Scalable Mechanisms for Rational Secret Sharing

Varsha Dani *   Mahnush Movahedi *   Jared Saia *

Abstract

We consider the classical secret sharing problem in the case where all agents are selfish but rational. In recent work, Kol and Naor show that, when there are two players, in the non-simultaneous communication model, i.e. when rushing is possible, there is no Nash equilibrium that ensures both players learn the secret. However, they describe a mechanism for this problem, for any number of players, that is an *ε-Nash equilibrium*, in that no player can gain more than ϵ utility by deviating from it. Unfortunately, the Kol and Naor mechanism, and, to the best of our knowledge, all previous mechanisms for this problem require each agent to send $O(n)$ messages in expectation, where n is the number of agents. This may be problematic for some applications of rational secret sharing such as secure multi-party computation and simulation of a mediator.

We address this issue by describing mechanisms for rational secret sharing that are designed for large n. Both of our results hold for $n \geq 3$, and are Nash equilbria, rather than just ϵ-Nash equilbria. Our first result is a mechanism for n-out-of-n rational secret sharing that is *scalable* in the sense that it requires each agent to send only an expected $O(\log n)$ bits. Moreover, the latency of this mechanism is $O(\log n)$ in expectation, compared to $O(n)$ expected latency for the Kol and Naor result. Our second result is a mechanism for a relaxed variant of rational m-out-of-n secret sharing where $m = \Theta(n)$. It requires each processor to send $O(\log n)$ bits and has $O(\log n)$ latency. Both of our mechanisms are non-cryptographic, and are not susceptible to backwards induction.

> *"Three can keep a secret if two of them are dead."*
> - Benjamin Franklin

1   Introduction

Secret sharing is one of the most fundamental problems in security, and is an important primitive in many cryptographic protocols, including secure multiparty computation. Recently, there has been interest in solving *rational secret sharing* [7, 4, 5, 1, 8]. In this setting,

---

there are n selfish but rational agents, and we want to distribute shares of a secret to each agent, and design a protocol for the agents ensures that: (1) if any group of m agents follow the protocol they will all learn the secret; and (2) knowledge of less than m of the shares reveals nothing about the secret. Moreover, we want our protocol to be a *Nash equilibrium* in the sense that no player can improve their utility by deviating from the protocol, given that all other players are following the protocol.

Unfortunately, all previous solutions to this problem require each agent to send $O(n)$ messages in expectation, and so do not scale to large networks. Rational secret sharing is a primitive for rational multiparty computation, which can be used to compute an arbitrary function in a completely decentralized manner, without a trusted external party. A typical application of rational multiparty computation might be to either run an auction, or to hold a lottery to assign resources in a network. It is easy to imagine such applications where the number of players is large, and where it is important to have algorithms whose bandwidth and latency costs scale well with the number of players. Moreover, in a game theoretic setting, standard tricks to circumvent scalability issues, like running the protocol only on a small subset of the players, may be undesirable since they could lead to increased likelihood of bribery attacks.

In this paper, we address this issue by designing scalable mechanisms for rational secret sharing. Our main result is a protocol for rational n-out-of-n secret sharing that (1) requires each agent to send only an expected $O(\log n)$ bits; and (2) has $O(\log n)$ expected latency. We also design scalable mechanisms for a relaxed variant of m-out-of-n rational secret sharing in the case where m is $\Theta(n)$. We note however that we pay for these improvements by requiring the payers to send $O(\log n)$ rather than a constant number of bits per round.

1.1   The Problem

Shares of a secret are to be dealt to n rational but selfish players, who will later reconstruct the secret from the shares. The players are *learning-preferring,* in the sense that each player prefers every outcome in which he learns the secret to any outcome in which he does not learn the secret. We note that in some previous work [7, 1] it is further assumed that the players are *competitive*: they prefer that others do not learn secret. However, this assumption is used mainly for the purpose of proving lower bounds, and is omitted in the upper bounds. We will *not* make this additional assumption.

The secret is an arbitrary element of a large (fixed) finite field $\mathbb{F}_q$. At the beginning of the game, a dealer provides the shares to the players. The dealer has no further role in the game. The players must then communicate with each other in order to recover the secret.

Communication between the players is *point-to-point* and through secure private channels. In other words, if player A sends a message to player B, then a third player C is not privy to the message that was sent, or indeed even to the fact of a message having been sent. Communication is *synchronous* in that there is an upper-bound known on the maximum amount of time required to send a message from one player to another. However, we assume *non-simultaneous* communication, and thus allow for the possibility of *rushing*, where a player may receive messages from other players in a round before sending out his

own messages.

Our goal is to provide protocols for the dealer and rational players such that the players following the protocol can reconstruct the secret. Moreover, we want a protocol that is *scalable* in the sense that the amount of communication and the latency of the protocol should be a slow growing function of the number of players.

1.2 Related Work

Since its introduction by Halpern and Teague in [5], there has been significant work on the problem of rational secret sharing, including results of Halpern and Teague [5], Gordon and Katz [4], Abraham et al. [1], Lysyanskaya and Triandopoulos [8] and Kol and Naor [7]. All of this related work except for [7], assumes the existence of simultaneous communication, either by broadcast or private channels. Several of the protocols proposed [4, 1, 8] make use of cryptographic assumptions and achieve equilibria under the assumption that the players are computationally bounded. The protocol from [1] is robust to coalitions; and the protocol from [8] works in the situation where players may be either rational or adversarial.

The work of Kol and Naor [7] is closest to our own work. They show that in the non-simultaneous broadcast model (*i.e.*, when rushing is possible), there is no Nash equilibrium that ensures all agents learn the secret, at least for the case of two players. They thus consider and solve the problem of designing an ϵ-Nash equilibrium for the problem in this communication model. An ϵ-Nash equilibrium is close to an equilibrium in the sense that no player can gain more than ϵ utility by unilaterally deviating from it. Furthermore, the equilibrium they achieve is *everlasting* in the sense that after any history that is consistent with all players following the protocol, following the protocol continues to be an ϵ-Nash equilibrium. As we have already discussed, our protocols make use of several clever ideas from their result.

The impossibility of a Nash equilibrium for two players carries over to the setting with secure private channels, since there is no difference between private channels and broadcast channels when there are only two players. However, one might hope that the algorithm of Kol and Naor [7] could be simulated over secure private channels to give an everlasting ϵ-Nash equilibrium. Unfortunately, simulation of broadcast over private channels is expensive, requiring each player to send $\Theta(n)$ messages per round.

In [2] we overcame this difficulty, providing a *scalable* algorithm for rational secret sharing, in which each player only sends $O(1)$ bits per round and the expected number of rounds is constant (although each round takes $O(\log n)$ time). Moreover, following the protocol is an ϵ-Nash equilibrium. Unfortunately, a certain bad event with small but constant probability caused some players, when they recognized it, to deviate from the protocol so that the equilibrium is not everlasting. This paper is the full version of [2]. However, we improve on the work in [2] in two ways. First, we remove all probability of error for n-out-of-n secret sharing, and improve the probability of error for m-out-of-n from a constant to an inverse polynomial. Second, we show that our new protocol is a Nash equilibrium, not just an ϵ-Nash equilibrium, as long as $n \geq 3$.

1.3   Our Results

The main result of this paper is presented as Theorem 1. This theorem builds on work from our extended abstract in [2]. It also improves on this result in two ways. First, our new protocol eliminates the probability of failure when compared with the protocol in [2]. Second, our new protocol has the added advantage of being a Nash equilibrium, not merely an ϵ-Nash equilibrium.

Theorem 1. Let $n \geq 3$. There exists a protocol for rational n-out-of-n secret sharing with the following properties.

- The protocol is an everlasting Nash equilibrium in which all players learn the secret.

- The protocol, in expectation, requires each player to send $O(\log n)$ bits, and has latency $O(\log n)$.

We also consider the problem of m-out-of-n rational secret sharing for the case where $m < n$. Designing scalable algorithms for this problem is challenging because of the tension between reduced communication, and the need to ensure that *any* active set of m players can reconstruct the secret. For example, consider the case where each player sends $O(\log n)$ messages. If $m = o(n/\log n)$, even if the set of active players is chosen *randomly*, it is likely that there will be some active player that will never receive a message from any other active player. Moreover, even if $m = \Theta(n)$, if the set of active players is chosen in a worst case manner, it is easy to see that a small subset of the active players can easily be isolated so that they never receive messages from the other active players, and are thus unable to reconstruct the secret.

Despite the difficulty of the problem, scalable rational secret sharing for the m-out-of-n case may still be of interest for applications like the Vanish peer-to-peer system [3]. To determine what might at least be possible, we consider a significantly relaxed variant of the problem. In particular, we require $m = \Theta(n)$ and that the set of m active players be chosen independently of the random bits of the dealer. In this setting we prove the following.

Theorem 2. Let $n \geq 3$. For any fixed positive k, λ, and threshold τ, there exists a protocol for rational secret sharing with absent players, which with probability at least $1 - \frac{1}{n^k}$ has the following properties, provided that the subset of m active players is chosen independently of the random bits of the dealer:

- The protocol is a Nash equilibrium.

- The protocol ensures that if at least a $(\tau + \lambda)$ fraction of the players are active, (i.e. $m/n \geq \tau + \lambda$) then all active players will learn the secret; and if less than a $(\tau - \lambda)$ fraction of the players are active, (i.e. $m/n \leq \tau - \lambda$) then the secret can not be recovered

- The protocol requires each player to send $O(\log^2 n)$ bits, and has latency $O(\log n)$

4

This is an improvement to the $\Theta(n)$-out-of-n result we proved in [2], in the sense that the probability of error in [2] is a small constant, but here it is $1/n^k$. However, we cannot completely eliminate the probability of failure.

1.4 Our Approach

The difficulty in designing a Nash equilibrium in a communication model where rushing is possible, is that the last player to send out his share has no incentive to actually do so. He already has the shares of all the other players and can recover the secret alone. To get around this, it is common (see [5, 1, 4, 8, 7]) for the protocol to have a number of fake rounds designed to catch cheaters. The uncertainty in knowing which is the "definitive" round, during which the true secret will be revealed causes players to cooperate.

In the work of Kol and Naor [7] this uncertainty is created by dealing one player only enough data to play until the round preceding the definitive one. Thus, there is a single "short" player and $n-1$ "long" players. None of the players know whether they are short or long. The long players must broadcast their information every round, since they cannot predict the definitive round in advance. The short player knows the definitive round in advance, but has no information about the secret. In the definitive round the short player is the last to speak so that he (and all the other players) receives the shares of all the long players and can recover the secret. His failure to broadcast a message is what cues the other players to the end of the game, and they too can recover the secret. Moreover, having learned the secret, the short player cannot pretend that he actually had a share for that round as the messages sent by all the players are verified by a tag and hash scheme (see, e.g., [11, 9, 7]). In fact, it is the small but positive chance of cracking the tag and hash scheme that results in this being an ϵ-Nash equilibrium rather than a Nash equilibrium.

Here, we also use short and long players. However we introduce two novel techniques to ensure scalable communication and to ensure a Nash equilibrium. The first technique is to arrange players at the leaves and nodes of a complete binary tree, and require that the players only communicate with their neighbors in the tree. The assignment of players to the leaves is independently random every round, and their assignment to internal nodes is related to their assignment to leaves by a labeling of the tree that is common knowledge. Every round of the game, information travels up to the root where it is decoded and then travels back down again to the leaves. The short players are the parents of the leaves in the definitive round, so that now about half the players are short players.

The second main idea is that we make use of an iterated secret sharing scheme over this tree in order to divide up shares of secrets among the players. This scheme is similar to that used in recent work by King and Saia [6] on the problem of scalable Byzantine agreement, and suggests a deeper connection between the two problems.

As in previous works [11, 9, 7] we use a tag-and-hash scheme to ensure that players cannot forge messages in the protocol. We note however, that unlike in previous work, our use of the verification scheme is such that even by breaking it, players who have learned the secret cannot prevent other players from learning it as well. Thus, in our case the small probability of forging messages without detection does *not* translate into the protocol being an ϵ-Nash

equilibrium. Instead we show that our protocol is a Nash equilibrium for all the players.

1.5 Paper Organization

The rest of this paper is laid out as follows. In Section 2, we give notation and preliminaries. In Section 3, we describe our algorithm for scalable n-out-of-n secret sharing. In Section 4, we analyze this algorithm; the main result of this section is a proof of Theorem 1. In Section 5, we give our algorithm and analysis for scalable m-out-of-n secret sharing where $m = \theta(n)$; the main result of this section is a proof of Theorem 2. Finally in Section 6, we conclude and give directions for future work.

2 Notation and Preliminaries

The secret to be shared is an arbitrary element of a set $\mathcal{S}$. There are n players with distinct player IDs in $[n] = \{1, 2, \ldots n\}$. During the course of the algorithm, we will want to do arithmetic manipulations with player IDs and shares of the secret, including adding in, or multiplying by random elements to preserve secrecy. In order to be able to do these sorts of manipulations, we embed the sets $\mathcal{S}$ and $[n]$ into a finite field $\mathbb{F}$ of size $q > \max\{n, |\mathcal{S}|\}$. The latter embedding will be the canonical one; the former may be arbitrary, but is assumed to be known to all parties.

The messages sent by players in the algorithm will be elements of $\mathbb{F}$. The length of any such message is $\log q = \Omega(\log n)$. Since our goal is to provide a scalable algorithm we cannot afford the message lengths to be much bigger than that. We will choose $\mathbb{F}$ to be a prime field of size $q = O(n)$. We remark that although generally $\mathcal{S}$ is of constant size, we can tolerate $|\mathcal{S}| = O(n)$.

2.1 Utility Functions

We will denote the utility function of player j by u_j. As mentioned before, we assume that the players are learning preferring, *i.e.*, each player prefers any outcome in which he learns the secret to every outcome in which he does not learn the secret. More formally, for outcome $\mathbf{o}$ of the game, let $R(\mathbf{o})$ denote the set of players who learn the secret. If $\mathbf{o}$ and $\mathbf{o}'$ are outcomes of the game such that $j \in R(\mathbf{o}) \setminus R(\mathbf{o}')$, then $u_j(\mathbf{o}) > u_j(\mathbf{o}')$. As in [7], we denote

$$U_j^+ = \max\{u_j(\mathbf{o}) \mid j \in R(\mathbf{o})\}$$
$$U_j = \min\{u_j(\mathbf{o}) \mid j \in R(\mathbf{o})\}$$
$$U_j^- = \max\{u_j(\mathbf{o}) \mid j \notin R(\mathbf{o})\}.$$

Thus U_j^+ is the utility to player j of the best possible outcome for j, U_j is the utility to j of the worst possible outcome in which j still learns the secret, and U_j^- is the best possible

utility to j when he does not learn the secret. By the learning-preferring assumption, we have for all j,

$$U_j^+ \geq U_j > U_j^-.$$

We will denote by $\mathcal{U}$, the quantity

$$\mathcal{U} := \max_{j \in [n]} \frac{U_j^+ - U_j^-}{U_j - U_j^-}.$$

Note that $\mathcal{U} \geq 1$. We assume that $\mathcal{U}$ is constant, *i.e.*, that it does not depend on n.[1]

We also assume that the utilities are such that *a priori* the players have an incentive to play the game rather than just guess the secret at random. In other words, we require that

$$\mathcal{U} < |\mathcal{S}|. \tag{1}$$

We have said earlier that $|S|$ may be as big as n. If that is the case, then (1) is trivially satisfied since $\mathcal{U} = O(1)$. When $\mathcal{S}$ is of constant size however, (1) is a genuine constraint.

2.2 Game Theoretic Concepts

In this section we review some game theoretic solution concepts.

Recall that an n-tuple of strategies for an n player game is called a *Nash equilibrium* if no player has an incentive to unilaterally deviate from the equilibrium strategy, when all others are following it.

In games of incomplete information which have multiple rounds, there is the further question of whether the players are forced to commit to their strategies before the start of the game or whether they have the option to change strategies in the middle of the game, after some rounds have been played and they may learn some new information. Kol and Naor [7] defined a Nash equilibrium to be *everlasting* if after any history that is consistent with all players following the equilibrium strategy, it is still true (despite whatever new information the players may have learned over that history) that a player choosing to deviate unilaterally cannot gain in expectation, *i.e.*, following the prescribed strategy remains a Nash equilibrium. This is a stronger concept than the usual Nash equilibrium, where the strategies are committed to up front.

3 Algorithm For All Players Present

We now describe our scalable mechanism for n-out-of-n secret sharing. First, in Section 3.1 we describe the communication tree that is used by the dealer and players. An informal description of the mechanism follows in Section 3.2. The formal descriptions of the dealer's and players' protocols appear respectively as Algorithms 1 and 4.

---

[1]Technically, we can achieve scalable (polylog) communication even if we allow $\mathcal{U}$ to be as big as polylog(n)
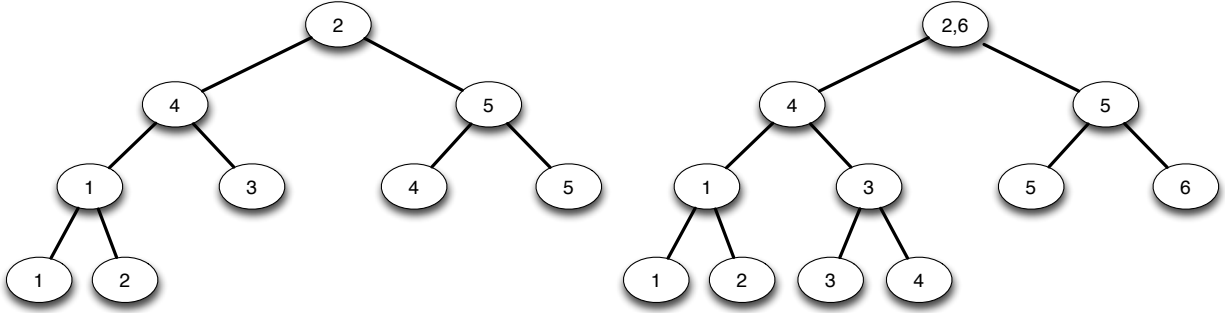
Figure 1: Communication trees for five players and six players

3.1 The Communication Tree

Recall that a complete binary tree is a binary tree in which all the internal nodes have exactly two descendants, all the leaves are at the two deepest levels, and the leaves on the deepest level are as far left as possible.

Our communication tree is a complete binary tree with n leaves. The leaves will be labelled 1 to n from left to right. Next every internal node which is a parent of two leaves is labelled with the odd label from among its two children. Finally, the remaining internal nodes are labelled in order with even numbers, proceeding top to bottom and left to right, starting with 2 at the root. If n is odd, then each even number appears at some internal node. If n is even, we will place the last even number, n at the root, along with 2 (so the root will have two labels.) The tree thus labelled has the following properties:

- Every even label occurs at some internal node. (Note that if n is odd, there will be an odd label that occurs only at a leaf and not at any internal node. This will not matter.)

- No even labelled internal node has an odd labelled node above it.

- Every path from root to leaf has exactly one odd label (the same odd label may occur once or twice on the path.)

Figure 1 illustrates the labelling scheme for five and six players.

3.2 Our Algorithm

The dealer is active only once at the beginning of the game, and during this phase of the game the players' inputs are prepared.

The dealer independently samples two random variables X and Y from a geometric distribution with parameter β (to be determined later). X will be the definitive iteration, or the round of the game in which the true secret is revealed. Y will be the amount of padding on the long players' input. Note we have two kinds of players: short players will receive

8

Algorithm 1 Dealer's Protocol

$\mathbb{F}$ field of size q (to represent messages in the algorithm) n players with distinct identifiers in $[n] \hookrightarrow \mathbb{F}$, $\beta \in (0,1)$: geometric distribution parameter. Complete binary tree with n leaves, labelled as described in Section 3.1 known to everyone.

1. Choose X, Y, independently from a geometric distribution with parameter β. Round X is the definitive one. Short players will receive full input for $X - 1$ rounds and partial input for round X. Long players will receive full input for $X + Y - 1$ rounds and partial input for round $X + Y$. For convenience we will create all the inputs for $X + Y$ rounds, and truncate them appropriately before sending them to the players.

2. For each round t between 1 and $L = X + Y$:

   - If $t < X + Y$, choose a random permutation $\pi_t \in S_n$. If $t = X + Y$ choose a permutation π_L which is random subject to the constraint that all the long players (determined by π_X) are assigned to odd labels under π_L. For round t player j will be assigned to all nodes marked $\pi_t(j)$ in the tree.

   - If $t = 1$, $m_1 = 0$ // (Otherwise m_t was set in the previous round)

   - For every player j, $(\pi_t(j), (\pi_t^{-1}(i)|\text{node } i \text{ is a neighbor of node } \pi_t(j) \text{ in the tree.}))$, is a tuple of elements of $\mathbb{F}$ representing j's position and the identities of his neighbors in the tree for round t. $P_t^j = (\pi_t(j) + m_t, ((\pi_t^{-1}(i) + m_t|\text{node } i \text{ is a neighbor of node } \pi_t(j) \text{ in the tree.}))$ is a masked version.

   - Choose a random mask $m_{t+1} \in \mathbb{F}$ (for the **next** round.)

   - Create shares of m_{t+1} by calling *RecursiveShares* (root, m_{t+1}).

   - If $t = X$ $s_t \leftarrow$ true secret.
     Otherwise, $s_t \leftarrow$ random element of $\mathcal{S}$

   - Create shares of s_t by calling *RecursiveShares* (root, s_t).

   - Create tags and verification functions for all the messages to be sent in round t

   - For each player j, j's (full) input I_t^j for round t consists of P_t^j, shares of m_{t+1} and s_t corresponding to node $\pi_t(j)$, tags to authenticate all messages to be sent by j and verification vectors for all the messages to be received by j. Partial input $\tilde{I}_t^j$ consists of all of the above except the authentication tags for sending messages to your children (in the down-stage).

3. Identify the short players as those players j who are at odd numbered nodes in the definitive iteration, *i.e.*, $\pi_X(j)$ is odd.

4. For each short player j, send j the list $I_1^j, \ldots I_{X-1}^j, \tilde{I}_X^j$.

5. For each long player j, send j the list $I_1^j, \ldots I_{L-1}^j, \tilde{I}_L^j$.

Algorithm 2 *RecursiveShares* (node w , $\mathbb{F}$-element y):

n-leaf complete binary tree global data structure V; for node w', $V^{w'}$ denotes the location for the data associated with w'.

Initially called with the root node and the value for which shares are to be created, this function populates V with intermediate values. The values at the leaves are the shares for the players at the corresponding leaves of the communication tree.

1. $V^w \leftarrow y$.

2. If w has children $\ell(w)$ and $r(w)$:

   (a) Choose random slope μ from field $\mathbb{F}$.

   (b) Let f be the line with slope μ and y-intercept y.

   (c) *RecursiveShares*($\ell(w), f(-1)$).

   (d) *RecursiveShares*($r(w), f(1)$).

Algorithm 3 Create Authentication Data ($\mathbb{F}$-element y,): // y is the message to be transmitted

1. Choose $a \in \mathbb{F}$ and $b \in \mathbb{F}^* = \mathbb{F} \setminus \{0\}$ independently, uniformly at random.

2. $c = y + b * a$.

3. a is the tag, to be given to the sender of message y, (b, c) is the verification vector, to be given to the recipient of the message y.

Algorithm 4 Protocol for Player j

S=0; M=0

If at any time you receive spurious messages (messages not expected uder the protocol), ignore them.

On round t:

> *Up-Stage*:
>
>> 1. $m_t = M$
>>
>> 2. Subtract m_t from all elements of P_t^j to find out your position in the tree and the identities of your neighbors for round t.
>>
>> 3. (as player at a leaf) Send your shares of s_t and m_{t+1} along with their tags to your parent in the tree.
>>
>> 4. (as player at an internal node)
>>
>>> (a) Receive (intermediate) shares of s_t and m_{t+1} and tags from left and right chidren. Use the appropriate verification vectors to check that correct messages have been sent. If a fault is detected (missing or incorrect message) output S and quit.
>>>
>>> (b) For each of s_t and m_{t+1}: interpolate a degree 1 polynomial f from $(-1, \text{left-share})$ and $(1, \text{right-share})$. Evaluate $f(0)$. This is your share.
>>>
>>> (c) If you are not at the root, send the above reconstructed shares of s_t and m_{t+1} to your parent(s) along with the appropriate tags. If you *are* at the root, these shares are the actual values of s_t and m_{t+1}.
>
> *Down-Stage*:
>
>> 1. If you are at the root, set $S = s_t$ and $M = m_{t+1}$ and send these values along with authentification tags to your left and right children.
>>
>> 2. Else
>>
>>> (a) (as a non-root internal node) Receive s_t and m_{t+1} and tags from your parent and use verificaton vectors to check them. If fault detected, output S and quit.
>>>
>>> (b) Set $S = s_t$ and $M = m_{t+1}$.
>>>
>>> (c) Send s_t and m_{t+1} to your children along with the appropriate tags. If you are a short player and have no authentication tags, output s_t and quit.
>
> $t \leftarrow t + 1$

enough input to last for X rounds of the game while long players will receive enough input to last for $X + Y$ rounds of the game. The partitioning of players into short and long will be random, and the players themselves will not know which are which. This is critical in our analysis as is discussed in Section 4.

Communication between the players in our protocol will be restricted to sending messages to their neighbors in the communication tree. In order not to reveal which players are the short players, the players will be reassigned to new positions in the tree in each round. This is accomplished by choosing a random permutation of the players each round and assigning them to labelled nodes of the tree according to it. The short players are the ones who are at odd labelled nodes in the definitive round.

Since the players must be at different nodes in the tree each round, their input must contain this information. At the same time, the positions of the players for all the rounds cannot be revealed up front, since this may give away information about who the short players are. A naive idea to solve this problem is, in each round, to distribute shares of the permutation for the next round. Then during each round, the players could reconstruct the permutation from the shares and use it to reposition for the next round. Unfortunately, there is a problem with this approach. To represent permutations of n symbols, we need a field of size at least $n!$. To transmit elements of such a field, players would need to send messages of length $\log(n!) \sim n \log n$. This is unacceptable if we desire scalability.

To get around this problem, we note that it is not really necessary for players to know the entire permutation. Each player only needs to know its own position and the identities of its neighbors. We only need a field of size order n to encode this, and so, symbols of this field may be transmitted with messages of length $\log n$. Since it is dificult via share reconstruction to transmit different messages to the leaves of the tree, we simply provide each player with a list of positional data for the entire game. But in order that players do not know their positional data for a round before actually getting to that round, this data is masked by adding in a random element of the field. Positional data for the first round is sent unmasked. The players also receive iterated shares of the masks for the next round. Thus, in each round, players reconstruct a mask, and use it to unmask the positional data and reposition themselves for the next round.

For each round, the full input consists of the following:

- iterated shares of a purported secret (the true secret in the definitive round);

- masked versions of positional data for the current round (position and identities of neighbors in the tree);

- shares of the mask for the *next round* of positional data;

- tags for all the messages to be sent; and

- verification vectors for all the messages to be received.

The iterated shares the players receive are constructed by starting with the symbol to be reconstructed at the root and recursively constructing 2-out-of-2 Shamir shares down the
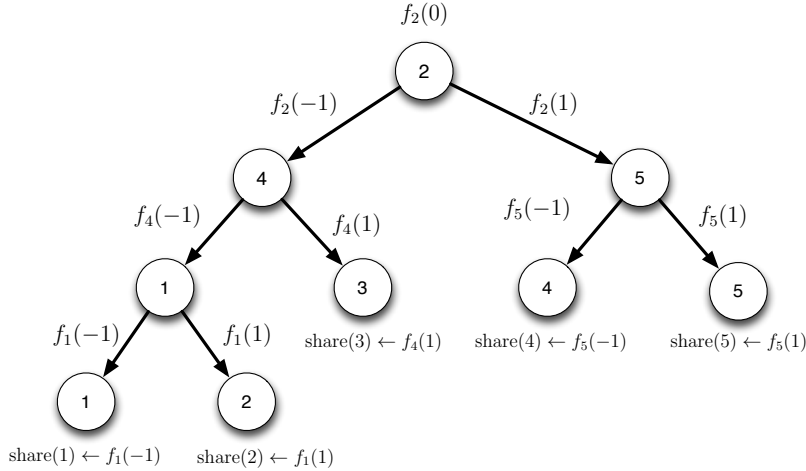
Figure 2: Construction of the iterated shares

tree, all the way down to the leaves. The shares at the leaves are the iterated shares the players receive. See Figure 2 and Algorithm 2 for details of how the iterated shares are constructed. At reconstruction time, shares are sent up the tree. At each internal node, a pair of shares received from the two children is reconstructed into a degree 1 polynomial which is used to obtain the value to be sent further up the tree. At the root, the original symbol is reconstructed and transmitted down the tree. Note that the advantage of this scheme over simply using n-out-of-n Shamir shares is that the size of the messages does not increase as the messages are transmitted up the tree.

As mentioned earlier, round X is the definitive round, when the encoded symbol is the true secret. Short players receive full input for every round prior to this round. For round X they only receive partial input. Long players receive input for $X + Y$ rounds. However, they, too receive only partial input for their last block of input. Otherwise, a player would be able to distinguish whether or not he is a short player by looking at his last block of input. Here, partial input consists of all of the pieces of data from the full input, *except* the tags to send the decoded message to your children in the down stage of the round.

Since, in the definitive round the short players (with odd labels) are in the level above the leaves, and all the long players are at internal nodes higher than that in the tree, the long players have learned the secret before the short players, although since they have input for more rounds of the game, they do not know (*i.e.*, cannot guess) that it is the definitive round, and that the secret they have learned is in fact the true secret. Thus they send the secret down the tree, and eventually it gets to the short players. Thus the short players learn the secret as well. Since they have no more input they know that the game is over and the secret is the true one. However, since they do not have any more authentication data, they cannot gain by remaining in the game and trying to fool the others into thinking that the secret has not yet been reconstructed. Finally, when the long players do not receive a message at the end of the definitive iteration, they too realize the game has ended and

output the correct secret.

4 Analysis of Algorithm for All Players Present

In this section we will prove Theorem 1, which shows that the secret sharing scheme we have described is in fact a scalable n-out-of-n secret sharing scheme, and that it is an everlasting Nash equilibrium in which all the players learn the secret.

We begin by showing that our rescursive scheme for encoding a symbol into n iterated shares (Algorithm 2) is an n-out-of-n scheme.

Lemma 3. Let $\sigma \in \mathbb{F}$ be a symbol that is encoded into n iterated shares, $\sigma_1, \ldots \sigma_n$, by Algorithm 2. Then σ can be decoded from all n of the shares, but knowledge of fewer than n of the shares reveals no information about σ

Proof. That σ can be decoded from all n shares follows easily from the fact that two points on a line determine it. Since the value at a node is the y-intercept of a line passing through the points $(-1,$ left-child value$)$ and $(1,$ right-child value$)$, it can be reconstructed using interpolation. Starting with the shares $\sigma_1 \ldots \sigma_n$ at the leaves of the tree, we can reconstruct the values bottom-up, and the value at the root is σ since this is exactly the reverse of the process used to create these shares.

To see why fewer than n shares give us no information about σ, observe that the values at the two children of the root were created by choosing a random slope in $\mathbb{F}$ for a line with y intercept σ, and then evaluating that line at -1 and 1. Both of these values together determine the line, but a single one of them does not eliminate any line as a possibility. Thus the values at the children of the root *individually* contain no information about the value of the root, and in order to decode the value at the root, we need both the values at its children. But now, this reasoning applies recursively to all the internal nodes, relative to their children. Suppose there is a leaf of the tree at which the share is missing. Then the share of its parent cannot be decoded because it is equally likely to be any element of the $\mathbb{F}$. This propagates up to its grandparent, and then its great-grandparent and so on all the way to the root, so that the root cannot be decoded. Thus, if even one of the shares is missing, the remaining shares provide no information about the value of σ. $\square$

Next, we discuss the tag-and-hash verification scheme used in the protocol. This scheme makes it hard for a sender to successfully fool the intended recipient of a message by sending a faked message. At the same time, it does not give the recipient of the message any information about the message prior to receiving it. Such schemes have been used before (see *e.g.* [11, 9, 7]); we include the following proposition for completeness. See Lemma 1 of [9] for the proof.

Proposition 4. The verification scheme (Algorithm 3) has the following properties:

1. The verification vector contains no information about the message, *i.e.*, the probability of correctly guessing the message given the verification vector is the same as the unconditional probability of guessing the message

2. The probability that a faked message will satisfy the verification function is $\frac{1}{q-1}$

We will now focus our attention on showing that it is a Nash equilibrium for all the players to follow our protocol. Consider player j and suppose that all other players are committed to following the protocol. The next lemma gives a necessary criterion for j to have an incentive to cheat.

Lemma 5. If all other players are following the protocol, player j prefers to also follow the protocol, unless his probability of successfully cheating is at least $\frac{U_j - U_j^-}{U_j^+ - U_j^-}$.

Proof. Suppose j is considering deviating from the protocol. We will consider the deviation to be successful if either j learns the secret right away, with or without being caught, or he does not get caught and is therefore still in a position to learn it later. The deviation will have failed if it is detected, causing the game to end without j learning the secret. Let p_j be the probability that the deviation succeeds. The maximum utility that j can get is U_j^+. With probability $1 - p_j$, the game ends without j learning the secret, in which case the maximum payoff possible is U_j^-. Thus a player's expected utility from cheating while everyone else follows the protocol is at most $p_j U_j^+ + (1 - p_j) U_j^-$.

On the other hand, if everyone else follows the protocol, then following the protocol guarantees a utility of at least U_j. Thus the protocol will be a Nash equilibrium if

$$U_j > p_j U_j^+ + (1 - p_j) U_j^-$$

Rearranging terms, we have a Nash equilibrium if

$$p_j < \frac{U_j - U_j^-}{U_j^+ - U_j^-}. \qquad \square$$

When and how might a player cheat? We note that since players are not required to commit to their strategy before starting the game, and since the progression of the game reveals information, a player may as well defer his decision to cheat in a future round until that future round. Thus, at any given time, the decision facing the player is whether to cheat in the current round. In order to weigh the benefits of such a decision, the player needs an estimate of whether the current round is likely to be the definitive one.

As remarked earlier, the purpose of having short and long players is to create uncertainty about when the definitive round of the game is, until it is too late to gain from this information.

The players know that X is chosen from a geometric distribution with parameter β. Thus, *a priori* the probability that X takes on any particular value is at most β, the most likely being $X = 1$, whose probability is exactly β. As the game progresses, players receive partial information about the value of X; as soon as they receive their inputs they can eliminate all values of X larger than their input length, if the game did not end on the first round, they learn that $X \neq 1$ and so on. Clearly, when a player reaches his last block of input, he knows that the current round is definitive. The next lemma shows that until that stage, a player's estimate that the current round is definitive remains small.

15

Lemma 6. Let j be a player who initially received input for $k > 1$ rounds of the game. and let $1 \leq t < k$ be the current round. Then j's estimate of the probability that the current round is definitive, conditioned on all the information he has learned, is at most 2β.

Proof. Let L_j denote the random variable which is the initial input length of player j. Then we know that

$$L_j = \begin{cases} X & \text{if } j \text{ is a short player} \\ X + Y & \text{if } j \text{ is a long player} \end{cases} \tag{2}$$

Also, let $\mathcal{L}_j$ denote the event that j is a long player and $\mathcal{L}_j^{\mathrm{C}}$ the event that j is a short player. By hypothesis, the current round is $t \geq 1$, and player j received an initial input of length $k > t$. What information does player j know in round t?

- Since his initial input was of length k he knows that $L_j = k$ and $X \leq k$ and moreover, that $X = k$ if and only if $\mathcal{L}_j^{\mathrm{C}}$.

- Since the game has entered round t he knows that $X \geq t$.

- He knows his position $\pi_t(j)$ and the identities of his neighbors in round t

- He also has learned s_t, m_{t+1} and using the latter to unmask his positional data, he knows $\pi_{t+1}(j)$ and the identities of his neighbors in round $t + 1$. Technically, he learns these just prior to his turn in the downstage in round t, but this is fine, as we will argue later that no player ever has any reason to cheat during the upstage of a round.

We note that knowing s_t does not benefit player j in any way as far as estimating the probability that $X = t$ goes, since s_t is equally likely to be any element of $\mathcal{S}$, independently of X. Similarly, knowing the identities of his neighbors does not affect his estimate, since all other players are equally likely to be his neighbors independently of X.

On the other hand, knowing $\pi_t(j)$ and $\pi_{t+i}(j)$ does affect the estimate. By construction:

- In the definitive iteration, short players have odd labels, and long players have even labels; and

- Each player has an odd label in his last round of input

Thus if $\pi_t(j)$ is odd, then player j knows that the current round is not definitive, since if $X = t$, then $k > t$ implies that j is a long player and should have an even label. In particular, conditioned on everything he knows, $\Pr(X = t) = 0$. Since $2\beta > 0$ the lemma is proved in this case.

For the remainder of the proof we will assume that $\pi_t(j)$ is even and denote this event $\mathcal{E}_t$.

Now what about $\pi_{t+1}(j)$? If $k = t + 1$, then we know that $\pi_{t+1}(j)$ is odd by construction and knowing this contains no additional information over knowing $L_j = k$. On the other hand when $k > t + 1$, if $\pi_{t+1}(j)$ is odd, then player j knows that X cannot be $t + 1$ and this affects the probability that $X = t$.

Let $b \in \{0, 1\}$ be the observed parity of $\pi_{t+1}(j)$ and let $\mathcal{E}^b_{t+1}$ denote the event that the parity of $\pi_{t+1}(j)$ is b. Note that if $k = t + 1$ we must have $b = 1$.

Let $P_{j,t}$ be player j's estimate that the current round, t, is definitive, conditioned on everything he knows. Then

$$P_{j,t} = \Pr(X = t \mid L_j = k \ \wedge \ t \leq X \leq k \ \wedge \ \mathcal{E}_t \ \wedge \ \mathcal{E}^b_{t+1})$$

$$= \frac{\Pr(X = t \ \wedge \ L_j = k \ \wedge \ \mathcal{E}_t \ \wedge \ \mathcal{E}^b_{t+1})}{\Pr(L_j = k \ \wedge \ t \leq X \leq k \ \wedge \ \mathcal{E}_t \ \wedge \ \mathcal{E}^b_{t+1})}$$

$$\leq \frac{\Pr(X = t \ \wedge \ L_j = k \ \wedge \ \mathcal{E}_t \ \wedge \ \mathcal{E}^b_{t+1})}{\Pr(L_j = k \ \wedge \ X \in \{t, k\} \ \wedge \ \mathcal{E}_t \ \wedge \ \mathcal{E}^b_{t+1})}.$$

where the inequality follows from the fact that the event $X \in \{t, k\})$ is a subset of the event $t \leq X \leq k$.

Now, if the current round *is* definitive *i.e.*, $X = t$, then j is not a short player, and $L_j = X + Y$. So the event $X = t \ \wedge \ L_j = k \ \wedge \ \mathcal{E}_t \ \wedge \ \mathcal{E}^b_{t+1}$ is the same as the event $\mathcal{L}_j \ \wedge \ X = t \ \wedge \ Y = k - t \ \wedge \ \mathcal{E}^b_{t+1}$. Note that $\mathcal{E}_t$ is implied by $\mathcal{L}_j \ \wedge \ X = t$ and can therefore be dropped.

For the denominator, the event $L_j = k \ \wedge \ X \in \{t, k\} \ \wedge \ \mathcal{E}_t \ \wedge \ \mathcal{E}^b_{t+1}$ can be split into the union of disjoint events $\mathcal{L}^C_j \ \wedge \ X = k \ \wedge \ \mathcal{E}_t \ \wedge \ \mathcal{E}^b_{t+1}$ and $\mathcal{L}_j \ \wedge \ X = t \ \wedge \ Y = k - t \ \wedge \ \mathcal{E}^b_{t+1}$. The latter summand is the same as the numerator, and loses the $\mathcal{E}_t$ term for the same reason.

Making these substitutions in the above expression, we get

$$P_{j,t} \leq \frac{\Pr(\mathcal{L}_j \ \wedge \ X = t \ \wedge \ Y = k - t \ \wedge \ \mathcal{E}^b_{t+1})}{\Pr(\mathcal{L}^C_j \ \wedge \ X = k \ \wedge \ \mathcal{E}_t \ \wedge \ \mathcal{E}^b_{t+1}) + \Pr(\mathcal{L}_j \ \wedge \ X = t \ \wedge \ Y = k - t \ \wedge \ \mathcal{E}^b_{t+1})}$$

The random variables X, Y and the indicator that j is a long player are independent. Thus the numerator of the above expression becomes

$$\Pr(\mathcal{L}_j \ \wedge \ X = t \ \wedge \ Y = k - t \ \wedge \ \mathcal{E}^b_{t+1})$$

$$= \Pr(\mathcal{E}^b_{t+1} | \mathcal{L}_j \ \wedge \ X = t \ \wedge \ Y = k - t) \Pr(\mathcal{L}_j) \Pr(X = t) \Pr(Y = k - t)$$

$$= \Pr(\mathcal{E}^b_{t+1} | \mathcal{L}_j \ \wedge \ X = t \ \wedge \ Y = k - t) \frac{\lfloor n/2 \rfloor}{n} (1 - \beta)^{t-1} \beta (1 - \beta)^{k-t-1} \beta$$

$$= \Pr(\mathcal{E}^b_{t+1} | \mathcal{L}_j \ \wedge \ X = t \ \wedge \ Y = k - t) \frac{\lfloor n/2 \rfloor}{n} (1 - \beta)^{k-2} \beta^2$$

Similarly we tackle the first term in the denominator. Since X and all the π_i are independent and $k \neq t$, then π_t and π_X are independent conditioned on $X = k$. It follows that the events $\mathcal{L}^C_j = \pi_X(j)$ is odd; $\mathcal{E}_t = \pi_t(j)$ is even; and $X = t$ are independent. Thus, we have the

following.

$$\Pr(\mathcal{L}_j^{\mathrm{C}} \ \wedge \ X = k \ \wedge \ \mathcal{E}_t \ \wedge \ \mathcal{E}_{t+1}^b)$$

$$= \Pr(\mathcal{E}_{t+1}^b | \mathcal{L}_j^{\mathrm{C}} \ \wedge \ X = k \ \wedge \ \mathcal{E}_t) \Pr(\mathcal{L}_j^{\mathrm{C}} \ \wedge \ X = k \ \wedge \ \mathcal{E}_t)$$

$$= \Pr(\mathcal{E}_{t+1}^b | \mathcal{L}_j^{\mathrm{C}} \ \wedge \ X = k \ \wedge \ \mathcal{E}_t) \Pr(\mathcal{L}_j^{\mathrm{C}}) \Pr(\mathcal{E}_t) \Pr(X = k)$$

$$= \Pr(\mathcal{E}_{t+1}^b | \mathcal{L}_j^{\mathrm{C}} \ \wedge \ X = k \ \wedge \ \mathcal{E}_t) \frac{\lceil n/2 \rceil \lfloor n/2 \rfloor}{n^2} (1 - \beta)^{k-1} \beta$$

$$\geq \Pr(\mathcal{E}_{t+1}^b | \mathcal{L}_j^{\mathrm{C}} \ \wedge \ X = k \ \wedge \ \mathcal{E}_t) \frac{\lfloor n/2 \rfloor}{2n} (1 - \beta)^{k-1} \beta$$

Now, if $k > t + 1$ then π_X and π_{t+1} are independent conditioned on X being either t or k, and hence, the events $\mathcal{E}_{t+1}^b$ and $\mathcal{L}_j \ \wedge \ X = t \ \wedge \ Y = k - t$ are independent, as are the events $\mathcal{E}_{t+1}^b$ and $\mathcal{L}_j^{\mathrm{C}} \ \wedge \ X = k \ \wedge \ \mathcal{E}_t$. It follows that $\Pr(\mathcal{E}_{t+1}^b | \mathcal{L}_j \ \wedge \ X = t \ \wedge \ Y = k - t)$ and $\Pr(\mathcal{E}_{t+1}^b | \mathcal{L}_j^{\mathrm{C}} \ \wedge \ X = k \ \wedge \ \mathcal{E}_t)$ both equal $\Pr(\mathcal{E}_{t+1}^b)$. On the other hand if $k = t + 1$ then $b = 1$ and $\mathcal{E}_{t+1}^b$ is implied in both cases Thus, $\Pr(\mathcal{E}_{t+1}^b | \mathcal{L}_j \ \wedge \ X = t \ \wedge \ Y = k - t)$ and $\Pr(\mathcal{E}_{t+1}^b | \mathcal{L}_j^{\mathrm{C}} \ \wedge \ X = k \ \wedge \ \mathcal{E}_t)$ are both 1. Either way, they are equal, and since their common value occurs in the numerator as well as in both terms in the denominator, it simply cancels out.

Putting everything together we see that

$$P_{j,t} \leq \frac{\frac{\lfloor n/2 \rfloor}{n} (1 - \beta)^{k-2} \beta^2}{\frac{\lfloor n/2 \rfloor}{2n} (1 - \beta)^{k-1} \beta + \frac{\lfloor n/2 \rfloor}{n} (1 - \beta)^{k-2} \beta^2}$$

$$= \frac{\beta}{\frac{1}{2}(1 - \beta) + \beta}$$

$$\leq 2\beta \qquad \qquad \qquad \square$$

We remark that although in the above proof we have bounded player j's estimate during the down-stage, a nearly identical proof shows the same bound for the up-stage (when $\pi_{t+1}(j)$ is unknown).

We are now ready to prove the main theorem.

Proof of Theorem 1. We will begin by showing that the protocol is a Nash equlibrium in which all the players learn the secret.

Suppose all the players follow the protocol. Then every round, during the up-stage players send their shares up toward the root where they are decoded, and during the down-stage the reconstructed secret and mask are sent back toward the leaves. If the odd players in the round do not drop out at the end of the round then play continues into the next round. In the definitive round, the real secret is reconstructed at the root and all the even labelled players, who are long players learn it first. Once it gets to the short players with odd labels, they drop out of the game since they have no tags to send any more messages. This signals the end of the game to the long players who then realize that the current reconstructed secret is the true one. Thus if all the players follow the protocol, everyone learns the secret.

Now suppose all players other than j are following the protocol. We want to show that player j prefers following the protocol over deviating.

At the beginning of the game, each player has set their current guess for the secret to 0. If no cheating occurred before the current round t then during round $t-1$ all the players set their current guess to s_{t-1}. Thus, at the beginning of round t, all players have the same guess for the secret. (Round $t-1$ has been eliminated as the definitive one, but s_{t-1} still has probability $1/|\mathcal{S}|$ of being the true secret.) Moreover, since by Lemma 3 partial information about the shares reveals no information about the symbol they encode, throughout the up-stage player j has no better guess than s_{t-1} for the secret. Since $\mathcal{U} < |\mathcal{S}|$, it is strictly better for j not to leave the game in the up-stage. If j sends incorrect messages in the up-stage, then even if he is not caught(which results in not learning the secret), this deviation will cause an incorrect value to be decoded instead of s_t. This results in j not learning the secret if t happened to be the definitive round. Thus, j has no incentive to deviate in the up-stage.

Now what about the down-stage? If j is on his last round of input, then he is a short player, and knows that the current round is definitive. At the same time, this means that he is an odd-labelled player and by the construction of the communication tree he cannot prevent anyone from learning the secret. Moreover, even if he successfully fakes a tag in order to convince the unique long player below him that the game has not ended, that player will detect in the next round that all other players have left the game and will therefore still output the correct secret. Thus this deviation does not change the outcome of the game, namely that all players learn the secret. It follows that j does not gain anything by this deviation. (Although he also does not lose anything by it.) Effectively, if j is a short player on his last round of input, it is too late for him to improve his payoff by deviating.

If j is not on his last round of input and is at an odd labelled node then, as remarked in the proof of Lemma 6, he *knows* the current round is not definitive, so cheating would be equivalent to randomly guessing the secret, which is correct with probability only $1/|S|$, This is worse than following the protocol by equation (1).

Now suppose that the current round, t, is not j's last round of input and j is at an even labelled node. Note that any spurious messages sent by Player j to players that are not expecting them, will be ignored. Also, any action involving not sending a message that is expected will be detected immediately, only by the involved players at first, but the knowledge will quickly propagates to all the players, before the end of the up-stage of the next round. Since detection of deviation causes other players to quit immediately, effectively such actions amount to player j leaving the game. Thus, the possible deviations we need to analyze for player j are:

- Leave the game, with or without sending fake messages first, and;

- Send fake messages to one or both children and hope to stay in the game by not being caught.

Let $P_{j,t}$ be j's estimate of the probability that the current round is definitive. Then by Lemma 6, $P_{j,t} \leq 2\beta$.

If player j leaves the game and outputs the value s_t then the probability that he has output the right value is $P_{j,t} + (1 - P_{j,t})/|\mathcal{S}|$, and since he has left the game, he has no later opportunity to improve that probability. By Lemma 5, in order to discourage this deviation it is sufficient if

$$P_{j,t} + \frac{(1 - P_{j,t})}{|\mathcal{S}|} \leq \frac{U_j - U_j^-}{U_j^+ - U_j^-} \tag{3}$$

Now consider the other kind of deviation. Suppose instead of sending s_t to his descendents, player j sends a fake value to one or both of them. Let α be the probability that he is not caught. By Proposition 4 we know that $\alpha = \frac{1}{q-1}$ if he sends a fake message to only one of his children, and, since the two verification functions are chosen independently by the dealer, $\alpha = \frac{1}{(q-1)^2}$ if he fakes both messages. If he is not caught, and if the current round is definitive, then player j has learned the true secret and has prevented some of his descendants from learning it. If the current round was not definitive and his deviation was not detected, the game continues and since the values s_i are all independent, it does not affect the next round.[2] This means that player j can either revert to following the protocol and guarantee learning the secret along with everyone else, or he may find further opportunities to cheat.

On the other hand, if the faked message is detected, which happens with probability $1 - \alpha$, then the game ends right away and player j outputs s_t. In this case, there is still a $P_{j,t}$ chance that the current round was definitive and an additional $(1 - P_{j,t})/|\mathcal{S}|$ chance that the value s_t was correct despite the current round not having been definitive. So the probability that the deviation succeeds is

$$\alpha + (1 - \alpha) \left(P_{j,t} + \frac{(1 - P_{j,t})}{|\mathcal{S}|} \right).$$

As this quantity is bigger when $\alpha = \frac{1}{q-1}$, faking only one message dominates faking both messages. Thus, again by Lemma 5, to discourage this deviation, it is sufficient if

$$\frac{1}{q-1} + \frac{q-2}{q-1} \left(P_{j,t} + \frac{(1 - P_{j,t})}{|\mathcal{S}|} \right) \leq \frac{U_j - U_j^-}{U_j^+ - U_j^-}. \tag{4}$$

Moreover, note that (4) implies (3). Thus for a Nash equilibrium, it is sufficient to show (4).

For the remainder of the proof, we are going to assume that n is sufficiently large, specifically that $n \geq \frac{2\mathcal{U}|\mathcal{S}|}{|\mathcal{S}|-\mathcal{U}}$. We will discuss the modifications required when $3 \leq n < \frac{2\mathcal{U}|\mathcal{S}|}{|\mathcal{S}|-\mathcal{U}}$ in Section 4.1.1.

We have so far not specified β. We do this now. Let

$$\beta = \frac{|\mathcal{S}| - \mathcal{U}}{4\mathcal{U}|\mathcal{S}|}$$

Then $1/\beta = O(1)$ so that the expected number of rounds in the game is constant.

---

[2]This is why player j does not try to fake the mask m_{t+1} - an successfully transmitted incorrect m_{t+1} will wreak havoc in the next round, since some players will be talking to the wrong players.

To show (4), recall that $q > n$, so that $q - 1 \geq n$. We have

$$\frac{1}{q-1} + \frac{q-2}{q-1}\left(P_{j,t} + \frac{(1-P_{j,t})}{|\mathcal{S}|}\right) < \frac{1}{n} + \left(P_{j,t} + \frac{(1-P_{j,t})}{|\mathcal{S}|}\right)$$

$$= \frac{1}{n} + \frac{|\mathcal{S}|-1)}{|\mathcal{S}|}P_{j,t} + \frac{1}{|\mathcal{S}|}$$

$$\leq \frac{|\mathcal{S}|-\mathcal{U}}{2\mathcal{U}|\mathcal{S}|} + 2\beta + \frac{1}{|\mathcal{S}|}$$

$$\leq \frac{|\mathcal{S}|-\mathcal{U}}{2\mathcal{U}|\mathcal{S}|} + 2\frac{|\mathcal{S}|-\mathcal{U}}{4\mathcal{U}|\mathcal{S}|} + \frac{1}{|\mathcal{S}|}$$

$$= \frac{|\mathcal{S}|-\mathcal{U}}{\mathcal{U}|\mathcal{S}|} + \frac{1}{|\mathcal{S}|}$$

$$= \frac{1}{\mathcal{U}}$$

as desired.

Finally, we analyze the resource costs of our protocol. The communication tree has $2n-1$ nodes. In each round, each player is mapped to one leaf and one internal node. Players only communicate with their neighbors in the tree. So on each round, during the up-stage each player sends up to three messages: one to his parent when he is a leaf; one to his parent when he is an non-root internal node; and if he is a child of the root and n is even, he has to send an additional message because there are two players at the root.

During the down stage each player sends two messages, one each to his two children. Thus each player sends at most five messages per round. Each message consists of four elements of $\mathbb{F}$ (shares of s_t, m_{t+1} and two tags) each of which is represented as $O(\log n)$, since $n < q \leq 2n$ Thus each player send $O(\log n)$ bits per round. The expected number of rounds is $1/\beta$ which is constant and so each player sends only $O(\log n)$ bits during the course of the game. Finally since the tree has depth $O(\log n)$ the number of rounds is constant (in expectation) and the communication is synchronous, it follows that the expected latency is $O(\log n)$. $\qquad\square$

4.1 Some Remarks

4.1.1 The Case of a Small Number of Players

When the number of players is a constant greater than 2, then scalability is not an issue, and one might hope to simply use the algorithm of Kol and Naor [7] by simulating non-simultaneous broadcast channels with secure private channels. Unfortunately their algorithm only provides an ϵ-Nash equilibrium, since the unique short player has a small chance of successfully pretending the game has not ended.

In our algorithm, $\lceil n/2 \rceil$ players are short players. In particular, even for $n = 3$, there are at least two short players, and none of the short players can increase their expected payoff by

cheating alone. Thus, we obtain a Nash equilibrium, provided that we can prove inequality (4). The above proof used the fact that n was at least $\frac{2\mathcal{U}|\mathcal{S}|}{|\mathcal{S}|-\mathcal{U}}$, so we need a separate argument.

However since when n is constant, scalability is immediate, we have more leeway to choose a larger field to work with.[3] So we can work in a prime field of size q where

$$\max\{|\mathcal{S}|, \frac{2\mathcal{U}|\mathcal{S}|}{|\mathcal{S}|-\mathcal{U}}\} < q \leq 2\max\{|\mathcal{S}|, \frac{2\mathcal{U}|\mathcal{S}|}{|\mathcal{S}|-\mathcal{U}}\}$$

and the proof of (4) goes through as before, giving us a Nash equilibrium.

4.1.2 Nash equilibria vs. Strict Nash equilibria

An n-tuple of strategies is called a strict Nash equilibrium if when all other players are following the prescribed strategy, a player unilaterally deviating achieves a strictly worse expected payoff than he would by following the equilibrium strategy.

Our algorithm fails to be a strict Nash equilibrium, for the following reasons:

- Any player may, at any time, send spurious messages that are not part of the protocol, to players that are not his neighbors in the tree. Such messages will be ignored by their recipients, who are following the protocol.

- At the end of the definitive round, a short player may try to fake a tag and send a message to the long player below him. This may go undetected with some small probability, but as noted in the proof, even in this case, it cannot fool that long player into outputting the wrong secret.

Our proof shows that our algorithm *does* have the property that any player deviating from the protocol in one of the above ways does not increase his payoff, and moreover *does not affect any other player's payoff either.* In other words, if a player deviating unilaterally from our protocol, does so in a manner that changes some other player's payoff, then he strictly reduces his own expected payoff. In this weaker sense, the equilibrium is strict.

4.1.3 A Note on Backwards Induction

The *backwards induction* problem arises when a multi-round protocol has a last round number that is known to all players. In particular, if it is globally known that the last round of the protocol is ℓ, then on the ℓ-th round, there is no longer any fear or reprisal to persuade a player to follow the protocol. But then if no player follows the protocol in the ℓ-th round, then in the $(\ell-1)$-th round, there is no reason for any player to follow the protocol. This same logic continues backwards to the very first round.

The backwards induction problem can occur with protocols that make cryptographic assumptions, since there will always be some round number, ℓ, in which enough time has passed so that even a computationally bounded player can break the cryptography. Even

---

[3]The upper bound of $O(n)$ on the field size came from the desire to keep $4\log q$, which is the size of an individual message, small.

though ℓ may be far off in the future, it is globally known that the protocol will end at round ℓ, and so by backwards induction, even in the first round, there is no incentive for a player to follow the protocol.

As in [7], we protect against backwards induction by having both long and short players. As the above analysis shows, if N is chosen sufficiently large, we can ensure that the probability of making a correct guess as to when the protocol ends is too small to enable profitable cheating for any player. Thus, even when a player gets to the second to the last element in all his lists, he can not be very sure that the protocol will end in the next round. All players are aware of these probabilities at the beginning of the protocol, and thus each player knows that no other player will be able to accurately guess when the protocol ends.

5   Algorithm for Case of Absent Players

In this section we discuss m-out-of-n secret sharing where $m < n$. Here we want a subset of m or more of the players to be able to reconstruct the secret even when the remaining players are absent. However, fewer than m players should *not* be able to reconstruct the secret on their own.

As discussed previously, it does not seem possible to design *scalable* algorithms for secret sharing in the case when either m is much smaller than n, or when the subset of m active players may be chosen in a completely arbitrary manner. We now address the situation where (1) $m < n$, but $m = \Theta(n)$; and (2) the subset of active players does not depend on the random bits of the dealer. More precisely, we will present an algorithm with parameters τ and λ such that when $m > (\tau + \lambda)n$ then with high probability the algorithm is an Nash equilibrium for the active players, and all the active players learn the secret. On the other hand, if $m < (\tau - \lambda)n$, then with high probability the active players cannot reconstruct the secret.

We will assume, as in previous work, that the set of active players is known to all the active players, before the start of the players' protocol. However, this set is not known to the dealer at the time of share creation. The set of active players may be arbitrary or randomly chosen, but it does not depend on the randomness used by the dealer for the algorithm.

Since the algorithm is a variant of the n-out-of-n scheme, for conciseness, we now describe only the places where the two algorithms differ. Let c be a large constant, which we will specify later. First, the dealer partitions the players into $Q = \frac{n}{c \log n}$ pairwise disjoint groups of $c \log n$ players each.[4] This is done using a random permutation of the players, where the first $c \log n$ players in the permutation are the first group, the next $c \log n$ are the second group and so on. The groups are labelled $1, 2, \ldots, Q$. The communication tree used for the algorithm is the labelled Q-out-of-Q tree, except that all of the nodes are *supernodes*, in that they correspond to groups of $c \log n$ players instead of single players. An edge of this

---

[4]It is convenient to assume that $c \log n$ is an integer and divides n, and that the quotient Q is odd. In fact it is always possible to choose c so that Q is odd, and for arbitrary n, there will be $Q = \left\lfloor \frac{n}{\lceil c \log n \rceil} \right\rfloor$ groups, each with either $\lceil c \log n \rceil$ or $\lceil c \log n \rceil + 1$ players.

communication tree will correspond to all-to-all communication between the active members of the groups at the endpoints of the edge.

As in Algorithm 1, the dealer samples X and Y independently from $G(\beta)$. (Here $\beta = (|\mathcal{S}| - \mathcal{U})/4\mathcal{U}|\mathcal{S}|$ is the same parameter as in the n-out-of-n algorithm.) For each round t, the dealer essentially implements the corresponding round of the Q-out-of-Q algorithm. For $t \neq X$, he picks random secret s_t for the round. If $t = X$, s_X is the true secret. The dealer then picks a random permutation $\pi_t \in S_Q$ to determine which group is assigned to which node for that round; and picks a random mask m_{t+1} to encode positional data. A key difference is that now the positional data constitutes not only which node your group is at and which groups are you neighbors, but also which $c \log n$ players are in all the relevant groups. For this reason we still need to work in a field with size bigger than n; a field of size merely bigger than Q is not enough.

The dealer then creates Q-out-of-Q iterated shares for s_t and m_{t+1}. Finally, the values of these Q-out-of-Q iterated shares at the leaves are further encoded into $\tau c \log n$-out-of-$c \log n$ shares via Shaimr's scheme [10]. These are the shares that will be distributed to the players for round t. This last step is to avoid all the players in a group having the same share, because if that were the case, then the secret could be decoded by $Q = o(m) = o(n)$ players, one from each group. As before, the dealer, creates authentication data for all the messages to be sent in the algorithm[5]; identifies the short players as the members of the groups at the odd labeled nodes in round X; truncates shares appropriately; and sends the shares to the players.

The players' protocol is very similar to Algorithm 4. The only differences are that each active player must send his messages to all active players at neighboring nodes, and at the beginning of each round the active players at each leaf node must first reconstruct the value at the leaf using their $\tau c \log n$-out-of-$c \log n$ Shamir shares. This means that unless there are at least $\tau c \log n$ active players in each group, the algorithm will fail. Herein lies the reason for the δ failure probability of the algorithm.

Formal descriptions of the protocols for the dealer and players are presented as Algorithms 5 and 6.

5.1 Analysis

In order to show correctness of the algorithm, we need to make two arguments. The first argument is that the active players are well distributed among the groups, so that at any stage of the players' protocol sufficiently many shares are available to do the desired reconstruction. The second argument is that, the protocol is an Nash equilibrium for the active players. The proof of this part is essentially identical to the proof that the n-out-of-n protocol was a Nash equilibrium for all the players, and we omit it here. In the remainder of this section, we sketch why reconstruction is possible with high probability, despite absent players.

We note that in any round, for the value at an internal node to be reconstructed it is

---

[5]Technically this is not necessary. Since there is all-to-all communication between adjacent groups, messages sent by individuals can be checked against those sent by other members of the group.

Algorithm 5 Dealer's Protocol

$\mathbb{F}$ field of size q (to represent messages in the algorithm) n players with distinct identifiers in $[n] \hookrightarrow \mathbb{F}$, $\beta \in (0,1)$: geometric distribution parameter. τ, λ, k threshold parameters.

1. Choose X, Y, independently from a geometric distribution with parameter β. Round X is the definitive one. Short players will receive full input for $X - 1$ rounds and partial input for round X. Long players will receive full input for $X + Y - 1$ rounds and partial input for round $X + Y$.

2. Let $c = \frac{2(k+1)}{\lambda^2}$ and $Q = \frac{n}{c \log n}$ Choose a random permutation $\pi \in S_n$, and use it to divide players into Q groups (numbered $1, 2, \ldots, Q$) of size $c \log n$. Use the complete binary tree with Q leaves described in Section 3.1.

3. For each round t between 1 and $L = X + Y$:

   - If $t < L$, choose a random permutation $\pi_t \in S_Q$. If $t = L$ choose a permutation π_L which is random subject to the constraint that all the long player groups (determined by π_X) are assigned to odd labels under π_L. For round t player g will be assigned to all nodes marked $\pi_t(g)$ in the tree. If $t = 1$, $m_1 = 0$ // (Otherwise m_t was set in the previous round)

   - For every group g, use π_t and m_t to create masked positional data for g for round t. Positional data consists of the group's position and members, neighboring groups and their members.

   - Choose a random mask $m_{t+1} \in \mathbb{F}$ (for the **next** round.)

   - Create shares of m_{t+1} by calling *RecursiveShares* (root, m_{t+1}).

   - If $t = X$ $s_t \leftarrow$ true secret.
     Otherwise, $s_t \leftarrow$ random element of $\mathcal{S}$

   - Create shares of s_t by calling *RecursiveShares* (root, s_t).

   - Create $\tau c \log n$-out-of-$c \log n$ Shamir shares of each of the leaf values of the shares created by *RecursiveShares* (one for each player in the group corresponding to the leaf.)

   - Create authentication data.

   - For each g, for each player $j \in g$, j's (full) input I_t^j for round t consists of positional data, Shamir shares of recursive shares of m_{t+1} and s_t corresponding to node $\pi_t(g)$, and authentication data. Partial input $\tilde{I}_t^j$ consists of all of the above except the authentication tags for sending messages to your children (in the down-stage).

4. Identify the short players as those players j who are in groups at odd numbered nodes in the definitive iteration, *i.e.*, $\pi_X(j)$ is odd.

5. For each short player j, send j the list $I_1^j, \ldots I_{X-1}^j, \tilde{I}_X^j$.

6. For each long player j, send j the list $I_1^j, \ldots I_{L-1}^j, \tilde{I}_L^j$.

Algorithm 6 Protocol for Player j

S=0; M=0

If at any time you receive spurious messages (messages not expected uder the protocol), ignore them.

On round t:

    Up-Stage:

        1. $m_t = M$

        2. Use m_t to unmask and discover your position in the tree and the identities of your group members and neighbors for round t.

        3. (as player at a leaf) Send your Shamir share to all active members of your group. Receive Shamir shares from all active members of your group. If insufficient shares received, output S and quit. Otherwise, reconstruct the leaf values of the recursive shares of s_t and m_{t+1} and send them along with their tags to all active members of the group at your parent node in the tree.

        4. (as player at an internal node)

            (a) Receive copies of (intermediate) shares of s_t and m_{t+1} and tags from active members of the groups at your left and right children nodes. Check that correct messages have been sent. If a fault is detected (missing or incorrect message) output S and quit.

            (b) For each of s_t and m_{t+1}: interpolate a degree 1 polynomial f from $(-1,\text{ left-share})$ and $(1,\text{ right-share})$. Evaluate $f(0)$. This is your share.

            (c) If you are not at the root, send the above reconstructed shares of s_t and m_{t+1} to all active members of the group at your parent node. If you *are* at the root, these shares are the actual values of s_t and m_{t+1}.

    Down-Stage:

        1. If you are at the root, set $S = s_t$ and $M = m_{t+1}$ and send these values along with authentification tags to all active members of the groups at your left and right children nodes.

        2. Else

            (a) (as a non-root internal node) Receive copies of s_t and m_{t+1} and tags from all active members of the group at your parent node and check them. If fault detected, output S and quit.

            (b) Set $S = s_t$ and $M = m_{t+1}$.

            (c) Send s_t and m_{t+1} to all active members of the groups at your children nodes. If you are a short player and have no authentication tags, output s_t and quit.

    $t \leftarrow t + 1$

---

necessary that the values of both of its children be received. Therefore, it is necessary that there is at least one active player at each internal node. Since no other reconstruction is to be done at internal nodes, this is also sufficient. However, note that the group of $c \log n$ players assigned to an internal node is the same group as those assigned to some leaf node. Hence if there are no active players at some internal node, then there is a leaf node at which there are also no active players. Since at the leaf nodes we will have a more stringent requirement for how many players need to be active, it is sufficient to consider the failure of the algorithm at the leaf nodes.

Now, the value at a leaf node is distributed as $\tau c \log n$-out-of-$c \log n$ Shamir shares to the $c \log n$ players associated with that leaf node. Thus, this value can be reconstructed if and only if there are at least $\tau c \log n$ active players at the leaf node. Moreover, in order for the protocol to succeed, the values at all the leaf nodes must be reconstructible.

Recall that the players are assigned to leaf nodes by their group number so all leaf node values are reconstructible in a particular round if and only if all the groups have at least $\tau c \log n$ active players in that round. Moreover, since the same groups are used for all rounds, though not the same assignment of them to leaves, and each player is either active or absent for the entire game, the following lemma holds.

Lemma 7. All the leaves are reconstructible throughout the algorithm if and only if all the groups have at least $\tau c \log n$ active players.

Now recall that the players are assigned to groups by the following random process. The dealer chooses a random permutation of the n players, and partition the players into $Q = \frac{n}{c \log n}$ pairwise disjoint groups by choosing successive contiguous blocks of length $c \log n$ in the permutation. Moreover, the choice of which players are active is made *independently* of the above process. It turns out that in this case, the number of active players in a fixed block (*i.e.* group) is tightly concentrated around its mean. A more precise statement is in the following lemma, whose proof is a simple application of the Azuma-Hoeffding inequality. We omit the details here.

Lemma 8. Let $b_1, b_2, \ldots, b_n$ be n bits such that exactly m of them are 1 and $n - m$ of them are 0. Let $\sigma \in S_n$ be a random permutation of n symbols and $b_{\sigma(1)}, b_{\sigma(2)}, \ldots, b_{\sigma(n)}$ be the induced permutation on bits. Fix any contiguous block $b_{\sigma(i+1)}, b_{\sigma(i+2)}, \ldots, b_{\sigma(i+c \log n)}$ of length $c \log n$, and let random variable Z denote the number of bits in the block which are 1. Then Z has expectation $\frac{mc \log n}{n}$ and satisfies the following concentration inequalities:

$$\Pr\left(Z - \frac{mc \log n}{n} > \lambda c \log n\right) \leq e^{-(\lambda^2 c \log n)/2}$$

and

$$\Pr\left(Z - \frac{mc \log n}{n} < -\lambda c \log n\right) \leq e^{-(\lambda^2 c \log n)/2}$$

We apply the above concentration inequality to show the following.

Lemma 9. Let $k \geq 1$, and let m denote the number of active players. If all the active players follow the algorithm, then with probability at least $1 - \frac{1}{n^k}$,

- If $m \geq (\tau + \lambda)n$, all active players learn the secret.

- If $m \leq (\tau - \lambda)n$, the secret cannot be reconstructed.

Proof. As already remarked in Lemma 7 all the leaf values of the iterated shares of the secret can be reconstructed every round if and only if each group has at least $\tau c \log n$ players. Also, by Lemma 3 the iterated shares can be decoded into the secret if and only if all the shares at the leaves are available. Thus, provided the active players follow the protocol, the secret is recovered if and only if each group has at least $\tau c \log n$ players, so that is what we need to prove.

Imagine the permutation of players is a bit string such that every bit corresponds to one player. If a bit is 1 it means the corresponding player is active and if a bit is 0 the corresponding player is inactive. There are two cases.

Case (1) $m \geq (\tau + \lambda)n$. Consider a particular group g and let Z_g denote the number of active players at g. By Lemma 8,

$$
\begin{aligned}
\Pr(Z_g < \tau c \log n) &= \Pr\left(Z_g - \frac{mc \log n}{n} < \left(\tau - \frac{m}{n}\right)c \log n\right) \\
&\leq \Pr\left(Z_g - \frac{mc \log n}{n} < -\lambda c \log n\right) \\
&\leq e^{-(\lambda^2 c \log n)/2}
\end{aligned}
$$

Taking a union bound over all the $\frac{n}{c \log n}$ groups we see that the probability that the algorithm fails to recover the secret, which happens only when *some* group does not have enough active players, is at most $ne^{-(\lambda^2 c \log n)/2} = n^{1-\lambda^2 c/2}$. Setting $c = \frac{2(k+1)}{\lambda^2}$ we see that the probability that the algorithm fails is at most $1/n^k$.

Case (2) $m \leq (\tau - \lambda)n$. Once again consider a particular group g and let Z_g denote the number of active players at g. By Lemma 8,

$$
\begin{aligned}
\Pr(Z_g > \tau c \log n) &= \Pr\left(Z_g - \frac{mc \log n}{n} > \left(\tau - \frac{m}{n}\right)c \log n\right) \\
&\leq \Pr\left(Z_g - \frac{mc \log n}{n} > \lambda c \log n\right) \\
&\leq e^{-(\lambda^2 c \log n)/2}
\end{aligned}
$$

Therefore with probability at least $1 - e^{-(\lambda^2 c \log n)/2} = 1 - 1/n^{k+1}$, g does not have enough active players. Since the failure of a single group to have enough active players is sufficient to break the protocol completely (by Lemma 3) it follows that with probability at least $1 - 1/n^{k+1} \geq 1 - 1/n^k$ the algorithm fails to recover the secret. $\square$

Why do players follow the protocol? As in the n-out-of-n case, we can argue that a player looking at his remaining input has a very low estimate of the current round being definitive, unless he is actually on his last round. In his last round of input, there is nothing he can do to prevent others from learning the secret that would not also prevent himself from learning it. We omit the details, which are essentially the same as in the proofs of Lemma 6 and Theorem 1. We conclude with

Proof of Theorem 2. Lemma 9 shows that with high probability, when the fraction of active players is higher than $\tau + \lambda$, all active players following the algorithm leads to their all learning the secret, while if the fraction is less than $\tau - \lambda$ then nobody learns it. We have also remarked that no player can do better by deviating from the protocol, so that it is a Nash equilibrum. The communication tree has depth $\log Q = \log n - \log \log n - \log c = O(\log n)$. On each round, each player sends messages to $\Theta(\log n)$ others, and the messages themselves are $\log q = O(\log n)$ bits. The algorithm runs for $1/\beta = O(1)$ steps in expectation. It follows that the algorithm has latency $O(\log n)$ and each player sends $O(\log^2 n)$ bits in expectation. $\qquad\square$

6   Conclusion

We have presented *scalable* mechanisms for rational secret sharing problems. Our algorithms are scalable in the sense that the number of bits sent by each player is $O(\log n)$ and the latency is at most logarithmic in the number of players. For n-out-of-n rational secret sharing, we give a scalable algorithm that is a Nash equilibrium to solve this problem. For m-out-of-n rational secret sharing where (1) $m = \Theta(n)$; and (2) the set of active players is chosen independently of the random bits of the dealer, we give a scalable algorithm with threshold parameter τ that is a Nash equilibrium and ensures that for any fixed, positive λ that if (1) at least a $m/n > \tau + \lambda$ fraction of the players are active, all players will learn the secret; and (2) if fewer than a $\tau - \lambda$ fraction of the players are active, then the secret can not be recovered.

Several open problems remain. First, while our algorithms lead to a $\Theta(n)$ multiplicative reduction in communication costs for rational secure multiparty computation (SMPC), the overall bandwidth for this problem is still very high. We ask: Can we design scalable algorithms for rational SMPC? This is related to our second open problem which is: Can we design scalable algorithms for simulating a class of well-motivated mediators? In some sense, this problem may be harder than the SMPC problem, since some types of mediators offer different advice to different players. In other ways, the problem is easier: a simple global coin toss is an effective mediator for many games. A final important problem is: Can we design coalition-resistant scalable algorithms for rational secret sharing?

Acknowledgments

References

[1] I. Abraham, D. Dolev, R. Gonen, and J. Halpern. Distributed computing meets game theory: robust mechanisms for rational secret sharing and multiparty computation. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 53–62. ACM, 2006.

[2] V. Dani, M. Movahedi, Y Rodriguez, and J. Saia. Scalable Rational Secret Sharing. In *Proceedings of the thirtieth annual ACM symposium on Principles of distributed computing*. ACM, 2011.

[3] R. Geambasu, T. Kohno, A.A. Levy, and H.M. Levy. Vanish: Increasing data privacy with self-destructing data. In *Proceedings of the 18th conference on USENIX security symposium*, pages 299–316. USENIX Association, 2009.

[4] S. Gordon and J. Katz. Rational secret sharing, revisited. *Security and Cryptography for Networks*, pages 229–241, 2006.

[5] J. Halpern and V. Teague. Rational secret sharing and multiparty computation: extended abstract. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, page 632. ACM, 2004.

[6] V. King and J. Saia. Breaking the O (n 2) bit barrier: scalable byzantine agreement with an adaptive adversary. In *Proceeding of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 420–429. ACM, 2010.

[7] G. Kol and M. Naor. Games for exchanging information. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 423–432. ACM, 2008.

[8] A. Lysyanskaya and N. Triandopoulos. Rationality and adversarial behavior in multiparty computation. *Advances in Cryptology-CRYPTO 2006*, pages 180–197, 2006.

[9] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 73–85. ACM, 1989.

[10] Adi Shamir. How to share a secret. *Communications of the ACM (CACM)*, 22(11):612–613, 1979.

[11] M.N. Wegman and J.L. Carter. New hash functions and their use in authentication and set equality. *Journal of computer and system sciences*, 22(3):265–279, 1981.