Communication Complexity of Secure Distributed Computation in the Presence of Noise

Eytan H. Modiano and Anthony Ephremides, *Fellow, IEEE*

Abstract—**A simple model of distributed computation that requires information exchange over a noisy channel is considered. A communication protocol is utilized that requires alternate bit exchanges between two processors. Interest in determining the communication complexity of this exchange is also shown. First, the case of a single public channel is considered and the number of bits that need to be exchanged between the processors to permit δ-accuracy in their goal is computed. For this computation, an error-detection-and-retransmission mechanism of error control, as well as an error-correction-and-retransmission mixture that are consistent with the logical protocol that governs this exchange are considered. Second, the case of the availability of an additional secret channel is considered and interest in determining the minimum number of bits that need to be exchanged over a secret channel in order to maintain ϵ-uncertainty about the computation for an eavesdropper on the public channel is shown. Various subcases under this case are considered and an upper bound on the number of secret bits when no error-control scheme is used is obtained.**

Index Terms—**Distributed computation, communication complexity, secrecy, error control, protocol security.**

I. Introduction

CONSIDER two processors P_x and P_y; each processor has some information at its possession and they wish to distributively compute the value of a binary function of their information. Towards that end, they exchange some information over a channel. The complexity of this exchange was first studied by Yao in [1]. In [2], Orlitsky and El Gamal consider the presence of an eavesdropper who is listening to the above exchange in order to obtain the value of the function. P_x and P_y wish to keep the value of the function ϵ-secure, namely, that the intruder's probability of guessing the value of the function before the exchange takes place is ϵ-close to that probability after the exchange.

To do so they exchange some of their information over a costly secure channel, to which the eavesdropper has no access. The remaining part of the information is exchanged over a less costly public channel, which the eavesdropper can of course access. Since the secure channel is more costly than the public channel, the objective is to transmit the smallest number of bits over the secure channel while maintaining the ϵ-security of the function. Orlitsky and El Gamal present a protocol in [2] that requires no more than $2\log(1/\epsilon) + 16$ secure bits in order to keep any binary function ϵ-secure. In this paper we study the effects of noise on this protocol. The presence of noise in the channel is interesting because it affects all of the parties involved; it affects P_x and P_y by introducing the possibility of an error in the computation of f; it affects the intruder by increasing his uncertainty about the communication. We consider both aspects of the noise effect. First we consider the complexity of the protocol and show that, in the presence of noise, fewer secure bits are required to keep the function secret. We go on to consider the effect of noise on the reliability of the protocol and propose two error-control schemes that can be used along with the previous protocol, to guarantee reliable communication between the two processors.

A. Problem Definition

Let $\mathcal{X}$ and $\mathcal{Y}$ be finite sets, X and Y random variables uniformly distributed over $\mathcal{X}$ and $\mathcal{Y}$ respectively. Let f be a function from $\mathcal{X} \times \mathcal{Y}$ to $\{0, 1\}$. Processor P_x knows the value of X and P_y knows the value of Y. P_x and P_y communicate according to a predetermined protocol in order to exchange their values for the purpose of computing f. An eavesdropper, who knows both their protocol and the function f, listens to their communication in order to obtain information about $f(X, Y)$. Processors P_x and P_y want to make sure that for every value (x, y) of (X, Y) the eavesdropper's probabilities of $\{f(X, Y) = 1\}$ before and after the communication takes place, are ϵ-close. Therefore they transmit some of the bits over a secure channel. The problem is to find a protocol that minimizes the worst case number of bits needed to be transmitted securely.

The two processors use a deterministic protocol P. For every $(x, y) \in \mathcal{X} \times \mathcal{Y}$, P specifies the following sequence:

$$\langle T_i, B_i, S_i \rangle, \quad i = 1, \cdots, N,$$

where T_i describes the originator of the ith bit (P_x or P_y), B_i describes the bit itself (0 or 1), S_i denotes the channel used (Secure or Public), and N is the total number of bits communicated. An eavesdropper who knows the originator of each bit but can decode only the publicly communicated

Manuscript received April 26, 1990; revised December 19, 1991. This work was supported by the Naval Research Laboratory Grant No. N00014-91-J2003. This work was presented in part at the IEEE International Symposium on Information Theory, San Diego, CA, January 14–19, 1990.

The authors are with the Electrical Engineering Department, University of Maryland, College Park, MD 20742.

IEEE Log Number 9108037.

bits constructs the modified sequence: $\langle T_i, B_i', S_i\rangle$, $i = 1, \cdots, N$, where

$$B_i' = \begin{cases} B_i, & \text{if } S_i = \text{public}, \\ 0, & \text{if } S_i = \text{secure}. \end{cases}$$

(Note that, if S_i = secure, the eavesdropper will have to guess the value of B_i. For simplicity let us assume that the eavesdropper always guesses $B_i = 0$.) Denote this modified sequence by $I(x, y)$.

A protocol is said to be ϵ-secure if for all transmission sequences $e = \langle (T_i, B_i, S_i)\rangle_{i=1}^{N}$ that have nonzero probability,

$$\left| P\{f(X, Y) = 1 | I(x, y)\} - P\{f(X, Y) = 1\} \right| \le \epsilon. \tag{1.1}$$

This insures that the eavesdropper's *a priori* and *a posteriori* probabilities of $f(X, Y) = 1$ are ϵ-close. Let $C_p(x, y)$ be the number of bits communicated securely under a protocol P when $X = x$ and $Y = y$. Then the complexity of P is defined as

$$C_p = \max C_p(x, y),$$

where the maximum is taken over all possible values x, y of X and Y, and the ϵ-complexity of f is defined as

$$C(f, \epsilon) = \min \{C_p \mid P \text{ is an } \epsilon\text{-secure protocol for } f\}.$$

The problem is to determine $C(f, \epsilon)$.

II. Upper Bound on Complexity

We start by invoking the main result obtained in [2].

Theorem 1 [2]: For all n, f: $[1, \cdots, n] \times [1, \cdots, n] \to [0, 1]$ and $\epsilon > 0$,

$$C(f, \epsilon) \le 2 \log (1/\epsilon) + 16.$$

The proof of Theorem 1 is based on the presentation of a protocol with the above complexity. In what follows, we describe the protocol and give a brief outline of its proof. We omit the parts of the proof that are not essential to this paper and can be referenced in [2].

The protocol is based on a decomposition tree for the function table of f. A function table, A, for f is a two-dimensional array in which the entry at the (x, y) coordinate position is the value of $f(x, y)$. Any such function table can be partitioned along its x-coordinate (row-partition) or y-coordinate (column-partition). A decomposition tree for the function table, A, is a binary tree whose nodes at a given depth represent a partition of A, i.e., they are disjoint subarrays of A such that their union is A. Each internal node is the union of its children. The root of the tree is A. The leaves of the tree are singleton elements of A (i.e., subarrays of A whose x, y coordinates consist of single elements $\{x\}, \{y\}$). Since the tree is binary, each node can be either column partitioned or row partitioned by its children. Also, there is a 1–1 correspondence between values of $f(x, y)$ and leaves of the tree specified by single elements $\{x, y\}$.

A. The Protocol

Given a decomposition tree for A, we label the children of each node by "0" and "1" in an arbitrary way and we proceed with the following rule: At each step of the communication, P_x and P_y consider one node of the tree starting at the root. If the node is row partitioned by its children, then P_x transmits the label of the child whose x-coordinate contains x. If the node is column partitioned by its children then P_y transmits the label of the child whose y-coordinate contains y. The process terminates when they arrive at a leaf, and the values of X and Y are obtained. They exchange their information over a public channel until they arrive at a subtree of height $\le 2 \log (1/\epsilon) + 16$. In this way, they assist each other in searching the tree up to that height. Following that, they transmit the remaining bits over a secure channel.

Clearly, the previous protocol requires no more than $2 \log (1/\epsilon) + 16$ secure bits. In order to prove Theorem 1, we must show that the eavesdropper's probability of guessing the value of f given the publicly transmitted information is within ϵ of his prior probability of guessing the value of f.

B. Outline of Proof

The proof of Theorem 1 is based on proper choice for a partition of A, the function table of f. A method for partitioning A was introduced in [2] which is used to form the decomposition tree for A as follows: Call A a table of level 0. Recursively partition each table of level i into subtables according to the method described in [2]. Call these subtables, tables of level $i + 1$. Continue in this fashion until the decomposition tree for A is formed. Processors P_x and P_y communicate over a public channel until they both know which subtree of height $\lfloor 2 \log (1/\epsilon) + 16 \rfloor$ contains their sequences. They then use a private channel to transmit their indices in this tree. In order to prove that the protocol is ϵ-secure, we need to show that each subtree of height $i \ge 2 \log (1/\epsilon) + 16$ has a proportion of 1's which is within ϵ of the original proportion of 1's in A. This can be done by showing that if A_i is a tree of height i with S_i 1's and a_i total number of elements, then

$$S - \epsilon \le \frac{S_i}{a_i} \le S + \epsilon, \tag{2.1}$$

where $S = P(f(X, Y) = 1)$ is the proportion of 1's in A. This is shown in [2] and completes the proof of the theorem. □

The proof of Theorem 1 may be better understood via the use of Fig. 1. As shown in the figure, every subtree of height greater than $2 \log (1/\epsilon) + 16$ corresponds to a subtree of A with a proportion of 1's which is within ϵ of S. Therefore, disclosure of the identity of the subtree containing X and Y does not violate the ϵ-security requirement. Also note that this protocol permits the two processors to compute the value of f and it also provides each one of them with the exact value of the variable that the other processor possesses. If the objective is to merely compute the value of f and not necessarily to perform the full exchange of the X, Y values,

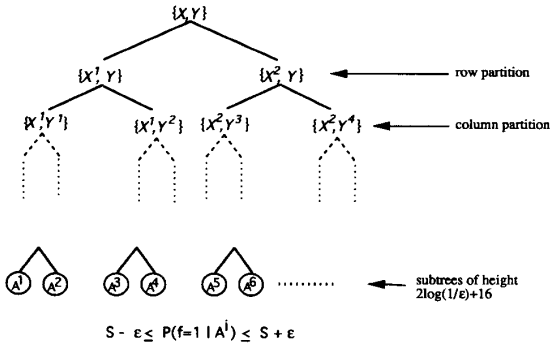

Fig. 1. Decomposition tree with subtrees of height $\lceil 2\ log\ (1/\epsilon) + 16 \rceil$.

then a different problem arises that has been studied in part by Yao [1] and Ja'Ja' [3].

Let us recapitulate some important properties of the protocol that will be useful later on.

1) Every subtree of height $\geq 2\log(1/\epsilon) + 16$ corresponds to a subtable of the function table for f with a proportion of 1's which is within ϵ of S. In other words, for all subtrees of height $i \geq 2\log(1/\epsilon) + 16$ with S_i 1's and a_i elements, the following holds,

$$S - \epsilon \leq \frac{S_i}{a_i} \leq S + \epsilon.$$

2) Since P_x and P_y exchange their respective locations in the decomposition tree for f, each bit sent is a function of previous bits sent and received. In other words if we let $x_1, \cdots, x_i, \cdots$ denote P_x's transmission sequence, and $y_1, \cdots, y_i, \cdots$ denote P_y's transmission sequence, then

$$x_i = F_d(y_1, \cdots, y_{i-1}, x_1, \cdots, x_{i-1}),$$

where F_d depends on the decomposition tree for f.

The protocol previously described can be used to secretly compute the value of f using a constant number of secure bits. If P_x and P_y communicate over a noiseless channel and make proper use of the protocol, they will always arrive at the correct value of f. However, if either the private or the public channels is noisy, then the exchange will no longer be error-free and the processors may end up with the wrong value of f. At the same time, the eavesdropper listening to this exchange will also receive a noisy version of the exchange and his uncertainty regarding the value of f may increase. Consequently, P_x and P_y may need to exchange fewer secure bits in order to keep f ϵ-secret. So, the presence of noise introduces a trade-off that may result in a need for fewer secure bits. In what follows, we examine the various aspects of this trade-off. First, in Section III, we consider the effects of noise on the eavesdropper's uncertainty, and the number of bits that must be exchanged over the secure channel. Then, in Section IV, we develop and analyze two error-control schemes which can be used to maintain the reliability of the protocol.

III. Communication in a Noisy Environment

In this section, we consider the effect of noise on the protocol described in Section II, from the point of view of the eavesdropper. Namely, we wish to determine the effect of the noise on the number of bits that need to be exchanged over the secure channel in order to keep f ϵ-secure. The protocol assumes the existence of two channels, a private channel and a secure channel. We can assume that either, or both, are noisy. However, here we are strictly concerned with the effect of the noise on the eavesdropper's uncertainty and, since the eavesdropper has no access to the private channel, it is reasonable to assume that only the public channel is noisy.

Consider the protocol described in Section II; according to that protocol P_x and P_y partition the decomposition tree for the function table of f in order to arrive at the values of X and Y. They exchange partitioning information publicly until they arrive at a subtree of height less than $2\log(1/\epsilon) + 16$, at which point they exchange the remaining information over a secure channel. It was shown in [2] that all subtrees of height $2\log(1/\epsilon) + 16$ or greater, have a proportion of 1's which is within ϵ of S. An intruder listening to the conversation will only be able to identify the subtree corresponding to the publicly communicated bits, and therefore will have a probability of $f(X, Y) = 1$ between $S - \epsilon$ and $S + \epsilon$. Consequently, P_x and P_y need only transmit $2\log(1/\epsilon) + 16$ bits securely in order to satisfy (1.1).

According to this protocol, the total number of information bits exchanged between P_x and P_y is

$$\text{Total Number of bits} = \lceil \log |X| + \log |Y| \rceil = 2\log(n).$$

For simplicity, we take n to be a power of 2. It can be easily shown that the results of this section hold for all values of n. The total number of bits that are transmitted over the public channel is given by $K = 2\log(n) - 2\log(1/\epsilon) - 16 = 2\log(n\epsilon/256)$.

Now we consider the effect of the noise. We assume that the public channel is modeled by a binary symmetric channel model. There are 2^K possible error patterns of length K that can occur. Each error pattern will result in a different subtree being eventually identified erroneously. We enumerate these error patterns in order of increasing weight, that is, we let E_0 denote the all-zero-error-pattern; then we let $E_1, E_2, \cdots, E_K$ denote those error-patterns that have weight 1; following that we let $E_{K+1}, \cdots, E_{K+\binom{K}{2}}$ denote those error patterns that have weight 2; we continue in this fashion the enumeration. As a result we have $\text{wt}(E_i) \leq \text{wt}(E_{i+1})$ for all i.

Let P_i = the probability of error pattern E_i; clearly (assuming probability of channel error is at most 0.5),

$$P_0 \geq P_1 \geq P_2 \geq \cdots \geq P_{2^K - 1}.$$

As indicated earlier, each error pattern identifies a subtree of the decomposition tree for the function table of f with a proportion of 1's which is between $S - \epsilon$ and $S + \epsilon$. If S_i is the proportion of 1's in the ith subtree, it satisfies

$$S - \epsilon \leq S_i \leq S + \epsilon. \tag{3.1}$$

Also, we have

$$\sum_{i=0}^{2^{K-1}} \frac{S_i}{2^K} = S. \tag{3.2}$$

Let, α denote the intruder's probability of $f = 1$; clearly

$$\alpha = \sum_{i=0}^{2^{K-1}} P_i S_i, \tag{3.3}$$

where S_i is the proportion of 1's in the subtree corresponding to P_i. Also note that n is a power of two and, therefore, all subtrees are of equal size. In the following lemma, we present upper and lower bounds on alpha by choosing the S_i's that correspond to a worst case f, namely,

$$S_i = \begin{cases} S + \epsilon, & \text{for } i \in [0, 2^{K-1} - 1], \\ S - \epsilon, & \text{otherwise.} \end{cases} \tag{3.4}$$

Lemma 1:

$$\sum_{i=0}^{2^{K-1}-1} P_i(S - \epsilon) + \sum_{i=2^{K-1}}^{2^K-1} P_i(S + \epsilon) \le \alpha \le \sum_{i=0}^{2^{K-1}-1} P_i(S + \epsilon) + \sum_{i=2^{K-1}}^{2^K-1} P_i(S - \epsilon).$$

Proof: Suppose there exists a function with corresponding S_i's that differ from those in (3.4) and that result in a larger value of α. Then there must exist an i between 0 and $2^{K-1} - 1$ such that $S_i = S + \epsilon - \epsilon_1$ for some ϵ_1 greater than 0. In order to satisfy (3.1) and (3.2), there must also exist a j between 2^{K-1} and $2^K - 1$ such that $S_j \ge S - \epsilon + \epsilon_2$, where $0 < \epsilon_2 < \epsilon_1$, but then, if we subtract ϵ_2 from S_j and add ϵ_2 to S_i, the value of α is increased and both (3.1) and (3.2) are still satisfied. Thus, we have a contradiction, and the assignment of (3.4) results in the maximum value of α. A similar proof exists for the lower bound. We complete the proof of the lemma by combining equations (3.3) and (3.4).

Let now

$$\alpha_1 = \sum_{i=0}^{2^{K-1}-1} P_i.$$

We can rewrite the inequality of Lemma 1 as

$$\alpha_1(S - \epsilon) + (1 - \alpha_1)(S + \epsilon) \le \alpha \le \alpha_1(S + \epsilon) + (1 - \alpha_1)(S - \epsilon), \text{ or}$$

$$|\alpha - S| \le (2\alpha_1 - 1)\epsilon. \tag{3.5}$$

Theorem 2: In order to keep the communication ϵ'-secure we need only transmit

$$2 \log \left(\frac{2\alpha_1 - 1}{\epsilon'} \right) + 16 \text{ bits securely.}$$

Proof: Let $\epsilon = \epsilon'/(2\alpha_1 - 1)$, then according to (3.5), $|\alpha - S| \le \epsilon'$, which by definition implies that by using the protocol discussed in Section II we achieve communication that is ϵ'-secure.

Theorem 2 states that fewer secure bits are needed to keep f ϵ-secure, if noise is present. The savings in the number of secure bits, of course, comes at a cost of incurring some probability of error in the exchange. This leads to an alternative interpretation of Theorem 2. As long as P_x and P_y are willing to tolerate errors in their exchange, they can reduce the number of secure bits by adding noise to the channel. In fact, they can add as much noise as they are willing to tolerate. So, suppose they are willing to tolerate δ error in their exchange. Then the probability of error in the exchange is the probability that at least one of the publicly communicated bits is in error. If $N = 2 \log(n)$ and p equals the resulting channel crossover probability, the number of publicly communicated bits will be less than N and $P(\text{error}) \le 1 - (1 - p)^N < \delta$, which yields $p \le 1 - (1 - \delta)^{1/N}$. This means that in order to maintain the probability of error in the exchange below δ it suffices that the channel error probability be kept below the value previously indicated.

IV. Error-Control Coding for Distributed Computation

We now turn our attention to the effect of the noise on the quality of the communication between P_x and P_y. We will again assume that they employ the protocol described in Section II, in order to exchange their values of X and Y. They communicate over a noisy binary symmetric channel, and wish to limit the probability of error in their exchange to level δ. Therefore, they will employ some error control scheme that will introduce redundant bits into the exchange. The problem is to find an error-control algorithm that will minimize the total number of redundant bits.

Recall that under the protocol presented in Section II, P_x and P_y distributively compute the value of f as follows: Given a decomposition tree for f, P_x and P_y consider one node of the tree starting at the root. If the node is row-partitioned, P_x transmits the label of the child whose coordinate contains the value of X (similarly, if the node is column-partitioned, P_y transmits the label of the child whose coordinate contains the value of Y). The communication terminates when they arrive at a leaf where the values of X and Y can be obtained. Clearly, if an error occurred in the transmission of any of the labels, the wrong subtree would be identified and as a result the subsequent transmissions become meaningless. Therefore, to avoid such logical errors, a transmission error must be identified and corrected before the transmission of the next bit.

In what follows we present two coding schemes that are built to satisfy the previous requirements. The first scheme is based on error-detection techniques, while the second scheme uses error-correction techniques. In both cases, we search among commonly used classes of error-control codes to identify the ones that result in smallest number of redundant bit transmissions.

A. Error-Detection Scheme

As we have already established, an error-control algorithm for our protocol P cannot involve error correction on blocks of size greater than one because such a scheme cannot correct

logical errors. In order to correct logical errors, upon detection of an error all digits from the point of the first error must be retransmitted. We, therefore, suggest the following algorithm: Each processor needs to transmit a total number of information bits N; it proceeds as before, according to the protocol P, except that after transmitting a block of k bits it transmits additional C check bits. Recall that these k bits are transmitted one at a time, alternating with the k bits that the other processor transmits back in pursuit of the tree search that the protocol P implements. Also, as a consequence of Remark 2, we require that the check bits follow the information bits.

Upon receiving the $k + C$ bits, both P_x and P_y check them for errors; if an error is detected by either P_x or P_y, the entire sequence of $2(k + C)$ bits is retransmitted.[1] The two processors repeat the process and continue to retransmit the same block until no errors are detected, at which point they proceed to the next sequence of k bits. Thus, no logical errors occur unless undetected bit errors occurred. Clearly, it is of interest to select the value of k to minimize the total number of transmitted bits. We proceed to analyze the performance of various classes of codes with respect to this error-control scheme.

1) Analysis of Error-Detecting Scheme: Let N, k, C be defined as before. Let

- U be the probability of at least one undetected bit error occurring in the sequence of $2(k + C)$ bits, and no detected bit errors occurring;
- D be the probability of at least one detected error occurring in a sequence of $2(k + C)$ bits.

Notice that U and D are not conditional probabilities, but rather the joint probabilities of an error occurring and being undetected or detected respectively. Therefore, $1\text{-}D\text{-}U$ = Prob(zero errors in the block) = $(1 - p)^{2(k+C)}$. Also note that, in effect, D represents the probability of a block being retransmitted.

Let E be the event of an error in the overall exchange of X and Y.

Let B_i be the event that there is at least one undetected error in the last transmission of sequence i.

Let X_i be the number of transmissions of sequence i.

2) Probability of Error Computation: Let $M = \lceil N/k \rceil$ be the number of distinct sequences or blocks of bits that will need to be exchanged to effect the full transmission of the N information bits (if k does not divide N then the last block will consist of the remainder of the information bits followed by zeros in order to obtain a sequence of k bits). We have

$$P(E) \le P\left(\bigcup_{i=1}^{M} B_i\right) = 1 - P\left(\bigcap_{i=1}^{M} \overline{B}_i\right) = 1 - \prod_{i=1}^{M} P(\overline{B}_i). \tag{4.1}$$

[1]We do not consider here the separate protocol that will be needed to signal that a retransmission rather than a new transmission commences after errors have been detected.

This upper bound results because an error in the Mth block may or may not result in an error in the exchange. Note that $P(B_i) = P(\bigcup_{X_i=1}^{\infty} G_i)$, where G_i is the event that at least one bit error is detected in the first $X_i - 1$ transmissions of sequence i and at least one undetected error and no detected errors occur in the X_ith transmission. Therefore,

$$P(B_i) = \sum_{X_i=1}^{\infty} (D)^{X_i-1}(U) = \frac{U}{1 - D}. \tag{4.2}$$

As noted earlier $P(E) = 1 - \Pi_{i=1}^{M} P(\overline{B}_i)$. Since the B_i's are independent we have

$$P(\overline{E}) = P(\overline{B}_i)^M.$$

$$P(\overline{B}_i) = 1 - \frac{U}{1 - D} = \frac{1 - (D + U)}{1 - D} = \frac{(1 - p)^{2(k+C)}}{(1 - D)}.$$

As a result,

$$P(E) = 1 - \left[\frac{1 - (D + U)}{1 - D}\right]^M = 1 - \frac{(1 - p)^{2(k+C)M}}{(1 - D)^M}. \tag{4.3}$$

This is one performance index that we are interested in keeping small. Next, we compute the total number of bits communicated. Let $n = k + C$ and T be the total number of bits transmitted. We have $E\{T\} = Mn(1 + E\{\text{number of retransmissions per block}\})$. Since the number of retransmissions is geometrically distributed with probability of retransmission D, we have,

$$E\{T\} = \frac{Mn}{1 - D} = \frac{M(k + C)}{1 - D}. \tag{4.4}$$

This is the other performance index that we wish to keep small. The question now remains how to calculate D for a given code. We noted before that

$$D + U + (1 - p)^{2(k+C)} = 1. \tag{4.5}$$

Since the exchange of the $2(k + C)$ bits proceeds alternately bit-by-bit and since each processor uses its own code for only half of the $2(k + C)$ bits that are exchanged, we need to relate the probabilities U and D to those that affect the transmission of $(k + C)$ bits in each direction separately. Thus, let

- u be the probability of an undetected error occurring in the transmission of $(k + C)$ bits from one processor to the other (and, of course, no detected errors occurring);
- d be the probability of a detected error occurring in such a transmission.

As before, we again have the relationship between d and u as follows:

$$d + u + (1 - p)^n = 1. \tag{4.6}$$

Now the event of undetected errors in the overall exchange of the $2(k + C)$ bits while no detected errors occur is the union of events that undetected errors occur in one direction (i.e.,

in the transmission of $(k + C)$ bits) while detected errors do not occur in the opposite direction. Therefore,

$$U = 2u(1 - d) - u^2. \tag{4.7}$$

Now, if G is an (n, k) linear block code, used by either processor (with $n = k + C$), let A_i denote the number of code vectors in G of weight i. Then, the numbers $A_0, A_1, \cdots, A_n$ constitute the weight distribution of G. When G is used for error detection over a binary symmetric channel, the probability of at least one undetected error is given by [4, p.77]

$$u = \sum_{i=1}^{n} A_i p^i (1 - p)^{n-i}. \tag{4.8}$$

Consequently, we can calculate D for a given code by combining equations (4.5)-(4.8). Then we use the calculated value of D in (4.3) and (4.4) to evaluate performance.

3) Performance Evaluation: We performed the computation previously outlined for various codes selected over typical classes of linear block codes (since we require that the check bits follow the information bits, we were restricted to the class of systematic codes; however, since every linear block code can be put in systematic form, this restriction did not limit our choice of codes). The objective was to select a code that requires the minimum overall number of bits exchanged, as expressed in (4.4), while keeping the probability of error, as expressed in (4.3), below a specified value δ. Both performance measures, as expressed in (4.3) and (4.4), depend on the probability of an undetected error for the particular code, which in turn is a function of that code's weight distribution as implied by (4.8). Theoretically, we can compute the weight distribution of an (n, k) code by examining its 2^k codewords. However, for large n and k the computation becomes impractical. The weight distributions of many linear codes are still unknown. Consequently, it is difficult, if not impossible, to perform the exact computation for an arbitrary code. It proved also difficult to find a good bound on u over broad classes of codes (an upper bound exists for the average probability of an undetected error over the ensemble of all (n, k) linear systematic codes. However, such a bound is not necessarily an upper bound on the probability of an undetected error for all linear block codes).

Luckily, the weight distribution of certain useful classes of codes is known, and is easily expressed as a polynomial of degree n. This polynomial is known as the weight enumerator polynomial of the code. Weight enumerator polynomials are available for a variety of codes including such codes as Hamming codes [4, p. 81], BCH codes [5, p. 451], the Golay code [6, p. 121], and a few other less commonly used codes. As a result, we performed the computations for these classes of codes and determined amongst these classes the code that requires the minimum number of bits and satisfies δ-error constraint.

We define the rate R of our error control scheme to be the ratio of the number of information bits to the expected value of the total number of bits transmitted, i.e., $R = N/E[T]$. Given specific values for N, δ, and p, we scan the various

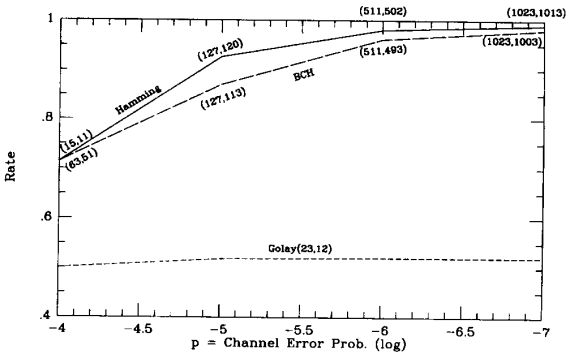

Fig. 2. Rate for the error-detection scheme with $N = 1000$ and $\delta = 10^{-6}$.

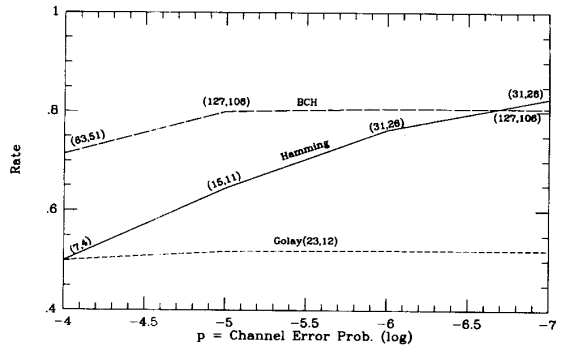

Fig. 3. Rate for the error-detection scheme with $N = 100$ and $\delta = 10^{-8}$.

classes of codes and for each class we find the specific code which minimizes the rate while keeping the probability of error below δ. In other words, for each code that we examine, we use (4.4) and (4.3) to compute the probability of error and the rate R for our scheme when that code is employed. We then select, from each class of codes, the particular code which results in the highest value of R while maintaining a probability of error below δ. In Fig. 2, we plot the rate (R) vs. the channel error probability (p) for $N = 1000$, and $\delta = 10^{-6}$, for various classes of codes. Similarly in Fig. 3, we plot R vs. p for $N = 100$, and $\delta = 10^{-8}$. For each sample value of p, we indicate the specific code that was used to achieve the indicated value of R. It should be noted that since R for our scheme accounts for both check bits as well as retransmissions, the rate R for the scheme can never exceed the rate of the particular code which was employed. As can be seen from the plot, in this particular example, the Hamming codes resulted in the best rate. Although unpredictable, these results are rather simple to understand. Clearly, for $N = 1000$ and $\delta = 10^{-6}$, the Golay code represents a bit of an overkill and provides much more protection from errors than necessary, therefore, it results in a much lower rate. The same holds, to a lesser degree for the BCH codes. The Hamming code is most efficient in the sense that it provides sufficient protection at a lower rate. For $N = 100$ and $\delta = 10^{-8}$, the Golay code still provides too

much protection and, therefore, has a lower rate. However for high values of p the only Hamming codes that proved sufficient protection are the low-rate ones and in a sense they provide more than the necessary protection (they result in actual error rate much lower than δ). On the other hand, the BCH codes provide sufficient protection at a much higher rate. As can be seen from the figure, when p gets very small the power of the BCH codes is no longer necessary and the (31, 26) Hamming code provides sufficient protection at a lower rate. A more complete presentation of the results, for a variety of values of N, δ, and p is available in [7].

B. Error-Correction Scheme

In the previous section, we described an error-control scheme that was based on error-detecting codes. In the event that an error was detected in a given block the entire block was retransmitted. That scheme has as its main drawback the need to retransmit the entire block upon the detection of an error. In this section, we describe a method that is based on error-correcting codes. This method requires retransmission not of the entire block but only from the point of the first detected error. The error-correcting codes serve the purpose of locating the first error in a block. As explained earlier, however, these codes cannot be used for the actual correction of errors because of the iterative nature of the computation that leads to logical errors in the event of a transmission error. Thus, even if the error is corrected, the subsequent, logically wrong, but correctly transmitted bits are useless and lead to an erroneous computation of f.

Let N be, as before, the total number of bits transmitted by each processor. Just as in the previous section P_x and P_y communicate in an iterative manner, a bit at a time. After transmitting k bits each, P_x and P_y transmit additional C bits each as check bits (again, as a consequence of Remark 2, we require that the check bits follow the information bits).

Upon receiving the $k + C$ bits, both P_x and P_y check them for errors. After that they exchange the location of the first detected error. They then accept all of the bits up to the location of the first error and retransmit everything from that location on. The additional protocol for the exchange of information on the error location and on the need to retransmit is again ignored. The processors continue to communicate in blocks of size $k + C$ until all of the N information bits have been accepted. We next quantify the total number of bits transmitted and the probability of error in the exchange.

1) Analysis of Error-Correcting Scheme: Let N, k, C, n, and T be defined as before and let

- Z_i be the number of useful information bits in the ith transmission (a bit is considered useful if it occurred before the location of the first detected error).

We keep transmitting blocks of size $k + C$ until the total number of transmitted useful bits is equal to or exceeds N. It is important to note that each transmission (and retransmission) contains exactly $k + C$ bits. This is so because if an error is detected in the middle of a block we do not just retransmit the remaining bits of the block starting from the error location but rather we jump ahead into what would have been the next block and transmit again a total of k information bits (plus C check bits); that is, we have a sliding window of k information bits the next position of which is determined by the location of the first detected error in the sequence of k information bits contained in the window in its previous position. In the event that fewer than k bits remain to be transmitted, the next window is obtained by transmitting the remainder of the information bits followed by zeros so that again we can have a block of k bits which, of course, is followed by C check bits.

Let J be the total number of block transmissions. Then $T = J^*(k + C)$, $E\{T\} = E\{J\}(k + C)$, and $E\{J\} = \sum_{j=1}^{\infty} jP_j$, where $P_j = P(J = j)$. Notice that P_i is the probability that the first $i - 1$ transmissions produce fewer than N useful bits, while the first i transmissions produce N or more useful bits. We can, therefore, express the P_i's as follows:

$$P_1 = P[Z_1 \geq N],$$

$$P_2 = P[(Z_1 < N) \cap (Z_2 + Z_1 \geq N)],$$

$$\begin{aligned} P_j &= P\left[\left(\sum_{i=1}^{j-1} Z_i < N\right) \cap \left(\sum_{i=1}^{j} Z_i \geq N\right)\right] \\ &= 1 - P\left[\left(\sum_{i=1}^{j-1} Z_i \geq N\right) \cup \left(\sum_{i=1}^{j} Z_i < N\right)\right] \\ &= P\left[\left(\sum_{i=1}^{j-1} Z_i < N\right)\right] - P\left[\left(\sum_{i=1}^{j} Z_i < N\right)\right]. \end{aligned} \quad (4.9)$$

We next consider the random variable Z_i.

2) The Distribution of Z_i: Let X_i be the location of the first error detected by P_x in block i and let Y_i be the location of the first error detected by P_y in block i. Then $Z_i = \min(k, X_i, Y_i)$, because all bits subsequent to the location of the first error detected by either P_x or P_y incorporate logical errors and, hence, are not useful (ignoring the possibility that X_i need not be retransmitted if Y_i is the first error). No more than k bits can be considered as useful since only the first k bits are information bits and the rest are check bits. Thus, if the first located error occurs after the kth information bit, it is simply corrected and nothing gets retransmitted.

Let $Z_i' = \min(X_i, Y_i)$. Then

$$Z_i = \begin{cases} Z_i', & Z_i' < k, \\ k, & Z_i' \geq k. \end{cases}$$

Next, we note that the Z_i's are i.i.d; let us denote their common probability mass function by $f_Z(z)$. Moreover, the X_i's and Y_i's are i.i.d with distribution $f_X = f_Y = f$. Therefore,

$$f_Z(z) = \begin{cases} f_Z'(z), & z < k, \\ \sum_{i=k}^{n} f_Z'(z), & z = k, \end{cases} \quad (4.10)$$

and

$$f_{Z'}(z) = P(Z' = z)$$
$$= \sum_{j=z}^{n} P(X = z)P(Y = j) + \sum_{i=z+1}^{n} P(Y = z)P(X = i)$$
$$= f(z)\left[f(z) + 2\sum_{i=z+1}^{n} f(i)\right]$$
$$= 2f(z)\sum_{i=z}^{n} f(i) - (f(z))^2. \quad (4.11)$$

To compute now the distribution of X, which represents the location of the first error detected in a block, we have

$$f_x(x) = f(x) = P(\text{location of first detected error} = x)$$
$$= \sum_{i=0}^{t} P(X = x \mid i \text{ errors occurred})P(i \text{ errors occurred}) + P(X = x \mid \text{more than } t \text{ errors occurred}) \times P(\text{more than } t \text{ errors occurred}),$$

where t is the maximum number of errors that the code can correct. Hence,

$$f(x) = \sum_{i=0}^{t} P(X = x \mid i \text{ errors occurred}) \times \binom{n}{i} p^i(1-p)^{n-i} + P(X = x \mid \text{more than } t \text{ errors occurred}) \times \sum_{i=t+1}^{n} \binom{n}{i} p^i(1-p)^{n-i}. \quad (4.12)$$

We first compute the probability that the first error location is x, given that $i < t + 1$ errors occurred. Let $(E_1, E_2, \cdots, E_n)$ denote the received error pattern, where $E_i = 1$ denotes an error in the ith bit. When transmitting over a binary symmetric channel all $\binom{n}{i}$ error patterns of weight i are equally likely, each with probability $p^i(1-p)^{n-i}$. All these error patterns are correctable and we know that when fewer than $t + 1$ errors occur the location of the first detected error is the same as the location of the first actual error. Therefore,

$$P(X = x \mid i \text{ errors occurred}) = P(\text{first error occurred in position } x \mid i \text{ errors occurred}) = P(E_1 = 0, \cdots, E_{x-1} = 0, E_x = 1 \mid i \text{ errors occurred}).$$

Since there are $\binom{n-x}{i-1}$ such patterns, we have

$$P(X = x \mid i \text{ occurred}) = \frac{\binom{n-x}{i-1}}{\binom{n}{i}}. \quad (4.13)$$

The location of the first detected error when more than t errors occur is difficult to characterize. When more than t errors occur the decoder decodes them into one of the 2^k codewords, and this process is entirely dependent on the decoder. Therefore, in the previous equation the $P(X = x \mid > t$ errors occurred) cannot be expressed in a general form. However, in order to quantify the total number of bits transmitted, we attempt to use an approximation, namely that the location of the first error is uniformly distributed over the codeword length, i.e.,

$$P(X = x \mid \text{more than } t \text{ errors occurred}) \approx \frac{1}{n}. \quad (4.14)$$

Clearly this is a poor assumption when t is relatively large; however multiple errors have very small probability. We compared this approximation to the actual performance of a number of codes. In particular, we compared its performance with those codes that are used in our analysis. We found that for channel error probabilities of less than 10^{-2} this approximation was within 2% of the actual results. When the channel error probability was less than 10^{-4}, this approximation was within 0.5% of the actual results [7]. Putting together (4.12)–(4.14), we approximate the distribution of X by

$$f(x) \approx \sum_{i=0}^{t} \binom{n-x}{i-1} p^i(1-p)^{n-i} + \left(\frac{1}{n}\right) \sum_{i=t+1}^{n} \binom{n}{i} p^i(1-p)^{n-i}. \quad (4.15)$$

Finally, the performance index $E\{T\}$ can be computed using (4.9)–(4.15).

3) Computation of Probability of Error: Let E be the event of an error in the overall exchange between P_x and P_y. Let B_i be the event that an error occurred before the first detected error in the ith transmission. Then, we have

$$P(E) = \sum_{j=1}^{\infty} P(E \cap (j \text{ blocks transmitted})) = \sum_{j=1}^{\infty} P(E \mid j)P_j, \quad (4.16)$$

where P_j is given by (4.9) and where

$$P(E \mid j \text{ blocks were transmitted}) = P\left[\bigcup_{i=1}^{j} B_i \mid j \text{ blocks transmitted}\right] = 1 - P\left[\bigcap_{i=1}^{j} (\overline{B_i}) \mid j\right].$$

This probability is difficult to compute because of the dependence on j, the number of blocks transmitted. Although it is not obvious, the probability of error in a block is dependent on the number of blocks transmitted; this dependence can be seen by observing that when j is small few errors must have occurred, and when j is large many errors must have occurred. Consequently, $P(B_i)$ is likely to be higher for large values of j. In order to compute the probability of error, one must have the joint distribution of j and B_i. This joint distribution is a function of the code and the

decoder used, and, therefore, cannot be computed for the ensemble of all codes. In fact, even for a given code and decoder, it is still not clear how to compute the joint distribution of j and B_i. As a result, we must again resort to an approximation of this distribution.

Let us (incorrectly) assume that B_i is independent of j; then (4.16) can be approximated by

$$P(E \mid j) \approx 1 - \prod_{i=1}^{j} P(\overline{B}_i) = 1 - P(\overline{B}_i)^j. \quad (4.17)$$

We next find an upper bound on $P(B_i)$, the probability that an error occurs before the location of the first detected error in the ith transmission. When using a t error-correcting code, all errors of weight t or less are detected; therefore, the only way for an error to occur before the first detected error is for more than t errors to occur in either P_x's or P_y's transmission. Even if more than t errors occur, it is possible that the location of the first detected error is (erroneously) before the location of the first actual error. However, $P(B_i)$ can be bounded above by the probability that either P_x or P_y sustain more than t errors. Let t_x be the event that P_x sustains more than t errors, and let t_y be the event that P_y sustains more than t errors. Then,

$$P(B_i) \le P(t_x \cup t_y) = 1 - P(\overline{t_x} \cap \overline{t_y}) = 1 - P(\overline{t_x})P(\overline{t_y}).$$

Let α be the probability that more than t errors occur in a block of size n ($\alpha = P(t_x) = P(t_y)$), when transmitting over a binary symmetric channel with crossover probability p. Then,

$$\alpha = \sum_{i=t+1}^{n} \binom{n}{i} p^i (1-p)^{n-i},$$

and

$$P(B_i) \le 1 - (1-\alpha)^2. \quad (4.18)$$

Combining (4.17) and (4.18) yields an approximation for the probability of error in the computation. Since equation (4.18) is an upper bound on $P(B_i)$ it is possible that the entire approximation yields an upper bound on the probability of error. In support of this hypothesis, we computed the actual probability of error for a number of select codes which we use in our performance analysis and compared the actual result to the approximation. The results of the comparison suggest that the previous approximation is in fact an upper bound for the codes examined [7].

4) Performance Evaluation: In analyzing the performance of this scheme, we again considered a few classes of codes for which we can compute the total number of bits transmitted and the probability of error in the exchange. These classes include: Hamming codes, BCH codes, Reed-Solomon codes,[2] and the Golay code. In Fig. 4, we plot the rate (as defined in Section IV-A) vs. channel error probability for $N = 1000$ and $\delta = 10^{-6}$, for the various classes of codes. Similarly, in Fig. 5, we plot the rate vs. channel error probability for $N = 100$ and $\delta = 10^{-8}$. It is interesting to note that the BCH code performs best here due to its high error-correcting capability. In this case, the Golay code and the Reed-Solomon codes that have a higher error-correcting capability are an overkill for the required level of error protection and therefore, result in lower rates. A more complete presentation of our results, for various values of N, δ, and p is available in [7].

[2]Reed-Solomon codes are nonbinary codes but can be used for transmission over a binary channel by representing q-ary symbols by blocks of binary digits.

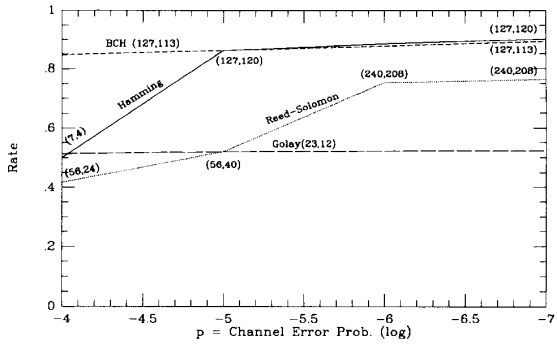

Fig. 4. Rate for the error-correction scheme with $N = 1000$ and $\delta = 10^{-6}$.

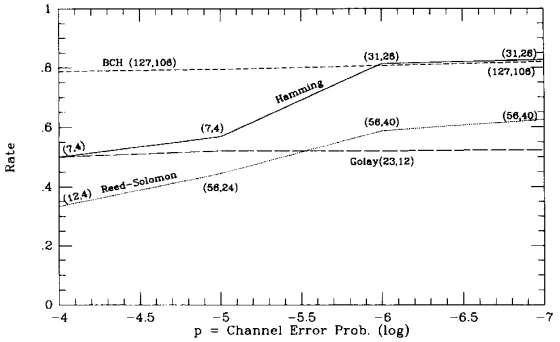

Fig. 5. Rate for the error-correction scheme with $N = 100$ and $\delta = 10^{-8}$.

C. Error Correction vs. Error Detection

In this section, we compare the performance of the error-correction scheme to that of the error-detection scheme. We base our analysis on the results presented in the previous two sections. In Section IV-A-3, we presented performance results for the error-detection scheme for a variety of codes; in Section IV-B-4, we did the same for the error-correction scheme. In order to compare the performance of the two schemes, we plot for each scheme the best rate that was achieved by any of the codes examined. In Fig. 6, we plot the best achievable rate for the error-detection and the error-correction schemes with $N = 1000$ and $\delta = 10^{-6}$, based on the information presented in Figs. 2 and 4, respectively. It can be seen from the figure that when the channel error probability is high, error-correction results in better rates than error detection. However, when the channel error probability is below 10^{-5}, the error-detection scheme results in

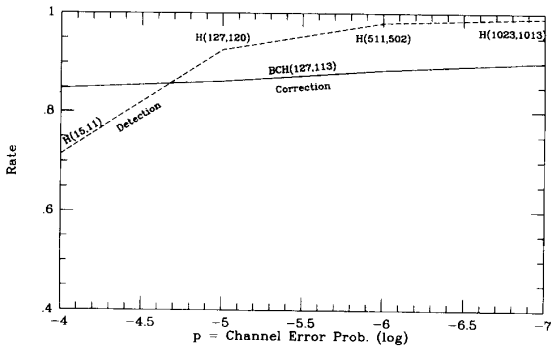

Fig. 6. Best achievable rate for the error-correction scheme and the error-detection scheme with $N = 1000$ and $\delta = 10^{-6}$.

better rates than the error-correction scheme. This result is reasonable because error detection requires fewer check bits than error correction; however, error correction requires the retransmission of fewer bits than error detection. Therefore, when the channel error probability is low, few errors occur, few retransmissions are required, and as a result the error-detection scheme performs better. When the channel error probability is high, many errors occur, many retransmissions are required, and as a result the error-correction scheme performs better. Similar results are obtained when comparing the two schemes for different values of N and δ [7].

V. Conclusion

In this paper, we examined the effects of noise on a specific distributed protocol which can be used to perform the computation of binary functions secretly and distributively. This protocol was used in [2] to prove an upper bound on the number of bits that must be transmitted over a secure channel in order to keep the value of any binary function ϵ-secret. It was assumed in [2] that both the public and secure channel are error free. We showed that the presence of noise in either the public or the private channels could impact the protocol significantly. The case of having a noisy public channel was studied in detail. It was shown that the complexity of the protocol was reduced as a result of the noise. Also, two error-control schemes were proposed in order to overcome the adverse effects of the noise. The case of a noisy secure channel remains unstudied. Also, it would be interesting to consider the effect of error control coding, as in Section IV, on the intruder. In particular, it is of interest to examine the effect of keeping such codes secret.

When studying the impact of coding on the eavesdropper one should also consider other possibilities for the noisy channels, such as having the intruder listen on an independent channel with independent noise. All of these cases will depend in some measure on the approach we developed here. Finally, it is of interest to study other protocols for the computation of f that might display superior performance.

References

[1] A. C. Yao, "Some complexity questions related to distributive computing," in *Proc. 11th Ann. ACM Symp. Theory of Computing*, Washington, DC, May 1979, pp. 209–213.

[2] A. Orlitsky and A. El Gamal, "Communication with secrecy constraints," *Proc. 16th Ann. ACM Symp. Theory of Computing*, Atlanta, GA, Apr. 1984, pp. 217–224.

[3] J. Ja'Ja', V. K. P. Kumar, and J. Simon, "Information transfer under different sets of protocols," *SIAM J. Computing*, vol. 13, no. 4, Nov. 1984.

[4] S. Lin and D. J. Costello Jr., *Error Control Coding: Fundamentals and Applications*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1983.

[5] F. J. MacWilliams and N. J. Sloane, *The Theory of Error-Correcting Codes*. New York: North Holland, 1977.

[6] W. W. Peterson and E. J. Weldon Jr., *Error-Correcting Codes*. Cambridge, MA: The MIT Press, 1972.

[7] E. Modiano, "Communication complexity of secure distributed computation in the presence of noise," Master's thesis, Dept. of Elect. Eng., Univ. of Maryland, College Park, MD, 1989.