Xor-Trees for Efficient Anonymous Multicast and Reception*

Shlomi Dolev† Rafail Ostrovsky‡

November 10, 1998

Abstract

In this work we examine the problem of efficient anonymous broadcast and reception in general communication networks. We show an algorithm which achieves anonymous communication with $O(1)$ amortized communication complexity on each link and low computational complexity. In contrast, all previous solutions require polynomial (in the size of the network and security parameter) amortized communication complexity.

*An extended abstract of this paper appears in the *Proc. of the 17th Annual IACR Crypto Conference, CRYPTO 1997.*

†Department of Mathematics and Computer Science, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel. Email: dolev@cs.bgu.ac.il. Part of this work was done while this author visited Bellcore with the support of DIMACS. Partially supported by the Israeli ministry of science and arts grant #6756195.

‡Bell Communications Research, 445 South St., MCC 1C-365B, Morristown, NJ 07960-6438, USA. Email: rafail@bellcore.com.

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/agents. Hence, one of the crucial ingredients in almost any network with command centers is to conceal and to confuse the adversary regarding which stations issue the commands. This paper shows how to use standard off-the-shelf cryptographic tools in a novel way in order 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 who is the sender and the receiver (or both) of the message/directive in a variety of threat models. 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) and in terms of computation efficiency (i.e., how much computation must be performed for concealment). Moreover, we establish rigorous guarantees about the proposed solutions.

1.1 The problem considered

Modern cryptographic techniques are extremely good in hiding all the *contents* of data, by means of encrypting the messages. However, hiding the contents of the message does not hide the fact that *some* message was sent from or received by a particular site. 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 these 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, the problem of how to hide, in an efficient manner, which site (i.e. command-and-control center) transmits (or receives) a lot of data to (or from, respectively) other sites in the network. This question was addressed previously in the literature [6, 20] at the price of polynomial communication overhead for each bit of transmission per edge. We show an amortized solution which after a fixed pre-processing 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 here the means of communication, i.e. it 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. To reiterate,

the question we are addressing in this paper is how we can hide which site 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.

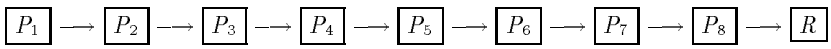
A simple attack to defend against is of a restricted adversary (called outside adversary) who is allowed only to monitor communication channels, but is *not* allowed to infiltrate/monitor the internal contents of any processor of the network. (As a side remark, such weak attack is very easy to defend against: all processors simply transmit either noise or encrypted messages on each communication channel – if noise is indistinguishable from encrypted traffic this completely hides a communication pattern.) Of course, a more realistic adversary, (and the one that we are considering in this paper) is the (internal) adversary that can monitor all the communication between stations and which *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, where he manages to compromise the security of one or several *internal* nodes of the network, whereby he is now not only capable of monitoring the external traffic pattern but is also capable of examining every message and all the data which passes through (or stored at) this infiltrated node. Thus, we define an *internal k-listening adversary*, an adversary 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 [20, 5] and references therein). We remark, though, that in this paper we restrict out attention only to listening adversary, that only monitors traffic, but does not try to sabotage it, similar to [10, 14], but with different objectives. Before we proceed, let us turn to a simple example, to better explain what are the issues that must be addressed.

1.3 A simple example

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

Suppose we are dealing with a network having 9 nodes:



where R is the "receiver" node and one of the P_i is the command-and-control center which must broadcast commands to R. The other P_j's for $j \neq i$ are "decoys" which are used for transmission purposes from P_i to R and also are used 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 which of the P_i is the real command and control center which sends messages to R. Before we explain our solution, we examine several inefficient, but natural to consider simple strategies and then explain what are their drawbacks.

Communication-inefficient solution: One simple (but inefficient!) way to hide which P_i 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 who is monitoring all the communication channels and which 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 throughput of how much information the real command-and-control center can send to R is only $\frac{1}{8}$ of the total capacity! As the network becomes larger this solution becomes even more costly.

Computation-inefficient solution: In the previous example, the drawback was that the messages from decoy command-and-control nodes were taking 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 shall use pseudo-random generators[1] [2, 12, 13]. We first pick 8 seeds $s_1, \ldots, s_8$ for the pseudo-random generator, and give to processor P_i seed s_i. Processor P_1 stretches its seed s_1 into long pseudo-random sequence, and sends, at each time step 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 pseudo-random sequence $G(s_2)$ and sends it to P_3 and so forth. The processor P_j which is the real command-and-control center additionally "xors" into each bit it sends out 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 pseudo-random sequences.) Hence, R can compute all the 8 pseudo-random sequences, subtract (i.e. xor) the incoming message with all the 8 pseudo-random sequences and get the original command-and-control message m. The advantage of this solution is that any P_j which is not a command-and-control center (and not R), clearly can not deduce which other processor is the real center. Moreover, 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. **Drawback:** The receiver must compute 8 different pseudo-random sequences in order to recover the actual message. As the network size grows, this becomes prohibitively expensive in terms of the computation that the receiver needs to perform in order to compute the actual message m.

Our solution for this simple example: 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 additionally can learn internal memory contents of any one of the intermediate processors. The seed distribution is as follows:

[1] Pseudo-random 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 pseudo-random bits. There are many such commercially available pseudo-random generators, and any such "off-the-shelf" generator that is sufficiently secure and efficient will suffice.

- Pick 9 random seeds for pseudo-random generator $s_0, s_1, \ldots, s_8$.
- Give to the real command-and-control processor seed s_0.
- Additionally, give to processor P_1 seed $\{s_1, s_2\}$; to processor P_2 two seeds $\{s_2, s_3\}$, to processor P_3 two seeds $\{s_3, s_4\}$, and so on. That is, we give to each processor P_i for $i > 1$ the seeds $\{s_i, s_{i+1}\}$.
- give to receiver, R, one seed s_0

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

$$\boxed{P_1(s_1, s_2)} \longrightarrow \boxed{P_2(s_2, s_3)} \longrightarrow \boxed{P_3(s_3, s_4)} \longrightarrow \boxed{P_4(s_4, s_5, s_0)} \longrightarrow \boxed{P_5(s_5, s_6)} \longrightarrow$$

$$\longrightarrow \boxed{P_6(s_6, s_7)} \longrightarrow \boxed{P_7(s_7, s_8)} \longrightarrow \boxed{P_8(s_8, s_1)} \longrightarrow \boxed{R(s_0)}$$

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 pseudo-random 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.

Notice, that adjacent processors "cancel" one of the pseudo-random 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 seeds of any single processor P_i which is not a command and control center, then it can not gain any information as to which other 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 presented works only provided that the adversary cannot monitor both the actual command-and-control center and can not monitor the memory contents of the receiver. (We note that these and other restrictions can be resolved – we 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. 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 pseudo-random 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 initial distribution of seeds can be done without revealing the command-and-control center and how the actual location of the command-and-control center can be hidden from the recipients of the messages as well. At last, we show how communication from stations back to the command-and-control center could be achieved without the stations knowing at which node of the network the center is located and how totally anonymous communication can be achieved.

1.4 Private-key solutions vs. Public-key solutions

The above simple solution is a private-key solution, that is, we assume that before the protocol begins, a set of seeds for pseudo-random function must be distributed in a private and anonymous manner. We note that we show how to 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 before communication begins, we do not assume that users share any private data. As usual in many of such cryptographic setting, our overall efficiency comes from the fact that we *switch* from public-key to private key solution and then show how to (1) make an efficient private-key implement and (2) how to set up private keys in a pre-processing stage by using public keys in an anonymous and private manner.

1.5 Comparison with Previous Work

One of the first works (if not the first one) to consider the problem of hiding the communication pattern in the network is the work of Chaum [6] where he introduced the concept of a *mix*: 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 in this direction appear in [15, 17, 18]. 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 where the messages are forwarded to. Hence, as Chaum pointed out, a sequence of "mixes" must be employed at the price of additional communication and computation. Moreover, the single mix scheme operates under some statistic assumption on the pattern of communication. In case a single message is sent to the mix then 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 [20] who embedded an n-element sorting network of depth polynomial 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 statistic assumptions on the pattern of communication is somewhat relaxed in [20] by introducing dummy communication: Every processor sends a message simultaneously. However, the number of (real and dummy) messages arriving to each destination is available to the traffic analyzer. Rackoff and Simon also presented in [20] a scheme that copes with passive internal adversaries by the use of randomly chosen committees and multi-party computation (e.g., [11, 3, 7, 4, 5].)

More generally, secure multi-party computation can be used to hide the communication pattern in the network (see, for example, [11, 8, 19, 3, 7, 4, 5]) 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.

In particular, David Chaum suggested in [8] to use the dc-net approach in order to achieve anonymous communication. Our approach is similar to the dining cryptographers solution in [8]. We present a specific way to select (small number of) keys for each processor, to securely distribute the keys and use $O(1)$ amortized communication complexity on each link. Our algorithm is proven correct by a new argument proving that each bit communicated has an equal probability to be 0 and to be 1 for a particular adversary. In [8] the case of anonymous sender is considered, in this work we suggest schemes also for the cases in which the receiver is anonymous, and in which both the sender and the receiver are anonymous to each other.

In [8] it is assumed that the underlying communication networks is a ring or 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 mentioned in [15, 16]. Our contributions include the scheme for seeds selection, the schemes for anonymous receiver as well as anonymous sender and receiver, and the rigorous treatment and proof of security.

In a network of n processors our algorithm (after a pre-processing stage) sends $O(1)$ bits on each tree link in order to transmit a clear-text bit of data and each processor computes $O(k)$ pseudo-random 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 *k-listening internal dynamic adversary*. (We remark, though, that we are only considering eavesdropping "listening" adversary, similar to [10, 14], and *do not* consider a Byzantine adversary which tries to actively disrupt the communication, as in [11].) Our algorithm starts with anonymous *seeds* distribution. These seeds are later used for the generation of pseudo-random sequences.

The rest of the paper is organized as follows. The problem statement appears in Section 2. The anonymous communication (our Xor-Tree Algorithm) which is the heart of our scheme appears in 3. Section 4 and 5 sketch the anonymous seeds transmission and the initialization and termination schemes, respectively. Extensions and concluding remarks appear in Section 6.

2 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 used starting with the adversary models.

- An *outside adversary* is an adversary that can monitor all the communication links but *not* the contents of the processors memory.

- An *internal dynamic k-listening adversary* (*inside adversary*, in 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, however they still behave according to the protocol. The adversary does not have to choose the k faulty processors in advance. While the adversary corrupts less than k processors the adversary can choose the next processor to be corrupted using the information the adversary gained so far from the processors that are already corrupted.

The following assumptions are used in the first phase of our algorithm which is responsible for the *seeds 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 non participating 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 non participating 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 non participating 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 non participating 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 to each other or anybody else their identity.)

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. 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 a pseudo-random 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 initialization (and termination) procedure 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 pseudo-random sequences. The anonymous seeds distribution is presented following the description of the anonymous communication phase.

3 Anonymous Communication

3.1 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 will chose a distinct seed for each processor. Then the sender can encrypt each bit of information using the seeds of all the processors including its own seeds. Each such seed is used for producing a pseudo-random sequence. The details of the algorithm appear in Figure 1. The symbol $\oplus$ is used to denote the binary xor operation.

Note that the i'th bit produced by the root is a result of xoring twice every of the i'th bits of the pseudo-random sequences except the senders' sequence: once by the sender and then during the 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 non-faulty. This simple scheme requires a single node (the sender) to compute $O(n)$ pseudo-random bits for each bit of data. (We remark that in contrast, our Xor-Tree Algorithm, requires the computation of only $O(k)$ pseudo-random bits to cope with an outside adversary and an internal dynamic k-listening adversary.) The next Lemma state the communication and computation complexities of the algorithm presented in Figure 1.

Lemma 3.1 *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.*
- *The computation overhead of our algorithm is $O(n)$ pseudo-random bits to be computed by each processor per each bit of data.*

Seeds Distribution — Assign (anonymously) distinct seed to each processor. In addition assign the receiver(s) with the sender seed and the sender with every assigned seed.

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, and let b' be the i'th bit of the pseudo random sequence obtained from the seed of P. The i'th bit P sends to its parent (if any) is $b_1 \oplus b_2 \oplus \cdots b_l \oplus b'_1 \oplus b'_2 \cdots \oplus b'_n \oplus d_i$, where $b'_1, b'_2, \cdots, b'_n$ are the i'th bits of the pseudo random sequence obtained from the seeds of all the processors.

P **is not the sender** — The i'th bit that P communicants to its parent (if any) is $b_1 \oplus b_2 \oplus \cdots b_l \oplus b'$.

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 the use of the senders' seed.

Figure 1: $O(n)$ Computation Steps Algorithm.

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 pseudo random bits in every time unit, namely $O(n)$ pseudo-random bits in every time units. ■

3.2 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 seeds distribution decisions of the sender we use $k+1$ *levels* each consists of two *layers* of seeds.

The seeds distribution procedure appears in Figure 2. An example for the choice of seeds for the processors appears in Figure 3.

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

- Each seed is shared by exactly two processors.

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, $1 \leq i < n$, receives the seeds s_i^1 and s_{i+1}^1. P_n receives s_n^1 and s_1^1.

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 $\mathcal{L}_2^l = s_{l+1}^l, s_{l+2}^l, \cdots, s_n^l, s_1^l, \cdots, s_l^l$; $\mathcal{L}_2^l$ is obtained by rotating $\mathcal{L}^l$ l times. P_i $1 \leq i \leq n-l$ receives the seeds s_i^l and s_{i+l}^l and P_j $n-l < j \leq n$ receives the seeds s_j^l and $s_{j-(n-l)}^l$.
Thus, at the end of this procedure every processor is assigned by $2k+2$ distinct seeds.

Figure 2: The choice of seeds.

Seeds of	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9
	s_9	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
Level 1	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9
	s'_8	s'_9	s'_1	s'_2	s'_3	s'_4	s'_5	s'_6	s'_7
Level 2	s'_1	s'_2	s'_3	s'_4	s'_5	s'_6	s'_7	s'_8	s'_9
	s''_7	s''_8	s''_9	s''_1	s''_2	s''_3	s''_4	s''_5	s''_6
Level 3	s''_1	s''_2	s''_3	s''_4	s''_5	s''_6	s''_7	s''_8	s''_9

Figure 3: An example for the distribution of seeds, where $n = 9$ and $k = 2$.

- For every processor P, P shares a (distinct) seed with every of the $k+1$ processors that immediately follow P, (if there are at least such $k+1$ processors), or with the rest of the processors including P_n, otherwise.

3.3 The Xor-Tree Algorithm

Here, we present out main algorithm, the Xor-Tree Algorithm. The Xor-Tree Algorithm appears in Figure 4.

3.4 An abstract game

In this subsection we describe an abstract game that will serve us in analyzing and proving the correctness of the Xor-Tree Algorithm presented in the previous subsection.

Seeds Distribution — Assign seeds to the processors as described in Figure 2. Assign the sender with one additional seed. Assign the receiver(s) with this additional seed of the sender.

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 $b'_1, b'_2, \cdots, b'_{2k+2}$ be the i'th bits of the pseudo-random sequences obtained from the seeds of P, and let b'_{2k+3} be the i'th bit of the pseudo-random sequence obtained from the additional seed of the sender. The i'th bit P sends to its parent (if any) is $b_1 \oplus b_2 \oplus \cdots b_l \oplus b'_1 \oplus b'_2 \cdots \oplus b'_{2k+2} \oplus b'_{2k+3} \oplus d_i$.

P **is not the sender** — The i'th bit that P communicants 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, to obtain the clear-text, by the use of the senders' additional seed.

Figure 4: The Xor-Tree Algorithm.

The adversary get 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[2].

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 non-faulty processor P. We order the processors by their index in a cyclic fashion such that the processor that follows the i'th processor, $i \neq n$, is the processor with the index $i+1$ and the processor that follows the n'th processor is the first processor. Then we assign a new index for each processor such that the sender has the index one, the processor that follows the sender has the index two and so on and so forth. 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 6. Recall that with overwhelming probability every two processors share at most one seed.

Note that by our special seeds selection, described in Figure 6, the special seeds are not known to the k faulty processors.

[2]If the adversary can predict who is the sender then we can use this adversary to break a pseudo-random generator.

Seeds Assignment — Assign seeds to the processors as described in Figure 2. 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.

Figure 5: The Abstract Game.

Theorem 3.2 *In the abstract game any of the $(n-k)$ non-faulty processors is equally likely to be the sender for any poly-bounded internal k-listening adversary.*

Proof: We prove that the i'th bit produced by any non-faulty processors is equally likely to be 0 or 1 for any poly-bounded adversary. Let P be the first non-faulty processor that follows the sender (P is among the first $k+1$ processors that follow the sender). Let b be the special seed of the sender that is shared only with (the non-faulty processor) P. The i'th bit that the sender outputs is a result of a xor operation with the i'th bit of the pseudo-random sequence (among other pseudo-random sequences) obtained from b. Thus, it is equally likely to be 0 or 1 for any poly-bounded internal k-listening adversary. A similar argument hold for the output of P and in general the output of every non-faulty processor. The same argument holds if *any* of the $n-k$ non-faulty processors is the sender. Thus, for any polynomially-bounded k-internal and external adversary, the distribution of the output is indistinguishable of the identity of the sender. ■

3.5 Reduction to the abstract game

In this subsection we prove that if there is an algorithm that reveals information on the identity of the sender in the tree then there exists an algorithm that reveals information on the identity of the sender in the abstract game. The above reduction together with Theorem 3.2 yields the proof of correctness for the Xor-Tree algorithm.

Assume that the adversary reveals information on the sender in a tree $\mathcal{T}$ of n processors. Then an abstract game of n nodes is mapped to the tree as follows:

Theorem 3.3 *In the Xor-Tree Algorithm any of the $(n-k)$ non-faulty 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 on the identity of the sender in a tree $\mathcal{T}$ then there exists an abstract game with the same number of processors and 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 processors P that is not among the $k+1$ last processors — If P is not among the $k+1$ last processors then P is assigned by $2k+2$ seeds $k+1$ seeds of these seeds are from the first layers of the $k+1$ seed levels. These $k+1$ seeds are *new* — they do not appear in any processor prior to 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 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 processors 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 the one in the $k+1$'th level). Fortunately, Q shares a single seed with every of the last processors. This fact allows us to continue the special seed selection procedure, by choosing the seed shared with the next non-faulty processor.

Figure 6: Choice of special seeds.

same seeds distribution, such that the application of the reduction in Figure 7 yields the communication pattern on $\mathcal{T}$ and reveals information on the sender identity in the abstract game. This contradicts Theorem 3.2 and thus contradicts the existence of $\mathcal{A}$. ■

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

Lemma 3.4 *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 $O(k)$ pseudo-random 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 send 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 $2k+3$ pseudo-random bits. ■

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 *hight* 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 hight 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 hight h we use these computed outputs to compute the outputs of processors in hight $h+1$. Let Q be a processor in hight $h+1$, 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 $b_1 \oplus b_2 \oplus \cdots b_l \oplus b_Q$.

Figure 7: The Reduction.

4 Anonymous Seeds Transmission

We first outline the main ideas in the seeds transmission scheme and then give full details. Every processor has a public-key encryption, known to all other processors. A virtual ring defined by the Euler tour on the tree is used for the seeds transmission. Note that the indices of the processor used in this description 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 [6] 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 non-dummy arriving messages but not their origin. In case more than one non-dummy message reaches P_1, a standard back-off algorithm is initiated by P_1. Once exactly one message (containing a collection of seeds) arrives to 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. Notice that this process is quadratic in the size of the ring, the number of colluding processors k, and the length of the security parameter, (i.e., let g be a security parameter and k as before, then we send $O((gkn)^2)$ bits per edge.) Thus, as long the message size p to be broadcasted is greater than $O((gkn)^2)$ we achieve $O(1)$ overall amortized cost per edge, and otherwise we get $O((gkn)^2/p)$ amortized cost.

The details follow. The seeds transmission procedure uses a virtual ring $\mathcal{R}$ defined by an 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 seeds 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 anti-clockwise 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 anti-clockwise order i.e. $\mathcal{L}_1$ in reversed 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 seeds transmission to P_1 appears in Figure 8.

As we prove in the sequel no information concerning the identity of the requesting processors is revealed during the anonymous seeds transmission to P_1 except the information that can be concluded by the value of n_t — the number of processors that would like to transmit. Once $n_t = 1$ the processors starts sending messages to P_2 in a fashion similar to the one used to send seeds to P_1. Then processors sends seeds to P_3 and so on and so forth, till 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 seeds distribution procedure every processor holds the seeds distributed by the sender.

Lemma 4.1 *A coalition of k colluding processors cannot reveal the identity of the seeds distributers.*

Proof: We prove the lemma for the transmission of the seeds from the sender to P_1. Note that one of the last $k+1$ real processors must be non-faulty. If P_1 is non-faulty then no information concerning the identity of the seeds distributers is revealed to the adversary. Otherwise, when P_1 is faulty then let P_i be the non faulty 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 order 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 $\{m_n^{i-1}, m_{n-1}^{i-1}, \cdots m_i^{i-1}\}$. ■

5 Initialization and Termination

When the seed distribution procedure is over, then 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 non-leaf processor receives the i'th 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 non leaf processor waits for receiving the i'th bit from each of its children, uses these bits and its seeds to compute its own i'th bit and sends the output to its parent. Note that buffers can be used in case the processors are not completely synchronized.

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 seeds transmission.

6 Extensions and Concluding Remarks

Our treatment so far considered the anonymous sender case, which is also anonymous to the non participating processors. A simple modification of the algorithm can support the anonymous receiver case: The receiver plays a 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.

To achieve anonymous communication in which both the sender and the receiver are anonymous, do the following: The two participants, P and Q, that would like to communicate (each) send anonymously distinct seeds to $P_1, \ldots, P_{k+1}$ (again, if there are several parties that are competing, a back-off mechanism is used). P_1 will encrypt and broadcast the two seeds it got, each seed encrypted (using distinct intervals of the pseudo-random 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. The same procedure continues for P_2, P_3, $P_4 \cdots P_{k+1}$. At this stage 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 send to P. They both act as senders using the set of $k+1$ seeds as the special seed known to the receiver. The back-off mechanism ensures that one of them starts the communication and then the other can replay (when the first allows him to, i.e., stops transmitting data). We remark that it is possible to have more than two participants by a similar scheme.

Acknowledgment: We thank Oded Goldreich and Ron Rivest for helpful remarks.

References

[1] M. Blum and S. Goldwasser, "An efficient probabilistic public-key encryption scheme which hides all partial information", *CRYPTO 84*.

[2] M. Blum, and S. Micali "How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits", FOCS 82 and *SIAM J. on Computing,* Vol 13, 1984, pp. 850–864.

[3] M. Ben-or, S. Goldwasser, and A. Wigderson, "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation", *STOC 88*.

[4] R. Canetti, U. Feige, O. Goldreich, and M. Naor, "Adaptively Secure Multi-Party Computation", *STOC 96*.

[5] R. Canetti, E. Kushilevitz, R. Ostrovsky, and A. Rosén, "Randomness vs. Fault-Tolerance", *PODC 97*.

[6] D. Chaum, "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms", *Communication of the ACM*, vol. 24, no. 2 (1981), pp. 84-88.

[7] D. Chaum, C. Crépeau, and I. Damgård, "Multiparty Unconditionally Secure Protocols", *STOC 88*.

[8] D. Chaum, "The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability", *Journal of Cryptology*, vol. 1 (1988), pp. 65-75.

[9] D. Chaum, "Achieving Electronic Privacy", *Scientific American*, vol. 267, no. 2 (1992), pp. 96-101.

[10] M. Franklin, Z. Galil and M. Yung "Eavesdropping Games: A Graph-Theoretic Approach to Privacy in Distributed Systems," *FOCS 93*.

[11] O. Goldreich, S. Micali and A. Wigderson, "How To Play Any Mental Game", *STOC 87*.

[12] J. Hastad, "Pseudo-Random Generators under Uniform Assumptions", *STOC 90*.

[13] R. Impagliazzo, L. Levin, and M. Luby "Pseudo-Random Generation from One-Way Functions," *STOC* 89.

[14] E. Kushilevitz, S. Micali, and R. Ostrovsky, "Reducibility and Completeness in Multi-Party Private Computations", *FOCS 94*.

[15] A. Pfitzmann, "How to Implement ISDNs Without User Observability — Some Remarks", TR 14/85, Fakultat fur Informatik, Universitat Karlsruhe, 1985.

[16] A. Pfitzmann, M. Waidner, "Network without User Observability," *Computer & Security* 6 (1987) 158-166.

[17] A. Pfitzmann, B. Pfitzmann and M. Waidner, "ISDN-MIXes — Untraceable Communication with Very Small Bandwidth Overhead," *Proc. Kommunikation in verteilten Systemen* (1991), pp. 451-463.

[18] P. F. Syverson, D. M. Goldsclag, M, G. Reed, "Anonymous Connections and Onion Routing" *Proc. of the Symposium on Security and Privacy* 1997.

[19] M. Waidner and B. Pfitzmann, "The Dining Cryptographers in the Disco: Unconditional Sender and Recipient Untraceability with Computationally Secure Serviceability *Eurocrypt 89.*

[20] C. Rackoff and D. Simon, "Cryptographic Defense Against Traffic Analysis", *STOC 93*

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 $n_t = 1$.

Figure 8: Anonymous Seeds Transmission to P_1.