZMANN, A., PFITZMANN, B., AND WAIDNER, M. 1991. ISDN-mixes: Untraceable communication with very small bandwidth overhead. In *Proceedings of the GI/ITG Conference on Communication in Distributed Systems* (Feb., Mannheim, Germany). 451–463.

RACKOFF, C. AND SIMON, D. R. 1993. Cryptographic defense against traffic analysis. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing* (STOC '93, San Diego, CA, May 16–18), R. Kosaraju, D. Johnson, and A. Aggarwal, Chairs. ACM Press, New York, NY, 672–681.

SYVERSON, P. F., GOLDSCHLAG, D. M., AND REED, M. G. 1997. Anonymous connections and onion routing. In *Proceedings of the 1997 IEEE Computer Society Symposium on Reasearch in Security and Privacy* (Oakland, CA, May). IEEE Computer Society Press, Los Alamitos, CA.

WAIDNER, M. 1990. Unconditional sender and recipient untraceability in spite of active attacks. In *Proceedings of the Workshop on Advances in Cryptology* (EUROCRYPT '89, Houthalen, Belgium, Apr. 10–13), J.-J. Quisquater and J. Vandewalle, Eds. Springer Lecture Notes in Computer Science. Springer-Verlag, New York, NY, 302–319.

WAIDNER, M. AND PFITZMANN, B. 1990. The dining cryptographers in the disco: Unconditional sender and recipient untraceability with computationally secure serviceability. In *Proceedings of the Workshop on Advances in Cryptology* (EUROCRYPT '89, Houthalen, Belgium, Apr. 10–13), J.-J. Quisquater and J. Vandewalle, Eds. Springer Lecture Notes in Computer Science. Springer-Verlag, New York, NY, 690.

Received: January 1999; revised: September 1999; accepted: March 2000