Privacy Preserving Data Sharing With Anonymous ID Assignment

Larry A. Dunning, Member, IEEE, and Ray Kresman

Abstract—An algorithm for anonymous sharing of private data among $N$ parties is developed. This technique is used iteratively to assign these nodes ID numbers ranging from 1 to $N$. This assignment is anonymous in that the identities received are unknown to the other members of the group. Resistance to collusion among other members is verified in an information theoretic sense when private communication channels are used. This assignment of serial numbers allows more complex data to be shared and has applications to other problems in privacy preserving data mining, collision avoidance in communications and distributed database access. The required computations are distributed without using a trusted central authority.

Existing and new algorithms for assigning anonymous IDs are examined with respect to trade-offs between communication and computational requirements. The new algorithms are built on top of a secure sum data mining operation using Newton’s identities and Sturm’s theorem. An algorithm for distributed solution of certain polynomials over finite fields enhances the scalability of the algorithms. Markov chain representations are used to find statistics on the number of iterations required, and computer algebra gives closed form results for the completion rates.

Index Terms—Anonymization and deanonymization, cloud and distributed computing systems, multiparty computation, privacy preserving data mining, privacy protection, security and trust in cooperative communications.

I. INTRODUCTION

The popularity of internet as a communication medium whether for personal or business use depends in part on its support for anonymous communication. Businesses also have legitimate reasons to engage in anonymous communication and avoid the consequences of identity revelation. For example, to allow dissemination of summary data without revealing the identity of the entity the underlying data is associated with, or to protect whistle-blower’s right to be anonymous and free from political or economic retributions [1]. Cloud-based website management tools [2] provide capabilities for a server to anonymously capture the visitor’s web actions. The problem of sharing privately held data so that the individuals who are the subjects of the data cannot be identified has been researched extensively [3]. Researchers have also investigated the relevance of anonymity and/or privacy in various application domains: patient medical records [4], electronic voting [5], e-mail [6], social networking [7], etc.

Another form of anonymity, as used in secure multiparty computation, allows multiple parties on a network to jointly carry out a global computation that depends on data from each party while the data held by each party remains unknown to the other parties [8], [9]. A secure computation function widely used in the literature is secure sum that allows parties to compute the sum of their individual inputs without disclosing the inputs to one another. This function is popular in data mining applications and also helps characterize the complexities of the secure multiparty computation [10], [11].

This work deals with efficient algorithms for assigning identifiers (IDs) to the nodes of a network in such a way that the IDs are anonymous using a distributed computation with no central authority. Given $N$ nodes, this assignment is essentially a permutation of the integers $\{1, \ldots, N\}$ with each ID being known only to the node to which it is assigned. Our main algorithm is based on a method for anonymously sharing simple data and results in methods for efficient sharing of complex data. There are many applications that require dynamic unique IDs for network nodes [12]. Such IDs can be used as part of schemes for sharing/dividing communications bandwidth, data storage, and other resources anonymously and without conflict. The IDs are needed in sensor networks for security or for administrative tasks requiring reliability, such as configuration and monitoring of individual nodes, and download of binary code or data aggregation descriptions to these nodes. An application where IDs need to be anonymous is grid computing where one may seek services without divulging the identity of the service requestor [13].

To differentiate anonymous ID assignment from anonymous communication, consider a situation where $N$ parties wish to display their data collectively, but anonymously, in $N$ slots on a third party site. The IDs can be used to assign the $N$ slots to users, while anonymous communication [6], [14] can allow the parties to conceal their identities from the third party.

In another application, it is possible to use secure sum to allow one to opt-out of a computation beforehand on the basis of certain rules in statistical disclosure limitation [15] or during a computation [16] and even to do so in an anonymous manner. However, very little is known with respect to methods allowing agencies to opt-out of a secure computation based on the results of the analysis, should they feel that those results are too informative about their data [17].

The work reported in this paper further explores the connection between sharing secrets in an anonymous manner, distributed secure multiparty computation and anonymous ID assignment. The use of the term “anonymous” here differs from its
meaning in research dealing with symmetry breaking and leader election in anonymous networks [18], [19]. Our network is not anonymous and the participants are identifiable in that they are known to and can be addressed by the others.

Methods for assigning and using sets of pseudonyms, have been developed for anonymous communication in mobile networks [20], [21]. The methods developed in these works generally require a trusted administrator, as written, and their end products generally differ from ours in form and/or in statistical properties. To be precise, with $N$ nodes $n_i$, the algorithms of this paper distribute a computation among the nodes generating a permutation $\pi$ of $\{1, 2, \ldots, N\}$ chosen with a uniform probability of $N^{-1}$ from the set of all permutations of $\{1, 2, \ldots, N\}$ where $n_i$ will know only $\pi_i$. Such a permutation can also be produced by algorithms designed for mental poker [22]. The algorithms for mental poker [23] are more complex and utilize cryptographic methods as players must, in general, be able to prove that they held the winning hand. Throughout this paper, we assume that the participants are semi-honest [10], also known as passive or honest-but-curious, and execute their required protocols faithfully. Given a semi-honest, reliable, and trusted third party, a permutation can also be created using an anonymous routing protocol [6], [14]. Despite the differences cited, the reader should consult [20] and consider the alternative algorithms mentioned above before implementing the algorithms in this paper.

This paper builds an algorithm for sharing simple integer data on top of secure sum. The sharing algorithm will be used at each iteration of the algorithm for anonymous ID assignment (AIDA). This AIDA algorithm, and the variants that we discuss, can require a variable and unbounded number of iterations. Finitely-bounded algorithms for AIDA are discussed in Section IX. Increasing a parameter $S$ in the algorithm will reduce the number of expected rounds. However, our central algorithm requires solving a polynomial with coefficients taken from a finite field of integers modulo a prime. That task restricts the level to which $S$ can be practically raised. We show in detail how to obtain the average number of required rounds, and in the Appendix detail a method for solving the polynomial, which can be distributed among the participants.

II. A REVIEW OF SECURE SUM

Suppose that a group of hospitals with individual databases wish to compute and share only the average of a data item, such as the number of hospital acquired infections, without revealing the value of this data item for any member of the group. Thus, $N$ nodes $n_1, n_2, \ldots, n_N$ have data items $d_1, d_2, \ldots, d_N$, and wish to compute and share only the total value $T = d_1 + d_2 + \cdots + d_N$. A secure sum algorithm allows the sum $T$ to be collected with some guarantees of anonymity. Again, we assume the semi-honest model of privacy preserving data mining [10]. Under this model, each node will follow the rules of the protocol, but may use any information it sees during the execution of the protocol to compromise security.

Should all pairs of nodes have a secure communication channel available, a simple, but resource intensive, secure sum algorithm can be constructed. In the following algorithm, it is useful to interpret the values as being integer on first reading:

\begin{itemize}
  \item[1)] Each node $n_i$, $i = 1, \ldots, N$ chooses random values $r_{1,i}, \ldots, r_{i,N}$ such that $r_{1,i} + \cdots + r_{i,N} = d_i$
  \item[2)] Each “random” value $r_{i,j}$ is transmitted from node $n_i$ to node $n_j$. The sum of all these random numbers $r_{i,j}$ is, of course, the desired total $T$.
  \item[3)] Each node $n_j$ totals all the random values received as:
  \[ s_j = r_{1,j} + \cdots + r_{N,j} \]
  \item[4)] Now each node $n_i$ simply broadcasts $s_i$ to all other nodes so that each node can compute:
  \[ T = s_1 + \cdots + s_N \]
\end{itemize}

Algorithm 1 (Secure Sum): Given nodes $n_1, \ldots, n_N$ each holding an data item $d_i$ from a finitely representable abelian group, share the value $T = \sum d_i$ among the nodes without revealing the values $d_i$.

\begin{itemize}
  \item[1)] Each node $n_i$, $i = 1, \ldots, N$ chooses random values $r_{1,i}, \ldots, r_{i,N}$ such that $r_{1,i} + \cdots + r_{i,N} = d_i$
  \item[2)] Each “random” value $r_{i,j}$ is transmitted from node $n_i$ to node $n_j$. The sum of all these random numbers $r_{i,j}$ is, of course, the desired total $T$.
  \item[3)] Each node $n_j$ totals all the random values received as:
  \[ s_j = r_{1,j} + \cdots + r_{N,j} \]
  \item[4)] Now each node $n_i$ simply broadcasts $s_i$ to all other nodes so that each node can compute:
  \[ T = s_1 + \cdots + s_N \]
\end{itemize}

Example 1 (A Secure Sum Computation): In Table I two examples are shown, one for later use. The reader can ignore the columns labelled $\tilde{r}_{i,1}$ and $\tilde{d}_i$ and need not attribute any significance to the boldface type. In the example, the initial data items held by nodes $n_1, n_2, n_3$ and $n_4$ are $d_1 = 6, d_2 = 10, d_3 = 6$ and $d_4 = 2$ respectively. For example, node $n_2$ would transmit $7, 3, -5,$ and 5 to nodes $n_1, n_2, n_3$ and $n_4$ respectively. Node $n_2$ would receive $-10, 3, 11,$ and $-8$ from nodes $n_1, n_2, n_3$ and $n_4$ respectively. Then node $n_2$ would compute and transmit the total $s_2 = -4$ of the values received to all nodes. Finally, $n_2$ would compute the total of all the second round transmissions received, $24 = 18 + (-4) + 8 + 2$.

Our choices for the “random” numbers $r_{i,j}$ are for illustration and are not realistic. For example, given that each data item was originally in the range $0$ to $10$, the total would be in the range $0$ to $40$, and choosing random numbers modulus $41$ would be more appropriate.

In seeking to specify the security and privacy provided by our algorithms, we are indeed fortunate to have an abundance of definitions to choose from [24], [25], even when restricting ourselves with the semi-honest assumption. The choice of definition should be dependent on considerations such as whether private or graphically secured communications channels are used, etc. We follow the suggestion of a reviewer that a particular information theoretic definition of privacy be used. The central arguments of the proofs should remain useful when evaluating the algorithms with respect to other models of secure multiparty computation.

We will have need of some notational conventions. Note that given a function or sequence $f : D \rightarrow R$, its restriction to a subdomain $S \subseteq D$ is denoted by $f|_S$. For a function $f$ of two variables we will denote its restriction to the domain $A \times B$ by $f_{AB}$. Using bracket notation the set $\{1, \ldots, n\}$ is denoted
by \( n \). We also may identify an indexed set with its set of indexing subscripts as in writing \( n \in_{\mathcal{N}} \) or even \( \mathcal{N} \), for the nodes \( \{n_1, \ldots, n_N\} \) when clear from the context.

Suppose that a coalition \( C \subseteq \mathcal{N} \) of the nodes seeks to garner information about the private data \( d|_D \) of the other nodes \( D = [\mathcal{N} \setminus C] \). Since the parties are semi-honest, the possibly useful outside information available to a coalition seeking to garner the private information of other parties, consists of only the random numbers in the messages received from those parties \( r|_{DC} \), the partial sums \( s|_D \) and the total \( T \). We view the algorithm at a valid completion meaning that the relations between \( r_i, d_i, s, \text{ and } T \) established by Algorithm 1 are satisfied. It is evident that the coalition, \( C \), learns nothing.

**Theorem 1:** Suppose that \( |C| < N \). Then for any given \( d_C \), \( T, s|_D \) and \( r|_{DC} \) there is a one-to-one correspondence between all possible outcomes with values \( d|_D \) and those with any valid replacement values \( d|_D \leftarrow \tilde{d}|_D \).

**Proof:** Given values \( d|_D \) at termination, suppose w.l.o.g. that \( 1 \in D \). An equally likely outcome would have been to have altered \( r_{i1} \) to \( r_{i1} + d_i - d_i \) for \( i \in D \) and \( d|_D \) replaced \( d_i \) where all other values remained unchanged. The revised outcome is also valid. Of note is that since \( \sum d|_D = \sum \tilde{d}|_D \), it follows that \( \sum \tilde{r}_{i1} = \sum r_{i1} \).

In Table I an example with \( C = \{4\} \) shows two corresponding outcomes with \( d|_D = (6, 10, 6) \) and \( (8, 9, 5) \) respectively with the values \( \tilde{r}_{i1} \) adjusted as specified in the proof. Items known by \( C \) are in boldface. Interpreting all entries modulus 41 is required to actually meet the conditions of the theorem.

More formally, let \( P(Y_1, \ldots, Y_N \mid X_1, \ldots, X_N) = P(Y|N \mid X|N) \) denote the conditional probability distribution computed by a distributed protocol where \( X|N \) is the distribution of inputs provided by the \( N \) parties \( n|\mathcal{N} \), respectively, and similarly the outputs known to the respective parties are \( Y|N \). For privacy nothing additional about \( D = (X|D, Y|D) \) should be learned by an adversary whose view \( V_F \) of the computation includes all information known by the coalition including \( C = (X|C, Y|C) \) and any random values generated by or messages passed to or from coalition members. Denoting the conditional mutual Shannon information between \( A \) and \( B \) given \( C \) by \( I(A; B; C) \) [26] define:

**Definition 1 (Subset Collusion Resistance):** The algorithm computing the conditional probability \( P \) will be said to be subset collusion resistant to the set of semi-honest parties \( C = n|\mathcal{N} \setminus D \) provided that:

\[
I(V_F; D_F | C_F) = 0
\]

Thus, in the above definition, privacy is preserved if an adversary possessing the collective information of the coalition can learn nothing beyond that by observing the algorithm—loosely that \( V_F \) and \( D_F \) are independent of each other when \( C_F \) is factored out. Because the outputs \( Y|C \) must be generated by any protocol implementing the function, they are included in the view \( V_F \). It can be useful to add global information explicitly to the definition, but it is not necessary for our results. If a protocol/al- gorithm is subset collusion resistant for all subsets \( C \) of size \( t \) or less, then it is said to be \( t \)-private.

**Theorem 2 (Secure Sum is \( N \)-Private):** The secure sum method of Algorithm 1 is resistant to the collusion of any subset of the participating nodes, where the data values are from a finite abelian group.

For use in the following proof, two applications of the chain rule for conditional information suffice to show that \( I(A; H, C \mid C, D) = I(A; B; C \mid D) = I(B; C \mid D) + I(A; C \mid D, B) \).

**Proof:** The cases \( C \geq N - 1 \) are vacuously true. Since all parties are semi-honest, we may take \( V_F = (r|_{DC}, s|_N, r|_{C|N}, d|_{C|N}, T) \). Because \( s|_C \) can be calculated from \( r|_{C|N} \) and \( d|_C \) from \( r|_{C|N} \) and \( r|_{DC} \), this reduces to \( V_F = (r|_{DC}, s|_D, r|_{C|N}, T) \) and we compute:

\[
I(V_F; D_F, C|F) = I(r|_{DC}, s|_D, r|_{C|N}; T|d|_D, T \mid d|_C, T)
\[
= I(r|_{DC}, s|_D, r|_{C|N}; d|_D, T \mid d|_C, T)
\[
= I(r|_{C|N}; d|_D \mid d|_C, T) + 0
\]

and

\[
\leq I(d|_C; d|_D \mid d|_C; T) = 0
\]

where the information processing inequality [28, p. 541], with probabilistic functions for conditional mutual information is used in showing the inequality above. That \( I(r|_{DC}, s|_D; d|_D \mid d|_C, T \mid r|_{C|N}) = 0 \) follows from Theorem 1.

The secure sum algorithm just given remains \( N \)-private even when the input data are known as a multiset to all parties. We term this input permutation collusion resistance as the coalition knows the data held by the remaining parties only as a multiset. Because the parties are semi-honest, it does not matter if the values are known a priori or a posteriori. In the preceding proof, given that the multiset of values \( \{d|_{C|N}\} \) is known, the input distribution simply requires that the revised values be a permutation of the original values, and that proof, as well as, more directly, the proof of Theorem 1, hold with this restricted choice. This yields a trivial result which will, nonetheless, be of importance in the sequel:

**Corollary 3 (Secure Sum Hides Permutations):** The secure sum method of Algorithm 1 is input permutation resistant to the collusion of any subset of the participating nodes.

Other secure sum algorithms [16], [29] certainly can be used with physically or cryptographically secured communications channels. For example, it is easy to see that secure sum using a single Hamiltonian cycle [10] is input permutation collusion resistant provided that the coalition \( C \) is trapped in a connected region of the cycle. Such results can also be extended to provide privacy guarantees for the algorithms in subsequent sections should they utilize, e.g., a Hamiltonian cycle based secure sum. The secure sum technique can be employed with finite abelian groups other than \( GF(P) \) such as \( GF(2^n) \).

**III. TRANSMITTING SIMPLE DATA WITH POWER SUMS**

Suppose that our group of nodes wishes to share actual data values from their databases rather than relying on only statistical information as shown in the previous section. That is, each member \( n_i \) of the group of \( N \) nodes \( n_i|\mathcal{N} \) has a data item \( d_i \), which is to be communicated to all the other members of the group. However, the data is to remain anonymous. We develop a
collusion resistant method for this task using secure sum as our underlying communication mechanism. Our data items \(d_i\) are taken from a, typically finite, field \(F\). In the usual case, each \(d_i\) will be an integer value and \(F\) will be the field \(GF(P)\) where \(P\) is a prime number satisfying \(P > d_i\) for all \(i\). Thus, arithmetic will typically be performed using modulus \(P\), but other fields will also be used.

**Algorithm 2 (Anonymous Data Sharing With Power Sums):**
Given nodes \(n_1, \ldots, n_N\) each holding a data item \(d_i\) from a finitely representable field \(F\), make their data items public to all nodes without revealing their sources.

1) Each node \(n_i\) computes \(d_i^e\) over the field \(F\) for \(n = 1, 2, \ldots, N\). The nodes then use secure sum to share knowledge of the power sums:

\[
P_1 = \sum_{i=1}^{N} d_i^1, \quad P_2 = \sum_{i=1}^{N} d_i^2, \quad \cdots \quad P_N = \sum_{i=1}^{N} d_i^N
\]

2) The power sums \(P_1, \ldots, P_N\) are used to generate a polynomial which has \(d_1, \ldots, d_N\) as its roots using Newton’s Identities as developed in [30]. Representing the Newton polynomial as

\[
p(x) = c_N x^N + \cdots + c_1 x + c_0 \quad (1)
\]

the values \(c_0, \ldots, c_N\) are obtained from the equations:

\[
c_N = -1, \quad c_{N-1} = -\frac{1}{1} (c_N P_1), \quad c_{N-2} = -\frac{1}{2} (c_{N-1} P_1 + c_N P_2), \quad \cdots
\]

\[
c_N - m = -\frac{1}{m} \sum_{k=1}^{m} c_{N-m+k} P_k \quad (2)
\]

3) The polynomial \(p(x)\) is solved by each node, or by a computation distributed among the nodes, to determine the roots \(d_1, \ldots, d_N\).

The power sums \(P_i\) can be collected and shared using a single round of secure sum by sending them as an array and applying the method to the vectors transmitted and received. The power sums are symmetric functions, and thus no association is made between \(x\) and the value of \(p(x)\). However, nonetheless, the information contained in these sums can be used to find the values of the data items \(d_1, \ldots, d_N\).

The choice \(c_N = -1\), chosen for consistency with [31], may be replaced by \(c_N = 1\) or any other nonzero value. Also, note that in the typical \(F = GF(P)\) case, the solution for the \(c_i\) requires finding the multiplicative inverse of the coefficients 2, 3, 4, \ldots, \(N\) modulo \(P\). While the Euclidean algorithm could be used, the inverses \(1/x\) can easily be computed in the order \(x = 1, 2, \ldots, N\) by the formulae:

\[
q = P/x + r; \quad 1/x = -q(1/r) \quad (\text{mod } P)
\]

After the integer division with remainder \(r\), \(1/r\) will already be known, since \(r < x\).

---

**Example 2 (Transmission of Simple Data):** Suppose that \(N = 4\) nodes \(n_i\) wish to share a data item \(d_i\) which takes values 0 to 10. Using our sample data again, the values will be \(d_1 = 6\) for \(n_1\), \(d_2 = 10\) for \(n_2\), \(d_3 = 6\) for \(n_3\), and \(d_4 = 2\) for \(n_4\). Choice of the prime \(P = 11\) and \(F = GF(11)\) will serve to represent these numbers. The modulus 11 inverses needed will be \(1/2 = 6, 1/3 = 4, 1/4 = 3\). The nodes compute the powers shown in Table II. Solving each of the Newton identities (2) in turn yields \(c_4 = -1, c_3 = 1, c_2 = 2, c_1 = 9\). All the nodes receive the values \(P_1, P_2, P_3, \text{ and } P_4\) and can compute the polynomial and its roots to recover the original data items, 2, 6, 6, and 10, but not their indices.

There are, of course, many methods available to accomplish the solution of the polynomial \(p(x)\). A simple method which proved fast in practice is given in the Appendix.

In the sharing procedure, each node \(n_i\) adds a vector \(D_i = (d_1^1, \ldots, d_N^N)\) to the secure sum. The vector \(D_i\) can be regarded as a single integer in a very high radix. However, note that the multiset of values \(\{D_1, \ldots, D_N\}\) becomes known to all participants. The protection of the values against collusion is dependent upon the version of secure sum used and the particular set of colluding parties (and not merely the size of that set). We content ourselves with observing:

**Theorem 4 (Power Sum Data Sharing is \(N\)-Private):** The data sharing method of Algorithm 2 is resistant to the collusion of any subset of the participating nodes when based on the secure sum Algorithm 1.

Because the input data is present as a multiset in the output of every party and all parties are semi-honest the result is implied by our previous discussions of the secure sum Algorithm 1. The data sharing is anonymous in the sense that the sources of the data items cannot be traced. Of course, it is possible that a given data value would make sense only for a certain participant due to some factor such as the relative sizes of the participants. The paper [16] shows how anonymous opt-out can be used to address some of these concerns.

**IV. Sharing Complex Data With AIDA**

Now consider the possibility that more complex data is to be shared amongst the participating nodes. Each node \(n_i\) has a data item \(d_i\) of length \(b\)-bits which it wishes to make public anonymously to the other participants.

As the number of bits per data item and the number of nodes becomes larger, the method of the previous section becomes infeasible. Instead, to accomplish this sharing, we will utilize an indexing of the nodes. Methods for finding such an indexing are developed in subsequent sections. Assume that each node \(n_j\) has a unique identification (ID) or serial
number \( s_i \in \{1, 2, \ldots, N\} \). Further, suppose that no node has knowledge of the ID number \( s_i \) of any other node, and that \( s_1, \ldots, s_N \) are a random permutation of \( 1, \ldots, N \). This, again, is termed an Anonymous ID Assignment (AIDA) [16].

Such an AIDA may be used to assign slots with respect to time or space for communications or storage. It may be possible to simply have a database with central storage locations \( C_i \) such that each node simply stores its data there setting \( C_{s_i} := d_i \). This could occur if there was a trusted central authority, or if the storage operation \( C_{s_i} := d_i \) was untraceable [6], [14].

Given that there is no central authority (the situation for which secure sum was designed), secure sum can be used to accomplish the desired data sharing. Let \( U \) be a vector of \( b \)-bits. Each node creates a data item \( D_i \) of \( N \cdot b \)-bits. Numbering each of the \( N \), \( b \)-bit components \( 1, 2, \ldots, N \) have we have:

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & \ldots & N \\
D_i & = & 0 & \hat{0} & \cdots & d_i & \cdots & 0 \\
\end{array}
\]

The secure sum algorithm, given earlier in this paper, may now be used to collect the data items \( D_1, \ldots, D_N \). The algorithm is applied using \( GF(2)^{N \cdot b} \) as the abelian group. The group operation is bitwise exclusive-or, and each node \( n_i \) will choose \( N - 1 \) random entries \( r_{i,j} \), each composed of \( N \cdot b \) randomly chosen bits while calculating one entry, e.g., \( r_{i,4} \) to ensure sum (by XOR) equal to \( D_1 \).

V. HOW TO FIND AN AIDA

We present a simple algorithm for finding an AIDA which has several variants depending on the choice of the data sharing method at step (3) below. At one step, random integers or “slots” between 1 and \( S \) are chosen by each node. A node’s position will be determined by its position among the chosen slots, but provisions must be made for collisions. The parameter \( S \) should be chosen so that \( S \geq N \).

Algorithm 3 (Find AIDA): Given nodes \( n_1, \ldots, n_N \), use distributed computation (without central authority) to find an anonymous indexing permutation \( s : \{1, \ldots, N\} \rightarrow \{1, \ldots, N\} \).

1) Set the number of assigned nodes \( A = 0 \).
2) Each unassigned node \( n_i \) chooses a random number \( r_i \) in the range 1 to \( S \). A node assigned in a previous round chooses \( r_i = 0 \).
3) The random numbers are shared anonymously. One method for doing this was given in Section III. Denote the shared values by \( q_1, \ldots, q_N \).
4) Let \( q_1, \ldots, q_k \) denote a revised list of shared values with duplicated and zero values entirely removed where \( k \) is the number of unique random values. The nodes \( n_k \) which drew unique random numbers then determine their index \( s_i \) from the position of their random number in the revised list as it would appear after being sorted:

\[
s_i = A + \text{Card}\{q_j : q_j \leq r_i\}
\]

5) Update the number of nodes assigned: \( A = A + k \).
6) If \( A < N \) then return to step (2).

Example 3 (Execution of Algorithm to Find an AIDA): Suppose that four nodes participate in searching for an AIDA. For simplicity we continue our running example with \( S = 10 \) and random number choices 6, 10, 6, 2 again in the first round. The choices of \( r_{1,1} \) and \( r_{1,2} \) are 5 and 6 respectively in the second round while \( r_{2,1} \) and \( r_{2,2} \) choose 0 as they will already have indices assigned at that point. A trace of critical steps in the procedure is shown in Table III. The final AIDA result is then \( s_1 = 3 \) for node \( n_1 \), \( s_2 = 2 \) for node \( n_2 \), \( s_3 = 4 \) for node \( n_3 \), and \( s_4 = 1 \) for node \( n_4 \).

The number of rounds this algorithm takes is modeled by a Markov chain. While no absolute upper bound is possible, we will see in Section VII that the performance is very good, as one might expect, when \( S \) is much larger than \( N \). The various methods for sharing the random numbers at step (3) are addressed in Section VI.

The collusion resistance of AIDA depends upon the underlying secure sum algorithm used and the collusion resistance of that algorithm for a particular set of colluding nodes \( C \). The strongest result possible can be obtained by using our simple, but inefficient, secure sum Algorithm 1:

Theorem 5 (AIDA is \( N \)-Private): Algorithm 3 is resistant to the collusion of any subset of the participating nodes when the secure sum method of Algorithm 1 is used.

Proof: We sketch the essential step of the proof by viewing the AIDA algorithm at its final termination. Suppose that there are \( M \) iterations of steps (2)–(6). Let \( r_{i,1}^1, \ldots, r_{i,N}^M \) denote the random values chosen by nodes \( n_1, \ldots, n_N \) at step (2) in iteration \( n_i \). Denote by \( i \rightarrow s(i) \) the final permutation \( s(i) = s_i \) produced by AIDA.

Let \( t(i) \) denote any permutation of \( \{N\} \). Suppose that the random choices by node \( n_i \) at iteration \( n_i \) during execution had been \( r_{i,1}^1 = r_{i,2}^m \), rather than \( r_{i,1}^m \). This choice of random numbers would be equally likely and would have resulted in the final assignment \( n_i \rightarrow s(t(i)) \).

Let \( C \) denote the coalition of colluding nodes and \( D = \{N \} \setminus C \) the remaining nodes. Given any desired permutation \( p : D \rightarrow D \), define

\[
p'(i) = p(\tilde{i}) | i \in D; \quad p'(i) = i | i \in C.
\]

The selection \( t(i) = s^{-1}(p'(i)) \) yields

\[
n_i \rightarrow s(s^{-1}(p'(i)) = p'(i).
\]

Thus all permutations of the values \( \{s(i) | i \in D\} \) are equally likely and this is independent of the number of iterations. Setting \( C \) to be the empty set, we have shown:

Corollary 6 (AIDA Produces Random Permutations): Algorithm 3 results in a permutation of the participating nodes chosen from the uniform distribution (all assignments are equally likely) when the secure sum method of Algorithm 1 is used.
TABLE IV

<table>
<thead>
<tr>
<th>Round</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon_1 = \varepsilon_6$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\varepsilon_2 = \varepsilon_{10}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\varepsilon_3 = \varepsilon_6$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\varepsilon_4 = \varepsilon_2$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$T$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

VI. COMPARISON OF AIDA VARIANTS

In the previous section the algorithm to find an AIDA required that the random numbers be shared anonymously at step (3). We now look at three methods which are variants of that procedure. The parameter $S$ must be chosen in each case. The expected number of rounds depends only on the selection of $S$ and not on the variant chosen.

A. Slot Selection AIDA

The slot selection method was developed in [16] where a more detailed explanation may be found. In this variant of the AIDA algorithm, each node $n_i$ submits the euclidean basis vector $e_{i,1}$ to a secure sum algorithm. A node which has received an assignment in a previous round, however, submits the zero vector. The sum $T$ of these vectors is computed over the abelian group $GF(N + 1)^2$ using a secure sum algorithm. The random numbers chosen and their multiplicities are simple to determine as $T_k = Card\{i : r_i = k\}$.

Example 4 (A Slot Selection AIDA): With the choice $S = 10$ the AIDA example from the previous section would have executions of secure sum at each round with results as shown in Table IV. Using our example secure sum algorithm $N = 4$ vectors of $S = 10$ random numbers (not shown) would need to be chosen by each of the $N = 4$ participating nodes at each round.

This variant of the algorithm has as its main drawback the very long message lengths that are encountered when using large $S$ to keep the number of expected rounds small.

B. Prime Modulus AIDA

A prime $P > S$ is chosen. Generally, $P$ will be chosen as small as possible subject to this restriction. The random numbers chosen are distributed at step (3) as in Section III using the field $F = GF(P)$ to compute the required power sums, the Newton polynomial $p(x)$, and the polynomial roots.

This variant will be seen to result in shorter message lengths for communication between nodes. Again, the computation required to find the roots of the Newton polynomial is addressed in the Appendix. However, note that this computation can be delayed and thus overlaps any additional required rounds.

Additional rounds of the AIDA algorithm can proceed almost immediately as it is not necessary to solve $p(x) = 0$ before proceeding to the next round. Each node $n_i$ merely computes the derivative polynomial $p'(x)$ and evaluates that polynomial at its chosen random value $r_i$. The value $r_i$ is a multiple root if and only if $p'(r_i) = 0$. Thus, if $p'(r_i) = 0$ then node $n_i$ chooses a new random number $r_i$ for use in the next round. If $p'(r_i) \neq 0$ then the node $n_i$ has an assignment and for subsequent rounds will use $r_i = 0$.

Example 5 (An Application of the Derivative Test): Continuing the example of Section V, at step (3) of the first round the Newton polynomial $p(x) = 10x^5 + 2x^3 + 3x^2 + x + 6$ and its derivative $p'(x) = 6x^3 + 6x^2 + 7x + 1$ can be calculated (modulus 11) and evaluated as follows:

$$x = r_1 = 6, r_2 = 10, r_3 = 6, r_4 = 2$$

$$p(x) = 0, 0, 0, 0, 0$$

$$p'(x) = 0, 4, 0, 7$$

Thus, only nodes $n_2$ and $n_4$ have received assignments in the first round.

C. Sturm’s Theorem AIDA

It is possible to avoid solution of the Newton polynomial entirely. Sturm’s theorem [32] allows the determination of the number of roots of a real polynomial $p(x)$ in an interval $(a, b]$ based on the signs of the values of a sequence of polynomials derived from $p(x)$. The sequence of polynomials is obtained from a variant of the Euclidean Algorithm.

As in the previous variant, the power sums are collected and the Newton Polynomial is formed. However, the field used for computation is the field of rational numbers $Q$. The test $p'(r_i) = 0$ is again sufficient to determine whether or not $n_i$ has received an assignment. A computational advantage arises in that the nodes do not need to solve the Newton polynomial $p(x)$ to determine the (now implicitly) shared values. Assume that $x = 0$ is not a root of $p(x)$ as $x^k$ has been factored out immediately if applicable. Each node $n_i$ which has received an assignment must count separately multiple roots and also forms $g(x) = gcd(p(x), p'(x))$. A multiple roots version of Sturm’s theorem [32] is then applied to calculate the number of roots for the polynomial $p(x)$ in the range, $(0, r_i]$. (Note that $r_i$ itself is not a multiple root allowing application of the theorem.) The polynomial $g(x) = gcd(p(x), p'(x))$ is a by-product of this computation. The same Sturm procedure is applied to $g(x)$ thus obtaining a count of the multiple roots in the same range, $(0, r_i]$. The index received by $n_i$ is then

$$s_i = Card\{x : p_i(x) = 0 \land 0 < x \leq r_i\}$$

$$- Card\{x : g_i(x) = 0 \land 0 < x \leq r_i\}$$

The collected power sums $P_i$ are integers. To guarantee privacy, ideally, compute sums with Algorithm 2 using a field $GF(P)$ with $P$ greater than any possible value of $P_i$.

Our timings showed that using Sturm’s theorem is not currently competitive with the various methods of polynomial solution using the “prime modulus” approach and runs twice as slow at best. Although we, therefore, do not give an example, the construction is straight forward. The application of Sturm’s theorem requires use of an ordered field resulting in large polynomial coefficients. Unfortunately, we do not currently know of a computationally reasonable analog of this result which is usable over a finite field. However, some results in this direction are available [33].
TABLE V
DATA BITS REQUIRED PER MESSAGE

<table>
<thead>
<tr>
<th>S</th>
<th>P</th>
<th>Method</th>
<th>N = 10</th>
<th>25</th>
<th>100</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>17</td>
<td>Prime</td>
<td>50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sturm</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>103</td>
<td>Prime</td>
<td>70</td>
<td>175</td>
<td>700</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sturm</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Slot</td>
<td>400</td>
<td>500</td>
<td>700</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>503</td>
<td>Prime</td>
<td>90</td>
<td>225</td>
<td>900</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sturm</td>
<td>1980</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Slot</td>
<td>2000</td>
<td>2500</td>
<td>3500</td>
<td></td>
</tr>
<tr>
<td>$10^4$</td>
<td>S + 7</td>
<td>Prime</td>
<td>1440</td>
<td>350</td>
<td>1400</td>
<td>14000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sturm</td>
<td>3080</td>
<td>22750</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Slot</td>
<td>40000</td>
<td>50000</td>
<td>70000</td>
<td>100000</td>
</tr>
<tr>
<td>$10^7$</td>
<td>S + 19</td>
<td>Prime</td>
<td>2400</td>
<td>600</td>
<td>2400</td>
<td>24000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sturm</td>
<td>5280</td>
<td>39000</td>
<td>84840</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Slot</td>
<td>4 · $10^7$</td>
<td>5 · $10^7$</td>
<td>7 · $10^7$</td>
<td>$10^8$</td>
</tr>
</tbody>
</table>

D. Communications Requirements of AIDA Methods

We now consider the required number of data bits for each of the three variant methods just described. This is the number of data bits that would be transmitted in each packet by the secure sum algorithm introduced earlier. The required numbers of data bits $B$ are slightly overestimated by the formulae:

$$B_{\text{prime}} = N \cdot \lceil \log_2(P + 1) \rceil$$
$$B_{\text{sturm}} = \frac{N(N - 1)}{2} \cdot \lceil \log_2(N) \rceil \cdot \lceil \log_2(S) \rceil$$
$$B_{\text{slot}} = S \cdot \lceil \log_2(N + 1) \rceil$$

Table V gives the number of required data bits for selected values of $N$ and $S$. Header bits required by the transmission protocol, etc. are not included. Where the number of data bits required by the Sturm method is not competitive, a “–” has been entered to enhance readability of the table.

The computational requirements of the “slot selection” appear, at first, to be trivial. However, for every root that the “prime modulus” method must check, “slot selection” must input/output and check $\lceil \log_2(N) \rceil$ bits. This simple method can be recommended, for small $N$ and $S$, when multiple rounds of the algorithm can be tolerated.

Computational time required by the “prime modulus” method grows with $S$. For most of the entries in the table, the amount of time is trivial. At $N = 1,000$ and $S = 10,000$ the computation required becomes a consideration. Here, somewhat less than 0.03 seconds of CPU time were required using the method described in the Appendix. When $S$ was raised to 10,000,000 this increased to 9 seconds. The computation can be distributed across the participating nodes easily and efficiently with the expected nearly $N$-fold reduction in wall clock time required for computation.

VII. The Completion Rate After $R$ Rounds

Two nodes might make identical choices of random numbers, or slots as they will be termed in this section. One can only guarantee that a complete assignment of $N$ nodes using $S$ possibilities for slots or random number choices and $R$ rounds will occur with at least a desired probability $P_{S-\text{uni}}(R, N) = P(R, N)$. The parameter $S$ will usually be implicitly set to $S = s$ by its omission in this section.

The reader may observe that estimating the number of assignments made in one round is essentially the well-known birthday problem. This and similar approaches have been used to compute bounds for our problem and others [12], [16]. However, to compute the expected number of assignments in $R$ rounds precisely, it is necessary to know the probability distribution of the assignments made in a single round, and not just the average number. We now show how to compute the probabilities $P(R, N)$.

As a first step, consider the probability $p(N, A, C)$ that a single round, starting with $N$ nodes and $s$ slots, results in $A$ assignments and $C$ conflicts with as those slots have been chosen by more that one node. Note that $Z = s - A - C$ is the number of slots that no (zero) nodes have chosen. The recurrence relations (3) determine $p(N, A, C)$:

$$p(0, 0, 0) = 1, \quad p(1, 0, 0) = 1$$
$$p(N, A, C) = 0 \text{ whenever } C < 0 \lor A < 0$$
$$p(N, A, 0) = 0 \text{ whenever } N \neq A$$
$$p(N, A, C) = \frac{1}{s} (s - A - C + 1) \cdot p(N - 1, A - 1, C) + \frac{1}{s} C \cdot p(N - 1, A, C) + \frac{1}{s} (A + 1) \cdot p(N - 1, A + 1, C - 1)$$

The formulae are derived by assuming that $N - 1$ nodes have chosen slots and looking at the next choice. The $N$th node is to choose a slot resulting in $A$ assignments and $C$ conflicts. The slot it chooses could be unassigned, already in conflict with multiple occupants, or already assigned with exactly one occupant. For example, in the last case, the number of assignments falls by one with probability $(A + 1)/S$ where the configuration before the choice occurred with probability $p(N - 1, A + 1, C - 1)$. After the additional choice, there are $N$ nodes, $A$ of which are assigned and the number of conflicted slots rises by 1 to $C$.

More base conditions than are necessary have been given in the recursion relations (3). For efficiency, the values $P(N, A, C)$ need to be cached during computation. However, we will need only to remember $p_{N,A}$, the probability that beginning with $N$ nodes and $s$ slots, exactly $A$ assignments have been made after one round:

$$p_{N,A} = \sum_{C=0}^{\lfloor N/2 \rfloor} p(N, A, C)$$

Let $P(R, N, A)$ be the probability of $A$ assignments being made after $R$ rounds so that $P(R, N) = P(R, N, N)$. Define $\vec{P}(R, N)$ to be a vector of $N$ entries, which may be interpreted as either row or column vectors, giving probabilities:

$$\vec{P}(R, N) = \begin{pmatrix} P(R, N, N) \\ P(R, N, N - 2) \\ \vdots \\ P(R, N, 1) \\ P(R, N, 0) \end{pmatrix}$$

$\vec{P}(0, N) = e_N = (0, 0, \ldots, 0, 1)$

There is no entry for $A = N - 1$ because a single collision results in $A = N - 2$. 
TABLE VI
PERCENT PROBABILITY OF AN INCOMPLETE ASSIGNMENT
AFTER R ROUNDS

<table>
<thead>
<tr>
<th>S</th>
<th>N = 10</th>
<th>25</th>
<th>100</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>R=1</td>
<td>98.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R=2</td>
<td>45.9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R=3</td>
<td>6.70</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>93.2%</td>
<td>45.9%</td>
<td>0.677</td>
<td>0.0097</td>
</tr>
<tr>
<td></td>
<td>&lt;100.0</td>
<td>&lt;100.0</td>
<td>&lt;100.0</td>
<td>&lt;100.0</td>
</tr>
<tr>
<td>500</td>
<td>0.200</td>
<td>0.236</td>
<td>27.3</td>
<td></td>
</tr>
<tr>
<td>10^4</td>
<td>0.450</td>
<td>2.96</td>
<td>39.2</td>
<td>&lt;100.0</td>
</tr>
<tr>
<td>10^5</td>
<td>4.53</td>
<td>0.000316</td>
<td>0.00961</td>
<td>36.3</td>
</tr>
<tr>
<td>5·10^5</td>
<td>0.000450</td>
<td>0.00300</td>
<td>0.495</td>
<td>4.88</td>
</tr>
<tr>
<td></td>
<td>0.000990</td>
<td>0.995</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The matrix \( P(N) \) gives the transition probabilities for a single round of AIDA starting with \( N \) nodes and ending with \( N - A \) nodes yet to be assigned.

\[
P(N) = \begin{pmatrix}
    P_{0,0} & P_{0,2} & P_{0,3} & \cdots & P_{0,N-N-0} \\
    0      & P_{2,0} & P_{2,1} & \cdots & P_{2,N-2} \\
    0      & 0       & P_{3,0} & \cdots & P_{3,N-3} \\
    0      & 0       & 0       & \cdots & P_{N,0} \\
    \vdots & \vdots  & \vdots  & \ddots & \vdots \\
    0      & 0       & 0       & \cdots & P_{N,N} 
\end{pmatrix}
\]

Note that across each row of the array, the number of unassigned nodes is the same. For example, all the probabilities in the second row are for a single collision. The equation for getting the vectors of probabilities for the next round from the previous round is then found by matrix multiplication.

\[
\tilde{P}(R, N) = P(N) \cdot \tilde{P}(R-1, N) = (P(N))^R \cdot \tilde{P}(0, N)
\]

For the conventional form of a Markov chain, transpose \( P(N) \) to get a right stochastic matrix whose rows each sum to 1.

\[
\tilde{P}(H, N) = \tilde{P}(R-1, N) \cdot P(N)^T
\]

Thus, the formulae (5) and (6) can be used to obtain the desired probability \( P(R, N) = \tilde{P}(R, N)_1 \) of termination in \( R \) rounds.

The probability of incomplete assignment in \( R \) rounds will be denoted by \( S(R, N) \). We have calculated these for small \( N \) with values as high as \( 1000 \) and \( S = 56,000,000 \) using floating point computation with 256 bit mantissas, and verified the results by simulation. We formally state:

**Theorem 7:** The completion rate \( P(R, N) = \tilde{P}(R, N)_1 \) of Algorithm 3 with \( N \) nodes after \( R \) rounds is given by

\[
P(R, N) = ((P(N))^R \cdot e)_N^1
\]

Table VI gives values for percentages of incomplete assignments. Percentages are rounded up to three significant digits.

**Example 6:** For \( N = 4 \) nodes, \( S = 10 \) slots, and \( R = 2 \) rounds, in (7), we show the probabilities for \( A = 4, 2, 1, 0 \) assignments in terms of the same for \( R = 1 \) as given by the matrix calculation \( P(2, 4) = P(4) \cdot P(1, 4) \) of (5). For \( R = 1 \) the probability of a complete assignment is \( P_{S=10}(1, 4) = 50.4\% = 1 - 49.6\% = 1 - IS_{10}(1, 4) \). This can be verified by multiplying the same matrix \( P(4) \) with the starting state \( (0, 0, 0, 1) \) as the column vector on the right showing that \( P(1, 4) \) is simply the right-most column of \( P(4) \). For \( R = 2 \) the probability of a complete assignment is \( 93.28\% = 1 - 6.71\% \).

\[
\begin{pmatrix}
    0.932832 \\
    0.00016 \\
    0.0000084 \\
    0.0000004
\end{pmatrix} = \begin{pmatrix}
    1 & 0.9 & 0.72 & 0.504 \\
    0 & 0.1 & 0.27 & 0.432 \\
    0 & 0.01 & 0.036 & 0.036 \\
    0 & 0 & 0.028 & 0.028
\end{pmatrix}
\]

The entries in the matrix are, of course, calculated from the recurrence relations. However, one can validate some of the entries indirectly. For example, suppose the four nodes each choose a number in the range 1–10 and no assignments are made. Let \( C(n, k) \) denote the number of combinations of \( n \) items taken \( k \) at a time. Then \( p_{4,0} = 0.028 = (10 + C(4, 2) \cdot C(10, 2))/10000 \) where the first term of the numerator is the number of ways in which all four numbers can be identical while the second term is the number of ways in which the four numbers can form two pairs of identical numbers.

An implementer would generally want to choose \( S \) to provide a high completion rate within just a few rounds. The material in this section is the most direct way to determine such completion rates. Section VIII indicates how additional information is obtained using computer algebra techniques.

**VIII. ALGEBRAIC COMPLETION STATISTICS**

For many purposes, the formulae of Section VII provide a satisfactory answer. However, the rich literature on absorbing Markov chains and the availability of computer algebra packages provide many other possibilities for analysis. To determine a desirable value for the number of slots “\( S = s \)” one can take advantage of the fact that the probabilities \( p_{N,N/p} \) are representative as rational functions of the number of slots \( S = s \). In fact \( P(N) \) is the \( N \) by \( N \) upper, left-hand corner of an infinite matrix \( P \). When \( N \) is small, the entries, which have no discernible pattern, can be calculated by a computer algebra package from the recurrence relations yielding:

\[
P = \begin{pmatrix}
    1 & z^{-1} & (z-2)(z-1)^{-1} & (z-3)(z-4)^{-1} & \cdots \\
    0 & 1 & 2(z-1)^{-1} & 3(z-2)^{-1} & \cdots \\
    0 & 0 & z & 2(z-1) & \cdots \\
    0 & 0 & 0 & z^2 & \cdots \\
    \vdots & \vdots & \vdots & \ddots & \ddots
\end{pmatrix}
\]

**A. The Average Number of Rounds**

Following standard methods for absorbing Markov chains [34], we determine the average number of rounds required.

Let \( Q(N) \) be the matrix formed by dropping the first row and the first column of \( P(N) \) which correspond to the absorbing state \( A = N \) of the Markov chain (note the transposition). Given that \( I_k \) is the \( k \) by \( k \) identity matrix, denote the fundamental matrix for the Markov chain [34] by:

\[
N(N) = (I_{N-1} - Q(N))^{-1}
\]

\( N \) is, interestingly, a lower-triangular, infinite matrix from which the upper-left \( N - 1 \) by \( N - 1 \) corner forms \( N(N) \). Let \( I \) denote a vector with all entries set to 1. The value of most
interest from the standard theory [34] in our context is probably the last entry of the vector \( \mathbf{N}(N) \cdot \mathbf{1} \)
\[
R_{\text{ave}}(N) = \left( \mathbf{N}(N) \cdot \mathbf{1} \right)_{N-1}
\]
which is the average number of rounds the algorithm will take. Related statistics such as the variance of \( R_{\text{ave}}(N) \) can be determined. One starting point would be the related work [35] which deals with bounds for the slotted ALOHA protocol where there are arrivals.

**Example 7 (An Average Number of Rounds Calculation):** For larger values of \( N \), say \( N > 16 \), we would normally compute \( \mathbf{P}(N) \) and \( \mathbf{N}(N) \) numerically, but for our small running example with \( N = 4 \) we proceed symbolically. A calculation with a computer algebra package gives
\[
R_{\text{ave}}(N) = \frac{s^2 (s^2 + 7s - 2)}{(s-1)(s+1)(s+2)}
\]
Substituting \( s = 10 \) yields 1.56127 rounds required on the average. If one is dissatisfied with this and would prefer a better average of 1.1 rounds, solution of
\[
s^2 (s^2 + 7s - 2) = \left( \frac{11}{10} \right) (s-1)^2 (s+1)(s+2)
\]
yields a root at \( s = 59.2225 \) requiring a choice of \( s > 60 \).

**B. Asymptotic Rates of Complete Assignments**

Exact rates or somewhat simpler asymptotic bounds on the percentage of incomplete assignments \( I(R, N) \) for any given number of rounds can be obtained. The second-largest eigenvalue of the matrix \( \mathbf{P}(N) \) gives the asymptotic rate at which the percentage of incomplete assignments decreases between rounds. However, more precise information is available.

Since \( \mathbf{P}(N) \) is triangular, the eigenvalues are easy to determine and for parameters of interest the second-largest is \( 1/s \). We restrict our attention to cases in which these eigenvalues are distinct. The only currently known exception is for the parameter, \( S = 9 \) which yields the eigenvalue \( 1/81 \) twice. For parameters \( 4 \leq N \leq S \leq 1000 \) there are no other occurrences. In the presence of duplicate eigenvalues Perron’s formula for matrix powers [36] can be used to obtain a similar analysis, albeit more complex. Instead, we suggest simply using a slightly decreased value of \( s \) to obtain the bounds using the methods developed below should duplicate eigenvalues occur.

Using the matrix \( \mathbf{Q}(N) \), \( 1/s \) is the spectral radius and Gelfand’s formula yields the asymptotic rate of contraction \( 1/s \). The sum of the entries of \( \mathbf{F}(R, N) \) is 1 and thus:
\[
I(R, N) = \mathbf{1} \cdot (e_{N-1} \cdot \mathbf{Q}(N)^R)
\]
Let \( v \) denote the eigenvalues taken from the diagonal of \( \mathbf{Q}(N) \) and let \( \mathbf{E}(N) \) denote the matrix composed of a basis of corresponding left eigenvectors as rows. Representing \( \mathbf{Q}(N) \) with respect to the eigenvector basis we arrive at
\[
I(R, N) = \mathbf{1} \cdot \left( \left( v_{N-1} \cdot \mathbf{E}(N) \right) \ast \left( v_1^R, \ldots, v_{N-1}^R \right) \right) \cdot \mathbf{E}(N)^{-1}
\]
where “\( \ast \)” indicates component-by-component multiplication of the two vectors. The formula transforms the starting state \( v_{N-1} \) into the eigenvector basis where the multiplication by powers of the eigenvalues performs the equivalent of the transformation by \( \mathbf{Q}(N)^R \). The result is then transformed back to euclidean coordinates and the probabilities of the incomplete states are tallied with the inner product using \( \mathbf{1} \).

Collecting the terms of formula (8) above yields an expression for \( I(R, N) \) of the form (9) with result:

**Theorem 8:** The expected percentage of incomplete assignments of \( N \) nodes after \( R \) rounds is:
\[
I(R, N) = a_1 v_1^R + a_2 v_2^R + \cdots + a_{N-1} v_{N-1}^R
\]
where \( v_1, \ldots, v_n \) are the distinct eigenvalues from the diagonal of the truncated transition matrix \( \mathbf{Q}(N) \), and \( a_1, \ldots, a_{N-1} \) are the solution of the linear equations:
\[
\begin{pmatrix}
I(0, N) \\
I(1, N) \\
\vdots \\
I(N-2, N) \\
I(N-1, N)
\end{pmatrix} =
\begin{pmatrix}
1 & \cdots & 1 \\
v_1 & \cdots & v_{N-1} \\
\vdots & \cdots & \vdots \\
v_{N-2} & \cdots & v_{N-1} \end{pmatrix}
\begin{pmatrix}
a_1 \\
a_2 \\
\vdots \\
a_{N-1}
\end{pmatrix}
\]
Note that the Vandermonde coefficient matrix can be ill-conditioned. Both special methods and explicit formulae exist for use in computing its inverse. It follows immediately that:

**Corollary 9:** The expected percentage of incomplete assignments of \( N \) nodes after \( R \) rounds is asymptotically:
\[
I_\ast(R, N) = a_1 \cdot \left( \frac{1}{s} \right)^R
\]
where \( a_1 \) is computed by Theorem 8; i.e., the limit of the quotient of the approximately equal quantities is 1 as \( R \to \infty \).

The quantities \( a_2 \) and \( v_1 \) are implicitly functions of \( s \). Again, keeping \( s \) a free variable is useful only for small \( N \).

**Example 8 (Calculation of an Asymptotic Bound):** In Section VII using \( N = 4 \) and \( S = 10 \), we already computed \( I(0, 4) = 1, I(1, 4) = 0.496, \) and \( I(2, 4) = 0.67168 \). For subsequence use, we sort the entries from the diagonal of \( \mathbf{Q}(N) \) into descending order \( v_1 = 1/10, v_2 = 7/250, \) and \( v_3 = 1/10! \).

Substituting into (10) yields:
\[
\begin{pmatrix}
1 \\
0.496 \\
0.67168
\end{pmatrix} =
\begin{pmatrix}
1 & 1 & 1 \\
0.1 & 0.028 & 0.01 \\
0.01 & 0.0001784 & 0.0001
\end{pmatrix}
\begin{pmatrix}
a_1 \\
a_2 \\
a_3
\end{pmatrix}
\]
Solving these linear equations, gives the form of (9) for the exact average rate of incompleteness:

\[
I(R, N) = 7.5 \left( \frac{1}{10} \right)^R - 10.5 \left( \frac{7}{250} \right)^R + 4 \left( \frac{1}{100} \right)^R
\]
Thus, we also obtain a simpler asymptotic rate for incomplete assignments of:
\[
I_\ast(R) = 7.5 \cdot \left( \frac{1}{10} \right)^R - a_1 \cdot \left( \frac{1}{s} \right)^R
\]
This is seen to be an upper bound that rapidly converges:

<table>
<thead>
<tr>
<th>( R )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( I(R) )</td>
<td>0.496</td>
<td>0.067168</td>
<td>0.0072735</td>
<td>0.000743586</td>
</tr>
<tr>
<td>( I_c(R) )</td>
<td>0.75</td>
<td>0.675</td>
<td>0.0075</td>
<td>0.00075</td>
</tr>
</tbody>
</table>

\[ \square \]

**IX. FINITE TERMINATION**

Although the algorithms developed here terminate with probability 1, there is no absolute upper bound on the number of rounds required. Under some assumptions, it has been proven that finite termination cannot be guaranteed for the simpler leader election problem [18], [19]. While there may be extreme conditions under which no algorithm for AIDA can be guaranteed to terminate, we conjecture only that at least \( N \) sequential communications are required in such an algorithm. On the other hand, the algorithms of [20], [21] are already collision free, but do not generate a permutation chosen at random from all possible permutations. For the current problem, the number of rounds is typically small and we do not recommend seeking finitely bounded termination.

For completeness, we sketch a cryptographic approach, that could guarantee finitely bounded termination, even without a trusted authority. (Note that some mental poker algorithms [23] could also be used to make this guarantee.) Suppose each node \( n_i \) has a unique, but not anonymous identifier \( 1 \leq A_i \leq N \), then the numbers \( R_i = E(s_i \cdot N + A_i) \) are unique where \( E() \) is an encryption function, and \( s_i \) is the usual sufficiently long seed (random number) known only to \( n_i \). The function \( E() \) may be cooperatively generated with inverse unknown to the nodes individually using techniques from [37]. The use of these random numbers \( (r_i = R_i) \) at step (2) of Algorithm 3 would guarantee termination in a single round. However, polynomial solution for \( S \) of the required size is impractical.

The computational problems with this approach can be overcome by using the numbers \( R_i \) as pseudorandom bit streams in Algorithm 3. Each node \( n_i \) takes \( b \) bits from the head of its \( R_i \) bit stream at each round to form \( r_i \). Two problems arise. First, nodes drawing identical random numbers in a round must add their position in the list of random numbers generated in that round \( p_i = \text{Card}(\{r_i < r_j : 1 \leq j \leq N\} < N \) to the tail of their bit stream. This keeps all the bit streams distinct. Second, the analysis of the number of rounds required in Sections VII and VIII no longer applies \textit{a priori} unless the encryption function \( E() \) has the difficult to obtain plausible deniability property [38]. However, provided that \( b > \lceil \log_2(N) \rceil \), the bit strings \( R_i \) become shorter at each round and a fixed bound is easily calculated.

**X. CONCLUDING REMARKS**

Each algorithm compared in Section VI can be reasonably implemented and each has its advantages. Our use of the Newton identities greatly decreases communication overhead. This can enable the use of a larger number of “slots” with a consequent reduction in the number of rounds required. The solution of a polynomial can be avoided at some expense by using Sturm’s theorem. The development of a result similar to the Sturm’s method over a finite field is an enticing possibility.

With private communication channels, our algorithms are secure in an information theoretic sense. Apparently, this property is very fragile. The very similar problem of mental poker was shown to have no such solution [22] with two players and three cards. The argument of [22] can easily be extended to, e.g., two sets each of \( N \) colluding players with a deck of \( 2N + 1 \) cards rather than our deck of \( 2N \) cards.

In contrast to bounds on completion time developed in previous works, our formulae give the expected completion time exactly. We conjecture the asymptotic formula of Corollary 9, based on computational experience, to be a true upper bound.

All of the noncryptographic algorithms have been extensively simulated, and we can say that the present work does offer a basis upon which implementations can be constructed. The communications requirements of the algorithms depend heavily on the underlying implementation of the chosen secure sum algorithm. In some cases, merging the two layers could result in reduced overhead.

**APPENDIX**

The polynomials that must be solved are always of the form

\[ p(x) = C \cdot (x - r_1) \cdot (x - r_2) \cdots (x - r_N) \]

where \( C \) is a constant. Thus, there are exactly \( N \) roots in \( GF(P) \), counting multiplicities. (Note that all formulae in this appendix are modulus \( P \).) Algorithms such as Berlekamp [39] and Cantor-Zassenhaus [40] are implemented in computer algebra packages and were compared with the simple method developed below. A custom implementation of one of these methods might be faster, but as tested, they were at best 2.7 times slower for our selected values of \( N \) and \( S \).

Following [41], we use a method well-known in computer graphics to simply evaluate the polynomial \( p(x) \) at all values \( x = 0, 1, 2, \ldots, P - 1 \) to find its roots. This method also has the merit of being a task which is easily distributed among the nodes. With \( N = 100 \) and \( S = 10,000 \) less than 0.03 seconds of CPU time were required for the nondistributed version (2.2 GHz processor). However, the computation time is less than 0.0004 seconds CPU time per node when distributed among the 100 nodes. For an extreme case, with \( N = 1,000 \) and \( S = 10,000,000 \), Horner’s rule needed 316 seconds versus 9 seconds of single processor CPU time for our implementation.

**Newton Extrapolation Evaluation for Finding Roots:** To use the Newton extrapolation formula [42], the polynomial \( p(x) \) must first be represented with respect to the Newton (backward) basis polynomials. Our computations will use the formulae specialized for consecutive integers. Let \( n_0(x), n_1(x), \ldots, n_N \) designate the basis Newton polynomials constructed at the values \( x_N = N + k, x_{N-1} = N - 1 + k, \ldots, x_0 = k \).

\[
\begin{align*}
n_{N}k(x) &= 1 \\
n_{N-1}k(x) &= (x + k - N) \\
n_{N-2}k(x) &= (x + k - N)(x + k - (N - 1)); \\
&\vdots \\
n_{0}k(x) &= (x + k)(x + k - (N - 1)); \\
&\vdots (x + k - 2)(x + k - 1)
\end{align*}
\]
The Newton polynomial for \( p(x) \) is then given by
\[
p(x) = \frac{b_{N-0,k}}{0!} + \frac{b_{N-1,k}}{1!} \cdot n_{N-1,k}(x) + \cdots + \frac{b_{N-N,k}}{N!} \cdot n_{0,k}(x)
\]
The backward repeated differences \( b_{j,k} \) are computed by:
\[
b_{j,k} = p(n + k); \quad b_{j-1,k} = b_{j,k} - b_{j-1,k-1} \quad (12)
\]
During the first phase of the algorithm, computing \( p(-N), \ldots, p(1), p(0) \), and then using the recursive formulæ above yields values for \( b_{N,0}, b_{N-1,0}, \ldots, b_{1,0}, b_{0,0} \). Since \( p(x) \) is a polynomial of degree \( N \), \( b_{0,k} = K \), a constant, for all \( k \). This fact allows the computation of \( b_{k,k+1} \) from \( b_{k,k} \) during the second phase of the algorithm by means of the formulæ:
\[
b_{0,k+1} = b_{0,k}; \quad b_{j,k+1} = b_{j-1,k+1} + b_{j,k} \quad (13)
\]
Thus, we have the initially found values \( b_{N,k}, \ldots, b_{0,k} \) with \( k = 0 \) and we proceed to compute these values for \( k = 1, \ldots, P - 1 \) in succession. At each iteration a root is found if \( b_{N,0} = p(k) = 0 \).

The complexity of the algorithm is effectively \( O(N \cdot P) \), when additions modulus \( P \) are performed in unit time.

**Deflation by a Root:** We gain an approximately two-fold speedup with our own method to deflate the polynomial when a root \( r \) is found.

During the second phase of the algorithm, suppose that a root \( r \) is found and that a deflation \( p_{1}(x) = p(x)/(x-r) \) occurs. Then, let our Newton coefficients be represented by \( b_{j,k} \) and \( b_{j,k}^{1} \) for \( p(x) \) and \( p_{1}(x) \) respectively. That \( r \) is a root implies \( b_{N,r}^{1} = 0 \). The values \( b_{*,r-1}^{1} \) are computed from the values \( b_{*,r} \) by:
\[
b_{N-1,j-1,r}^{1} = \frac{1}{(j+1)} \cdot b_{N-1,j-1,r} \quad (14)
\]
Here \( N_{1} = N - 1 \) since \( deg(p_{1}) = deg(p) - 1 \). These relations are obtained by dividing the formula for \( p(x) \) expressed as a Newton polynomial by \((x-r)\) and equating coefficients.

**Coding the Polynomial Solution Algorithm:** The algorithm was coded using the programming language C and tested using several different compilers and computers. It is assumed that any zero roots have been eliminated by shifting of coefficients. We use a single array \( b[j] \) to contain the values \( b_{j,k} \). The array is initialized with the values
\[
b[0] = p(-N), \ldots, b[N-1] = p(-1), b[N] = p(0)
\]
The first set of values \( b_{j,k} \) for \( k = N \) are found by taking recursive differences as developed in (12). This task is accomplished by the following code:

```c
repeat k = k + 1:
  for \( j = 0 \) to \( j = N \) do
    b[j] <= b[j] + b[j - 1]
    if \( b[j] > P \) then \( b[j] <= b[j] - P \)
  od
until \( b[N] = 0 \) or \( k = P - 1 \)
```

The entry \( b[N] \) contains \( p(k) \) at each iteration. When \( b[N] \) is zero, the polynomial is deflated by:
\[
  \text{if} \ (b[N] = 0) \ \text{then}
  \quad \text{for} \ \{ j = 0 \ \text{to} \ j = N - 1 \} \ \text{do}
  \quad \quad b[j] <= 1/(N - j) * b[j] \mod P
  \quad \od
  \quad N <= N - 1
  \quad k <= k - 1
\fi
```
where \( 1/(N - j) \) is looked up in a table. Control returns to the repeat-until loop when \( k < (P - 1) \).

The last two code segments above both contain the test \( b[N] = 0 \). We eliminated an if-then-else structure in favor of the two separate tests in order to create a smaller innermost loop. This doubled the execution speed. Introducing registers to contain \( b[j] \) and \( b[j - 1] \) and rearranging the code to test \( b[j] < 0 \) rather than \( b[j] > P \) in the inner loop also provided gains in speed. Use of optimization options in compilation did reduce runtime, but did not obviate the need for the noted coding changes. Our faith in the current generation of optimizing compilers has been slightly diminished.

**ACKNOWLEDGMENT**

Ideas contributed by Dr. W. E. Clark were crucial in this research. He also tested several existing algorithms for finding roots of polynomials. The assistance of the reviewers is gratefully acknowledged, especially with respect to formalizing and strengthening the privacy results.

**REFERENCES**


Larry A. Dunning (M’79) received the B.S. and M.S. degrees from Wichita State University, Wichita, KS, in 1968 and 1970, respectively, and the Ph.D. degree from North Carolina State University in 1976, all in mathematics. He is Professor Emeritus of Computer Science at Bowling Green State University, Bowling Green, OH. His research interests include error correcting codes, distributed computing, and combinatorics. Dr. Dunning is a member of the ACM and the MAA.

Ray Kresman is a Professor of Computer Science at Bowling Green State University, Bowling Green, OH. He received the Ph.D. degree from Indian Institute of Technology, New Delhi, India. His research interests include computer security, data mining, and distributed systems.