Randomness versus Fault-Tolerance*

Ran Canetti† Eyal Kushilevitz‡ Rafail Ostrovsky§ Adi Rosén¶

April 30, 1998

Abstract

We investigate the relations between two major requirements of multiparty protocols: *fault tolerance* (or *resilience*) and *randomness*. Fault-tolerance is measured in terms of the maximum number of colluding faulty parties, t, that a protocol can withstand and still maintain the privacy of the inputs and the correctness of the outputs (of the honest parties). Randomness is measured in terms of the total number of random bits needed by the parties in order to execute the protocol.

Previously, the upper bound on the amount of randomness required by general constructions for securely computing any non-trivial function f was polynomial both in n, the total number of parties, and the circuit-size $C(f)$. This was the state of knowledge even for the special case $t = 1$ (i.e., when there is at most one faulty party). In this paper, we show that for any linear-size circuit, and for any number $t < n/3$ of faulty parties, $O(poly(t) \cdot \log n)$ randomness is sufficient. More generally, we show that for any function f with circuit-size $C(f)$, we need only $O\left(poly(t) \cdot \log n + poly(t) \cdot \frac{C(f)}{n}\right)$ randomness in order to withstand any coalition of size at most t. Furthermore, in our protocol only $t + 1$ parties flip coins and the rest of the parties are deterministic. Our results generalize to the case of *adaptive* adversaries as well.

Keywords: Secure multiparty protocols, Randomness, Limited independence, Composition of protocols.

*A preliminary version of this paper appears in the proceedings of *16th PODC,* 1997, 35-45.

†IBM T.J. Watson Research Center. e-mail:canetti@watson.ibm.com

‡Dept. of Computer Science, Technion, Haifa, Israel. e-mail: eyalk@cs.technion.ac.il; Supported by MANLAM Fund. Part of this work was done while visiting Bellcore. URL: http://www.cs.technion.ac.il/~eyalk

§Bell Communications Research, MCC-1C365B, Morristown, NJ 07960-6438, USA. e-mail:rafail@bellcore.com

¶Dept. of Computer Science, University of Toronto, Toronto, Canada. e-mail: adiro@cs.toronto.edu. Part of this work was done while visiting Bellcore.

1 Introduction

The goal of this work is to explore the interplay, in the context of multiparty computations, between two fundamental concerns: *security* (i.e., fault-tolerance combined with privacy) and *randomness*. Over the past decade, both striving for stronger security and saving random bits received considerable amount of attention and yielded many interesting results.

Secure protocols. Secure multiparty protocols (first studied in [53, 28]) are protocols that guarantee the privacy of the inputs and, at the same time, the correctness of the outputs of honest participants, even if some of the parties are maliciously faulty ("Byzantine"). Secure multiparty computations has been extensively studied, in a variety of adversarial models. The following basic settings were considered. The adversary controlling the corrupted (i.e., faulty) parties can be either computationally unbounded (in which case the communication channels are assumed to be private) [6, 16], or it can be limited to efficient (probabilistic polynomial time) computations [53, 28]. In addition, the adversary can be either *passive* (in which case the corrupted parties are honest-but-curious; they follow their protocol and only collude to gather extra information), or *active* (in which case the corrupted parties may arbitrarily and maliciously deviate from their protocol). A protocol resilient against passive adversaries is sometimes called *private*, rather than *secure*. In all settings, a salient parameter is the *resilience* t, i.e. the maximum number of colluding faulty parties tolerable by the protocol. An additional parameter regarding the power of the adversary is *adaptivity:* A *static* adversary controls a fixed set of faulty parties, whereas an *adaptive* adversary may choose which parties to corrupt as the computation proceeds, based on the information gathered so far. In this work we consider *adaptively secure* protocols — that is, protocols secure against adaptive adversaries.

We mention some known results: In [53, 28] it was shown that, if trapdoor permutations exist, every poly-time computable function f can be computed securely tolerating a computationally bounded, active adversary that controls up to $t < n/2$ parties. Moreover, in the case of passive adversaries, any number $t \leq n$ of colluding parties is tolerable. In [6, 16] protocols for securely computing any function in the presence of computationally unbounded adversaries are presented. In the case of passive adversaries these protocols withstand up to $t < n/2$ corrupted parties. In the case of active adversaries these protocols withstand up to $t < n/3$ corrupted parties. In both cases this is the maximum attainable resilience. Considerable amount of work has been done in this area (e.g., [2, 3, 7, 8, 13, 20, 22, 26, 39, 40, 41, 42, 44]); in the sequel we concentrate on works concerning the relation between multiparty security and randomness.

Randomness. Randomness plays an important role in computer science. In particular, in the context of distributed computing there are important examples of problems where there is a provable gap between the power of randomized algorithms and their deterministic counterparts. For instance, achieving Byzantine agreement with linear number of faults requires linear number of rounds deterministically [24] and constant number of rounds if randomization is allowed [23]; reaching a consensus in an asynchronous distributed system with faults is impossible with deterministic protocols [25], but is possible with the use of randomized protocols (see [17]). Various techniques to minimize the amount of randomness needed were extensively studied in computer science (e.g., [36, 52, 10, 47, 54, 18, 32, 4, 46, 1, 50, 35, 37, 40, 33, 34]) and tradeoffs between randomness and other resources were found (e.g., [14, 48, 38, 15, 21, 9, 8, 44, 7, 42, 40]).

Security vs. Randomness. It is not hard to show that *some* randomness is essential to maintain security (if all parties are deterministic then the adversary can infer information on the parties' inputs from their messages). We are interested in the *amount of randomness* required for carrying out a t-resilient computation against computationally unbounded adversaries.[1]

All previous (generic) secure protocols require $\Theta(poly(n) \cdot m)$ random bits, where n is the number of parties, and m is the size (i.e., number of gates) of the circuit representing the function to be computed. This applies both to passive and active adversaries. Previous research concentrating on reducing the amount of randomness used in secure computations was limited to the case of *passive* and *static* adversaries. Furthermore, results were obtained either for a specific function (namely XOR) or for the special case $t = 1$:

1. For the XOR function, $\Omega(t)$ random bits are necessary for t-private computation, while $O(t^2 \log(n/t))$ random bits are sufficient [40]. Additionally, for any function f with *sensitivity* n, if $t \geq n - c$ for some constant c, then $\Omega(n^2)$ random bits are required [7].

2. For the special case of 1-privacy, any linear-size circuit can be computed 1-privately with constant number of random bits [42]. More generally, every circuit of m boolean gates can be computed 1-privately with $O(m/n)$ random bits [42].

Our Results. We generalize both of the above results. That is, we show that for both passive and active adaptive adversaries, and for *any* value of t for which secure computation is possible, any circuit of m boolean gates can be securely evaluated using only $O(poly(t) \cdot \log n \cdot \frac{m}{n})$ random bits overall. While these results do not substantially improve on [6, 16] for $t = \Theta(n)$, they constitute big improvement for smaller values of t. In particular, for $t = polylog(n)$, circuits with quasi-linear (i.e., $m = O(n \cdot polylog(n))$) number of gates can be securely evaluated using only $polylog(n)$ random bits. For $t = 1$, we are only $O(\log n)$ away from the specialized (to passive adversaries only) result of [42].

An Alternative Perspective. We suggest the following alternative perspective on our results. Any distributed computing task (i.e., a task whose input is partitioned among several parties) can, in the absence of faults, be solved in a centralized manner: all parties send their input to a single party, who performs the task locally and announces the results. In many cases this may be the preferred solution, but this solution requires that the correctness (and privacy) be trusted to a single party. A natural extension of the centralized solution to the case when up to t faults are possible is to have all parties share their inputs among a predefined small set S of $c \cdot t$ parties ($c > 1$), and have the parties in S compute the function and announce the results. This "small decentralization" approach seems especially viable when $t = o(n)$, since the set S need not be much bigger than t. Our work shows that, with respect to the amount of randomness used, this "small decentralization" solution is considerably inferior to a fully distributed computation: in contrast to our results (we need only $O(\frac{m}{n} \cdot \log n \cdot poly(t))$ randomness), the above "small decentralization" solution according to presently known methods requires $O(m \cdot poly(t))$ random bits.

Our Constructions. Our results build on many previous ideas in the area of privacy as well as on limited independence distributions. In particular, we use the general framework of [6], and combine it with ideas from [42] together with techniques for limited independence, in order to save in randomness. That is, the parties evaluate the given circuit gate by gate; each gate is computed in a manner similar to the construction of [6]. (In particular, we use the [6] modules for secret sharing and evaluating individual gates as building blocks.) However, as in [42], not all parties participate in evaluating each gate. Instead, the parties are partitioned into *teams* of small size, and each gate is evaluated by a single team. We generalize

[1] When the adversary is limited to probabilistic polynomial time and intractability assumptions are used, as in [53, 28], then by the results of [10, 30, 31] we may as well assume the existence of a pseudorandom generator. In this case parties can expand "small" seeds of truly random bits into "long" sequences of pseudo-random bits and use them. Therefore, in this case the quantification of the "amount of randomness needed" is not meaningful. (In particular, the amount of randomness needed inherently depends on a security parameter.)

the technique of [42] in a way which allows us to use limited independence, and then show how this can be done in a secure and robust manner, building on previous work on both secure protocol design and de-randomization techniques.

Interestingly, we show that not only we can use a small amount of randomness but also only $t+1$ parties need to be randomized, and the rest of the parties can be deterministic. This is nearly optimal against coalitions of size t, since it was shown in [40] that t-private computations of simple functions require at least t parties to use randomness, and that in some cases, such as the XOR function, t is sufficient.

The Protocols Composition Technique. To show the security of our protocols, we use general definitions of secure multiparty protocols. In particular, we use the formalization of [11], which allows modular composition of secure protocols. (This formalization is based on the [3] approach.) That is, in order to avoid re-proving the security of the [6] construction from scratch, we separately prove the security of the overall design of our protocol, assuming that the [6] modules for secret-sharing and for evaluating individual gates are secure. We then conclude that the composition of our "overall design" with the [6] modules is secure using the [11] composition theorem. We remark that a formal proof of security for [6] was never published. (It can be inferred, say, from the security proof of [5] as it appears in [12].) The modular proof technique used here can be applied also to proving the security of the [6] protocol itself.

Organization. In Section 2 we provide some necessary definitions, including those of privacy and randomness. In Section 3, we review the solution of [6] for the case of passive adversaries. In Section 4 we provide our solution for the same case. In Section 5, we review the solution of [6] for the case of active (i.e., Byzantine) adversaries and in Section 6 we extend our solutions from the case of passive adversaries to the case of active adversaries. In the Appendix we describe an extension of the results of [50, 40] for sample spaces with limited independence; we use this extension in our constructions.

2 Preliminaries

We start by specifying the requirements from a protocol for securely computing a function f whose inputs are partitioned among several parties. Several definitions of multiparty secure computation have been proposed in the past (e.g., [45, 29, 3, 13, 11]). In this work we use the definition of [11] which, for self containment, we sketch below. We concentrate on the 'secure channels' setting of [6, 16], where the adversary is computationally unbounded but has no access to the communication between non-faulty parties. Moreover, we concentrate on the case of strong adversaries; i.e., those which are both *active* and *adaptive*. The definition for weaker adversaries (passive, non-adaptive) can be inferred in a straightforward way. Figuratively, secure protocols "emulate" an ideal setting where all parties privately hand their inputs to a centralized trusted party who computes the results, hands them back to the parties, and vanishes.

Real-Life Model. The 'real-life' computation is modeled as follows. The parties interact via their private communication links; an adversary *corrupts* parties during the course of the computation based on the information seen so far. Once a party P_i is corrupted it hands all the information it has seen so far to the adversary. If the adversary is active then from this point on party P_i follows the adversary's instructions. If the adversary is passive then the corrupted party P_i still follows its original protocol. We call this adversary a *real-life adversary.* At the end of the computation each uncorrupted party locally outputs whatever is specified in its protocol. The corrupted parties output s special 'I am corrupted' symbol, and the adversary outputs some arbitrary function of the information gathered during the computation.

We use the following notation. Let $\text{ADV}_{\pi,\mathcal{A}}(\vec{x},\vec{z},\vec{r})$ denote the output of real-life adversary $\mathcal{A}$ when interacting with parties running protocol π on input $\vec{x}=x_1,\ldots,x_n$, random input $\vec{r}=r_0,\ldots,r_n$ and auxiliary information $\vec{z} = z_0, \ldots, z_n$ (where z_0, r_0 are for $\mathcal{A}$; and x_i, z_i, r_i are for party P_i).[2] Let $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, \vec{z}, \vec{r})_i$ denote the output of party P_i after running protocol π on input $\vec{x}$, random input $\vec{r}$, auxiliary information $\vec{z}$, and with a real life adversary $\mathcal{A}$. If P_i is uncorrupted then $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, \vec{z}, \vec{r})_i$ is just the output specified by the protocol; if P_i is corrupted then $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, \vec{z}, \vec{r})_i = \perp$. Let $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, \vec{z}, \vec{r})$ denote the joint output of the adversary and all parties. That is,

$$\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, \vec{z}, \vec{r}) \quad = \quad \text{ADV}_{\pi,\mathcal{A}}(\vec{x}, \vec{z}, \vec{r}), \text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, \vec{z}, \vec{r})_1, \ldots, \text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, \vec{z}, \vec{r})_n.$$

Finally, let $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, \vec{z})$ denote the random variable describing $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, \vec{z}, \vec{r})$ where $\vec{r}$ is uniformly chosen.

Ideal Model. The 'ideal model' for multiparty function evaluation is formulated next. This ideal model captures "the highest level of security we can hope to get from such evaluation". A computation in this ideal model goes as follows. First, an *ideal-model adversary* corrupts a set of parties, one by one, in an adaptive way[3]. Once a party P_i is corrupted, the adversary learns the party's input x_i and auxiliary input z_i; if the adversary is active then it may also *modify* the party's input. Next, all parties hand their (possibly modified) inputs to an incorruptible *trusted party.* The trusted party computes the outputs (i.e., it evaluates the given function f at the given inputs) and it hands each party its designated output. If the function is randomized then the trusted party makes the required random choices. Once the function values are announced, the adversary may continue corrupting parties at wish. Finally, the uncorrupted parties output whatever they receive from the trusted party; the corrupted parties output a special 'I am corrupted' symbol (same as in the real-life model) and the adversary outputs some arbitrary function of the information gathered during the computation. (This information consists of the corrupted parties' identities, inputs, auxiliary inputs, and values received from the trusted party, and the random and auxiliary input of the adversary.)

Let $\text{ADV}_{f,\mathcal{S}}(\vec{x}, \vec{z}, r, r_f)$ denote the output of ideal model adversary $\mathcal{S}$ on random input r when interacting with parties having input $\vec{x} = x_1, \ldots, x_n$ and auxiliary information $\vec{z} = z_0, \ldots, z_n$, and with a trusted party for computing a (possibly randomized) n-argument function f with random input r_f. Let the $(n+1)$-tuple

$$\text{IDEAL}_{f,\mathcal{S}}(\vec{x}, \vec{z}, r, r_f) \quad = \quad \text{ADV}_{f,\mathcal{S}}(\vec{x}, \vec{z}, r, r_f), \text{IDEAL}_{f,\mathcal{S}}(\vec{x}, \vec{z}, r, r_f)_1, \ldots, \text{IDEAL}_{f,\mathcal{S}}(\vec{x}, \vec{z}, r, r_f)_n$$

denote the outputs of the parties on inputs $\vec{x}, \vec{z}$, adversary $\mathcal{S}$, and random inputs r, r_f as described above (party P_i outputs $\text{IDEAL}_{f,\mathcal{S}}(\vec{x}, \vec{z}, r, r_f)_i$). Let $\text{IDEAL}_{f,\mathcal{S}}(\vec{x}, \vec{z})$ denote the distribution of $\text{IDEAL}_{f,\mathcal{S}}(\vec{x}, \vec{z}, r, r_f)$ when r, r_f are uniformly distributed.

Security Definition. A protocol π for computing a function f is secure if executing π in the 'real-life' setting is "equivalent" to evaluating the function f in the ideal model, in the sense that any effect on the computation that can be achieved by a real-life adversary can already be achieved by an ideal-model adversary. More precisely, an adversary is called *t-limited* if it corrupts no more than t parties at each execution. A protocol π for computing function f is t-secure if for *any* t-limited real-life adversary $\mathcal{A}$ attacking π there *exists* an ideal-model adversary $\mathcal{S}$ that has the same effect on the computation (of f) as $\mathcal{A}$, *even though $\mathcal{S}$ operates in the ideal model* (and is limited to time polynomial in the running time

[2] The auxiliary information $\vec{z}$ represents information gathered by the parties and adversary in some other interaction external to the current execution of protocol π. In particular, it is used for proving the composition theorem described in the sequel. It is stressed that the protocol, π, makes no use of this information.

[3] We stress that the adaptivity in the ideal model is only in the sense that the adversary can choose who to corrupt next based on the inputs of previously corrupted parties. This is *weaker* than the adaptive real-life adversary who also sees the internal data of the corrupted parties and the messages received by them.

of $\mathcal{A}$). The effect of an adversary on a computation is defined as follows. On any inputs for the parties, the random variable describing the outputs of all parties in the ideal model with adversary $\mathcal{S}$ should be distributed identically to the random variable describing the outputs of all parties in the real-life model with adversary $\mathcal{A}$. That is:

Definition 1: Let f be an n-argument function and let π be a protocol for n parties. We say that π *t-securely computes* f, if for any real-life adversary $\mathcal{A}$ there exists an ideal-model adversary $\mathcal{S}$ whose running time is polynomial in the running time of $\mathcal{A}$, such that for every input vector $\vec{x}$ and every auxiliary information vector $\vec{z}$,

$$\text{IDEAL}_{f,\mathcal{S}}(\vec{x},\vec{z}) \stackrel{\text{d}}{=} \text{EXEC}_{\pi,\mathcal{A}}(\vec{x},\vec{z}) \tag{1}$$

where $\stackrel{\text{d}}{=}$ stands for "identically distributed".

(Here an *n-argument function* is a function $f : D^n \times \mathcal{R}_f \to E^n$, where the last input is the random input.[4])

We stress that in (1) the two *entire* $(n+1)$-tuples describing the output of the computation must be identically distributed. This captures the requirement that the adversary's view and the outputs of the uncorrupted parties should be considered *together*. Also, note that this definition captures the intuitive notion of "not being able to learn any information from the communication" since the output of the adversary includes the communication seen by the corrupted parties.

Composition of Secure Protocols. In the sequel we use the fact that secure protocols can be composed in a modular way while maintaining security. For a full exposition and a proof see [11]. Here we briefly review the set-up and state the theorem.

Informally, the composition theorem can be stated as follows: Suppose that protocols $\rho_1, \ldots, \rho_k$ securely compute functions $f_1, \ldots, f_k$ respectively, and that a protocol π securely computes a function g using subroutine calls for "ideal evaluation" of $f_1, \ldots, f_k$. Let $\pi^{\rho_1,\ldots,\rho_k}$ be a protocol that is identical to protocol π with the exception that every subroutine call for an ideal evaluation of f_i is replaced by an invocation of the corresponding protocol ρ_i. Then, the resulted protocol $\pi^{\rho_1,\ldots,\rho_k}$ securely computes g from scratch.

We call this type of composition of protocols *modular composition.* (This notion was first suggested in [45]. There it is called *reducibility* of protocols.) In formalizing this theorem we concentrate on the case where at most one subroutine invocation is running at any computational round. Showing that security is maintained even in the more general case, where several subroutine invocations may be running at the same time, requires a stronger security property than the one presented here and is not dealt with in this paper. Yet, we remark that our protocols do enjoy this stronger security property. (See [11] for details.)

To be able to state the composition theorem, we first formulate a model for computing a function g with the assistance of a trusted party for computing a function f, and define secure protocols in that model. This model, called the *semi-ideal model* with ideal access to f (or in short the *f-ideal model*), is obtained as follows. We start with the *real-life model* described above. This model is augmented with an incorruptible trusted party T_f for computing a function f. At special rounds (determined by the protocol run by the uncorrupted parties) all parties hand their f-inputs to T_f (party P_i hands ξ_i), and are handed back their respective outputs (P_i learns $f(\xi_1, \ldots, \xi_n, r_f)_i$). As usual, the adversary can adaptively corrupt parties immediately before and immediately after the call to T_f, learn the internal data of corrupted parties, and determine the values that the corrupted parties hand T_f. (The case of ideal evaluation of several possibly different functions $f_1, \ldots, f_k$ is treated similarly, where the protocol specifies in each invocation of the trusted party which function f_j to evaluate.)

[4] In the sequel it will sometimes be convenient not to specify explicitly the random input in the description of the function. Using this representation, an n-argument function is $f : D^n \to \mathcal{D}(E^n)$, where $\mathcal{D}(E^n)$ is the set of distributions over E^n.

Given n-argument functions $f_1, \ldots, f_k$, let $\text{EXEC}^{f_1,\ldots,f_k}_{\pi,\mathcal{A}}(\vec{x}, \vec{z})$ denote the random variable describing the output of the computation in the $(f_1, \ldots, f_k)$-ideal model with protocol π, adversary $\mathcal{A}$, input $\vec{x}$ and auxiliary information $\vec{z}$ for the parties, analogously to the above definition of $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, \vec{z})$. (We stress that here π is not a real-life protocol and that it uses ideal calls to a trusted party $T_{f_1,\ldots,f_k}$.) Protocols for securely computing a function g in the $(f_1, \ldots, f_k)$-ideal model are defined in the usual way:

Definition 2: Let $f_1, \ldots, f_k$ and g be n-argument functions and let π be an n-party protocol in the $(f_1, \ldots, f_k)$-ideal model. We say that protocol π *t-securely computes g in the $(f_1, \ldots, f_k)$-ideal model* if for any t-limited adversary $\mathcal{A}$ (in the $(f_1, \ldots, f_k)$-ideal model) there exists an ideal-model adversary $\mathcal{S}$, whose running time is polynomial in the running time of $\mathcal{A}$, and such that for every input vector $\vec{x}$ and every auxiliary information vector $\vec{z}$,

$$\text{IDEAL}_{g,\mathcal{S}}(\vec{x}, \vec{z}) \stackrel{\text{d}}{=} \text{EXEC}^{f_1,\ldots,f_k}_{\pi,\mathcal{A}}(\vec{x}, \vec{z}).$$

Replacing a call of protocol π for an ideal evaluation of f_i with a call to a real-life subroutine protocol ρ_i is done in a straightforward way: the code of π within each party is changed so that the call for ideal evaluation of f_i is replaced with an invocation of ρ_i. The value to be handed to the trusted party is used as input to ρ_i; and, in addition, ρ_i is given a new, unused part of the party's random input. Once the execution of ρ_i is completed the local output is treated as the value returned by the trusted party, and the execution of π resumes. We assume that all parties terminate protocol ρ at the same round. Let $\pi^{\rho_1,\ldots,\rho_k}$ denote protocol π where each ideal evaluation call to f_i is replaced by an invocation of protocol ρ_i.

Theorem 1: [11]. Let $f_1, \ldots, f_k$ and g be n-argument functions. Let π be an n-party protocol that t-securely computes g in the $(f_1, \ldots, f_k)$-ideal model, in a way that no more than one ideal evaluation call is made at each round. Let $\rho_1, \ldots, \rho_k$ be n-party protocols that t-securely compute $f_1, \ldots, f_k$, respectively. Then, the protocol $\pi^{\rho_1,\ldots,\rho_k}$ t-securely computes g (in the real-life model).

Measuring randomness. We measure the amount of randomness used by a protocol as follows. We provide each party P_i with a random string r_i of independent and uniformly distributed symbols in the set $\{0, 1, \ldots, p-1\}$, for some p. Let d_i be the rightmost position on the tape r_i that party P_i reads. In this case we say that party P_i used $d_i \cdot \lceil \log p \rceil$ random bits.[5]

Definition 3: A d-random protocol is a protocol such that for every input assignment $\vec{x}$ and every auxiliary input $\vec{z}$, the total number of random bits used by all parties in *every* execution is at most d.

We stress that the definition allows, for example, that in different executions each individual party will toss a different number of coins. This number may depend on both the input of the parties, and previous coin tosses.

Circuits. In the sequel we represent the functions computed by the parties as arithmetic circuits. That is, we fix a prime $p > n$ (where n is the number of parties); the circuit consists of two types of gates: addition modulo p and multiplication modulo p. All gates have fan-in two, and unbounded fan-out. The size of a circuit, denoted m, is the number of gates in the circuit.

[5] It is standard to view a random selection in the set $\{0, 1, \ldots, p-1\}$ as "choosing" $\lceil \log p \rceil$ random bits. This can be justified either by Entropy considerations, or simply by the fact that to choose a random number in $\{0, 1, \ldots, p-1\}$ an *expected* number of $O(\lceil \log p \rceil)$ random bits suffices (simply choose $\lceil \log p \rceil$ random bits; if you get a number in the range $\{0, 1, \ldots, p-1\}$ output this number; otherwise, try again). Hence, any protocol that uses r random bits according to our definition can be converted into a protocol that uses expected $O(r)$ random bits in a setting where only choices in $\{0, 1\}$ are allowed. Alternatively, we can restrict ourselves to choices in $\{0, 1\}$ and consider the *worst case* number of random bits if we allow a (small) probability of failure.

We remark that a boolean circuit (e.g.,, a circuit consisting of standard Or, And, and Not gates) can be transformed into an equivalent arithmetic circuit in a way that preserves the number of gates, up to a small multiplicative factor. For instance, consider the transformation $\text{NOT } a \Rightarrow (1-a)$; $a \text{ AND } b \Rightarrow a \cdot b$; and $a \text{ OR } b \Rightarrow 1-((1-a)(1-b))$.

3 An Overview of the [6] Protocol for Passive Adversaries

Our construction for passive adversaries, described in the next section, uses components used in the [6] general construction for t-securely computing any function in the presence of passive adversaries, for any $t < n/2$. Therefore, we present in this section a brief overview of [6]. The construction (and proof) is presented in a modular way, using the composition theorem described in the previous section. This form of presentation will enable us to use components of [6] without re-proving their security from scratch.

For the [6] protocol, the parties agree on an arithmetic circuit for the function f to be computed. (This involves agreeing on a prime $p > n$; all the arithmetic in the sequel is done modulo p.) Each party holds values for some of the input wires. In addition, each random input to the circuit is assigned to some party. Each party chooses random values to the random inputs assigned to it, and from this point on treats the random inputs as regular inputs to the circuit. Finally, each output wire of the circuit is assigned to one or more parties (these are the parties that will learn the value of this wire).

First each party uses Shamir's secret-sharing scheme to share its inputs among the parties. Then, the parties evaluate the circuit in a gate-by-gate fashion (from inputs to outputs); for each gate, the parties engage in a protocol for computing shares of the output value of the gate from their shares of the input values of the gate. Finally, the parties let each party reconstruct the values of the output gates assigned to it. More precisely, the [6] protocol consists of a 'high-level' protocol for evaluating the circuit; this protocol uses as 'subroutines' protocols for secure evaluation of the following n-argument functions:

Secret Sharing. $\text{SEC-SHAR}_n(F(\cdot), \epsilon, \ldots, \epsilon) = F(1), \ldots, F(n)$, where $F(\cdot)$ is an arbitrary polynomial in $GF[p]$, and ϵ denotes the empty input. In the high-level protocol the parties will evaluate SEC-SHAR with polynomials F that are distributed uniformly among all polynomials of degree t over $GF[p]$ with some fixed free coefficient. We let $\text{SEC-SHAR}_{n,i}$ denote the function SEC-SHAR_n where the dealer (i.e., the party with non-empty input) is P_i.

Evaluating an addition gate. $\text{ADD}_n(a_1|b_1, \ldots, a_n|b_n) = a_1 + b_1, \ldots, a_n + b_n$ (where '|' denotes concatenation). This function for evaluating an addition gate is trivial and can be computed securely without any interaction between the parties.

Evaluating a multiplication gate. $\text{MULT}_n(a_1|b_1, \ldots, a_n|b_n) = C(1), \ldots, C(n)$, where C is distributed uniformly among all polynomials of degree t over $GF[p]$ with free coefficient $a \cdot b$. Here a (resp., b) is the free coefficient of the lowest degree polynomial A (resp., B) satisfying $A(i) = a_i$ (resp., $B(i) = b_i$) for all i. Note that we do not specify how the coefficients of C are chosen; this is regarded as the 'intrinsic randomness' of the function MULT.

Reconstruction. $\text{RECONS}_{n,W}(a_1, \ldots, a_n) = \alpha_1, \ldots, \alpha_n$, where $W \subseteq [n]$, and $\alpha_i = (a_1, \ldots, a_n)$ if $i \in W$, and $\alpha_i = \epsilon$ otherwise. In the high-level protocol the parties in W will interpolate a (degree t) polynomial A satisfying $A(i) = a_i$ for all i, and will output $A(0)$.

Theorem 2: [6]. Let $t < n/2$. Then, there exist protocols for t-securely computing each of the above four functions, in the presence of passive adversaries, for all $i \in [n]$ and $W \subseteq [n]$.

We do not prove this theorem here. Yet we note that the the protocols for computing $\text{SEC-SHAR}_{n,i}$ and $\text{RECONS}_{n,W}$ are just Shamir's secret sharing and reconstruction protocols [49]. The sharing protocol requires the dealer to choose t random values in $GF[p]$; namely $O(t \log p)$ random bits. The function ADD_n can be computed by each partly locally summing its two inputs. Below we sketch Rabin's simplification of the protocol for securely computing MULT_n, as it appears in [27]. This protocol requires each participating party to choose $O(t \log p)$ random bits (hence total of $O(nt \log p)$ random bits in each invocation of multiplication protocol). For completeness, we also state the following theorem:

Theorem 3: [6] Let $t < n/2$. Then, given an arithmetic circuit for computing an n-argument function f there exists a protocol for t-securely computing f in the semi-ideal model with passive adversaries and with ideal access to the functions $\text{SEC-SHAR}_{n,i}$, ADD_n, MULT_n and $\text{RECONS}_{n,W}$, for all $i \in [n]$ and $W \subseteq [n]$.

Using the composition theorem (Theorem 1), we get that for any $t < n/2$ there exist protocols for t-securely computing any n-argument function in the presence of passive adversaries.

The [27] multiplication step. First, each party P_i locally computes the value $d_i = a_i \cdot b_i$. These values define a polynomial $D(x)$ whose free coefficient is the value $a \cdot b$. However, the degree of D is $2t$ (and not t) which may lead to problems in revealing the output at the end. In addition, D is not even a random polynomial of degree $2t$ (as D cannot be irreducible). We overcome these problems as follows. We show below that there is a linear combination

$$D(0) = \sum_{i=1}^{2t+1} \gamma_i D(i), \tag{2}$$

where the γ_i's are known coefficients. Once this is established, the parties can proceed as follows: Each party P_i, $(1 \le i \le 2t+1)$ locally computes $\alpha_i = \gamma_i \cdot D(i)$. It then chooses a random polynomial $\Delta_i(x)$ of degree t whose free coefficient is α_i. It shares this polynomial among the parties (i.e., P_i sends $\Delta_i(j)$ to P_j). Each party P_j sums up the $2t+1$ shares that it receives. This sum is just the share of P_j for the polynomial $\Delta(x) = \sum_{i=1}^{2t+1} \Delta_i(x)$ which is a random, degree t, polynomial whose free coefficient is $\sum_{i=1}^{2t+1} \alpha_i = D(0)$.

It remains to show γ_i's that satisfy Equation (2). Denote by $\vec{d} = (d_0, d_1, \ldots, d_{2t})$ the vector of coefficients of the polynomial D and let V be the $(2t+1) \times (2t+1)$ Vandermonde matrix whose (i,j) entry (for $1 \le i,j \le 2t+1$) contains the value i^{j-1}. Also denote, $\vec{D} = (D(1), D(2), \ldots, D(2t+1))$. With this notation we get that $\vec{D} = V \cdot \vec{d}$. Since V is non-singular (see, e.g., [51]), we can write $\vec{d} = V^{-1} \cdot \vec{D}$ and note that the value that we are interested in sharing is $D(0) = d_0$, the first element of $\vec{d}$, which can therefore be written as $D(0) = d_0 = \sum_{i=1}^{2t+1} V_{1,i}^{-1} \cdot D(i)$ (where V^{-1} is a fixed matrix).

4 Our Protocol for Passive Adversaries

In this section we present our randomness-efficient protocol with respect to passive adversaries. For simplicity, we restrict the presentation to *deterministic* functions where each party has boolean input and output. That is, we prove the following theorem:

Theorem 4: Let $t < n/2$. Then, any function $f : \{0,1\}^n \to \{0,1\}^n$ that has a circuit of size m, can be t-privately computed by a $O(t^2 \log n + (m/n)t^5 \log t)$-random protocol.

We first describe our protocols assuming the existence of a trusted dealer whose role is restricted to distributing random values to the parties. The trusted dealer does not receive any messages and has no input. Later, in Section 4.3, we eliminate this assumption by using $t+1$ parties who will perform the task of the trusted dealer (in addition to their other tasks).

4.1 Overview

Known generic constructions of protocols for secure computations share the following structure, described in the previous section: First, each party shares its input; next, the parties evaluate the given circuit in a gate-by-gate manner from inputs to outputs, maintaining the property that the value of each wire in the circuit is shared among the parties. Finally, the parties reconstruct the value of the output wire from their shares. Our approach can be applied to any protocol that follows this outline. For concreteness, however, we concentrate on the [6] construction (reviewed in the previous section).

We develop a variation of the above outline. Instead of having the value of each wire shared among *all* parties, and having *all* parties participate in evaluating each gate, we use a different method. We partition the parties into sets of size $s = 2t + 1$ which we call *teams*. The input of each party will be shared only among the members of its team (using the [49, 6] secret-sharing procedure). Each gate will be assigned a team, and will be evaluated only by the parties in that team. Consequently, the output wire of each gate will be shared among the parties in the corresponding team. Each of the $\approx \frac{n}{s}$ teams will be assigned to roughly $m/(n/s) = m \cdot s/n$ gates.

To evaluate a gate g, each party of the corresponding team T first receives a share of the value of each of the two input wires to the gate g. These shares are communicated by the parties of the teams that evaluated the gates leading to those wires. Now team T invokes the [6] procedure for evaluating gate g. (This can be done since $s > 2t$.) At the end of this computation, the parties in T hold shares of the output wire of the gate. When the values of the output wires of the circuit are known, the corresponding teams provide the specified parties with the information needed to reconstruct these values.

The parties' randomness (needed for sharing their inputs and for the gate evaluations) is provided by the trusted dealer. Our method ensures that the view of each subset of at most t parties depends only on a "small" fraction of the total number of dealt random values. Thus, the random values dealt to the parties may have only limited independence, which leads to saving in overall randomness.

4.2 Detailed Description

We now state the protocol in detail, assuming a trusted dealer. Let $s = 2t+1$ and let $n = k \cdot s$ (we assume for convenience that n is divisible by s; see Remark 1 below). We partition the n parties into k teams of parties of size s each. Each team will evaluate (at most) $\ell = \lceil \frac{m}{k} \rceil$ gates. We also specify an enumeration of the parties in each team. Denote by $P_{T,j}$ the j'th party in team T. Let $p > s$ be a prime (to be determined later). All the computations described below are over $GF[p]$. We first describe the "high-level" protocol in the semi-ideal model with access to ideal evaluation of the functions $\text{SEC-SHAR}_{s,i}$, ADD_s, MULT_s and $\text{RECONS}_{s,W}$. (These functions were described in the previous section.)

1. (Input Sharing)
   For each party $P_{T,j}$ the parties in team T invoke the trusted party for ideal evaluation of $\text{SEC-SHAR}_{s,j}$. The input of the dealer, $P_{T,j}$, is chosen at random among all polynomials of degree t whose free coefficient is the dealer's input to the computation.

2. (Computation)
   The gates of the circuit are evaluated one by one from the inputs to the outputs. Each gate g is evaluated by the parties in the team T assigned to it, as follows.

   - *Collect shares of inputs to the gate ("baton hand off"):*

   Let x and y be the input wires of gate g, and let T_x and T_y denote the teams that hold the shares for these inputs (the inputs x and y may come from either the inputs for the circuit, as shared in the

INPUT SHARING stage, or from the outcome of previously evaluated gates).[6] Then, the ith party in T_x and the ith party in T_y send their shares of the values of x and y, respectively, to $P_{T,i}$ (i.e., the ith party in T). Let a_i (resp., b_i) denote the value received from $P_{T_x,i}$ (resp., $P_{T_y,i}$). Now, for each of the two input wires to gate g, the parties in T hold shares of a polynomial of degree t whose free coefficient is the value of that wire.

- *Compute shares for the output of the gate:*

Once the parties in T receive their shares of the input wires, they evaluate the gate by invoking the trusted party for evaluating the appropriate function (i.e., either ADD_s or MULT_s).

At the end of this step the parties in T hold shares of a polynomial of degree t (over $GF[p]$) whose free coefficient is the value of the gate.

3. (OUTPUT)
Let T be a team that computes the value of an output wire of the circuit. Then the parties in T invoke the trusted party for evaluating the reconstruction function $\text{RECONS}_{s,W}$ where W is the set of parties that are assigned to this wire. Next, each party in W interpolates a (degree t) polynomial A satisfying $A(i) = a_i$ for all i, and outputs $A(0)$.

The high-level protocol above is turned into a full-fledged protocol by replacing the ideal evaluation calls with subroutines that securely evaluate the corresponding functions; for concreteness, the [6] subroutines sketched in the previous section.

We now turn to describe how the dealer distributes the random numbers (in $GF[p]$) to the parties. In the INPUT SHARING stage the dealer distributes coefficients of n polynomials (one polynomial to each party). Then, the dealer distributes additional s polynomials per each gate to be computed (one polynomial for each party in the simulating team). Each polynomial is defined by t coefficients in $GF[p]$. Therefore, the dealer generates a total of $M = n \cdot t + m \cdot s \cdot t$ numbers (in $GF[p]$). In order to save in randomness, the dealer does not generate these M numbers independently. Instead, we observe that the view of each subset (of size at most t) of parties depends on a "relatively small" set of at most $\beta = \Theta((m/n) \cdot t^4)$ numbers, as follows:

- The number of *shares* that a single party P_i sees is counted as follows. $s - 1$ shares are seen in the INPUT SHARING stage (one share from each member of P_i's team). For each of the (at most) ℓ multiplication gates that P_i evaluates, it gets messages that depend on the inputs and outputs of all members of P_i's team. These add up to at most $O(\ell s)$ shares. Hence, any set of t parties sees at most $O(t \cdot \ell \cdot s)$ shares which are thus depending on at most $O(t^2 \cdot \ell \cdot s)$ numbers that the dealer distributes as coefficients of polynomials.

- In addition, every party P_i receives some numbers directly from the dealer: t numbers in the INPUT SHARING stage (to share its input among its team members); plus, for each of the (at most) ℓ multiplication gates that P_i takes part in their evaluation, it gets t numbers (coefficients of a polynomial to be used for sharing its value $D(\cdot)$). All together, a set of t parties gets $O(st\ell)$ numbers directly from the dealer.

To conclude, by the choice of parameters ($s = \Theta(t)$, and $\ell = \Theta(m/k) = \Theta(m \cdot t/n)$), the view of a subset of at most t parties depends on at most $O(t^2 \ell s) = O(t^3 \ell) = O(t^4 \frac{m}{n})$ numbers from the distribution. Hence the dealer can generate M numbers $Z_1, Z_2, \ldots, Z_M$ in $GF[p]$ which are uniformly distributed and are $\beta = \Theta((m/n) \cdot t^4)$-wise independent. This may be done as follows. Set $p > M$. The dealer chooses a random polynomial $R(x)$ in $GF[p]$ of degree β and then generates the M numbers $Z_1 = R(1), Z_2 =$

[6] T, T_x and T_y need not be distinct.

$R(2), \ldots, Z_M = R(M)$. It is a well known fact (and easy to prove) that if $p > M$ these M numbers are β-wise independent. This procedure uses $O(\beta \cdot \log p) = O((m/n) \cdot t^4 \cdot \log m)$ bits of randomness.

The amount of randomness used can be further reduced using a more careful analysis of the needed independence of the numbers generated by the dealer. The view of any subset of size t of parties indeed depends on at most $\beta = \Theta((m/n) \cdot t^4)$ numbers generated by the dealer. However, we do not need *all* subsets of size β to be uniformly distributed. It suffices that the $\binom{n}{t}$ subsets of size β, defined by the $\binom{n}{t}$ subsets of t parties, be uniformly distributed. To take advantage of the relaxed requirement, we use an extension of the results of [50, 40] (which, for self containment, appears in Appendix A): Set $p > 2t + 1$. The dealer will uniformly sample a space of M-tuples over $GF[p]$, which is constructed to suit the specific $\binom{n}{t}$ subsets. By [50, 40], there is a sample space of size $\binom{n}{t}p^\beta$ such that if we sample the space uniformly, then the projection of the chosen vector on any of the $\binom{n}{t}$ subsets is uniformly distributed.[7] To sample this space, the dealer will need randomness of $O(t \log(n/t) + (m/n)t^4 \log p)$ bits.

Proof of Theorem 4 [Trusted dealer case]. Let $t < n/2$, and let f be the computed function. Fix an arithmetic circuit for f and a prime p. Let π be the (full-fledged) protocol described above with respect to that circuit. As explained, protocol π (in its trusted dealer version) is $O(t \log n + (m/n)t^4 \log t)$-random. We show that protocol π satisfies the conditions of Definition 1 via the following two claims. Let π_R be a protocol identical to protocol π with the exception that the trusted dealer deals totally random values to the parties.

Claim 1: Protocol π_R t-securely computes f. That is, for any t-limited (passive) real-life adversary $\mathcal{A}$ there exists an ideal-model adversary $\mathcal{S}$ such that for all inputs $\vec{x}$ and any auxiliary inputs $\vec{z}$

$$\text{IDEAL}_{f,\mathcal{S}}(\vec{x}, \vec{z}) \stackrel{\text{d}}{=} \text{EXEC}_{\pi_R,\mathcal{A}}(\vec{x}, \vec{z}).$$

Claim 2: For any input $\vec{x}$ and auxiliary input $\vec{z}$, and for any real-life adversary $\mathcal{A}$, the global output of the parties in π and π_R is identically distributed. That is, $\text{EXEC}_{\pi_R,\mathcal{A}}(\vec{x}, \vec{z}) \stackrel{\text{d}}{=} \text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, \vec{z})$.

The above two claims (to be proven below) imply Theorem 4. $\square$

Proof of Claim 1. Let $\hat{\pi}$ denote the high level protocol that corresponds to protocol π_R in the semi-ideal model with ideal evaluation access to the functions $\text{SEC-SHAR}_{s,i}$, ADD_s, MULT_s and $\text{RECONS}_{s,W}$. It suffices to show that $\hat{\pi}$ is t-secure in the semi-ideal model. Theorems 1 and 2 then imply that protocol π is t-secure in the real-life model as well.

Given a real-life adversary $\mathcal{A}$, the ideal-model adversary $\mathcal{S}$ proceeds via a simulation of $\mathcal{A}$. Adversary $\mathcal{S}$ starts running $\mathcal{A}$ on its auxiliary input z_0 and random input r_0. Next, $\mathcal{A}$ may corrupt parties, and will expect to see the internal data and the messages received by the corrupted parties. $\mathcal{S}$ proceeds as follows. (Recall that $\mathcal{S}$ has to simulate the internal data of parties upon corruption and the messages sent from uncorrupted parties to corrupted parties.)

- Whenever the real-life adversary $\mathcal{A}$ issues a "`corrupt` P_i" command, the ideal-model adversary $\mathcal{S}$ also corrupts P_i (in the ideal model) and receives P_i's input, x_i, and auxiliary input z_i. Then, $\mathcal{S}$ hands $\mathcal{A}$ the values x_i, z_i, and the following completion of P_i's internal data. P_i's internal data consists of P_i's local input and output values from the already evaluated ideal calls. $\mathcal{S}$ sets these values to be consistent with the data already known to the adversary; whenever a value is not determined it is chosen at random from

[7] Both [50, 40] deal with the field $GF[2]$ but can be extended to $GF[p]$. We note that the time-complexity of sampling in the sample space is $poly(n, \log \binom{n}{t}, \beta)$ but the complexity of the known algorithms that find such a space is $poly(\binom{n}{t})$. Note however that this can be done "off-line" and can be hard-wired into the protocol.

its domain. That is: The input to the evaluation of $\text{SEC-SHAR}_{n,i}$ is set to a random polynomial of degree t with free coefficient x_i. The outputs of all other evaluations of SEC-SHAR are set to random numbers in $GF[p]$. For each gate that P_i's team has already evaluated, the simulator sets the shares that P_i received from uncorrupted parties to be random numbers in $GF[p]$. If the gate is an addition gate then P_i's output value is determined by its inputs. If the gate is a multiplication gate then P_i's output value is set to a random number in $GF[p]$. We show how to simulate he outputs of RECONS below.

- For each party $P_{T,i}$, the ideal-model adversary $\mathcal{S}$ simulates an interaction of team T with the trusted party for computing $\text{SEC-SHAR}_{s,i}$. That is, if $P_{T,i}$ is corrupted then $\mathcal{S}$ receives the polynomial F handed by $\mathcal{A}$ to the trusted party, and hands $\mathcal{A}$ the value $F(j)$ for each corrupted party $P_{T,j}$. If $P_{T,i}$ is not corrupted then, for each corrupted party in team T, the ideal-model adversary $\mathcal{S}$ hands $\mathcal{A}$ a random number in $GF[p]$ as the value given by the trusted party.
- For each gate g in the circuit, $\mathcal{S}$ simulates the "baton hand-off" step of the shares of the input lines to the gate. That is, let T be the team that computes gate g, and let T_1, T_2 be the teams that hold the values of the input lines to the gate. Then, for each i, if $P_{T,i}$ is corrupted and $P_{T_1,i}$ (resp., $P_{T_2,i}$) is not corrupted, then $\mathcal{S}$ hands $\mathcal{A}$
- Once the "baton hand-off" step of a gate g is simulated, $\mathcal{S}$ simulates an interaction of team T with the trusted party for computing the function that corresponds to gate g (i.e., either ADD_s or MULT_s). If the gate g is an addition gate then $\mathcal{S}$ hands $\mathcal{A}$ the sum of the two input values given by each corrupted party in team T to the trusted party. If the gate is a multiplication gate then $\mathcal{S}$ hands $\mathcal{A}$ a random number in $GF[p]$ as the value given by the trusted party to each corrupted party in team T.
- When the simulation of a gate leading to an output wire of the circuit is complete, $\mathcal{S}$ simulates an interaction with the trusted party for computing $\text{RECONS}_{s,W}$, where W is the set of parties that are to learn the value of this wire. If no corrupted party is in W then $\mathcal{S}$ need do nothing. Otherwise, $\mathcal{S}$ invokes its own trusted party for the output value of the main function, f. Let v be the output value that corresponds to this output wire, let T be the team that holds the value of this wire, and let a_i be the share that each corrupted party $P_{T,i}$ in T hands the trusted party for $\text{RECONS}_{s,W}$. Then, $\mathcal{S}$ chooses a random polynomial A of degree t such that $A(0) = v$ and $A(i) = a_i$ for each corrupted party $P_{T,i}$. Next, for each corrupted $P_{T,i}$ the simulator $\mathcal{S}$ hands $\mathcal{A}$ the vector $(A(1), \ldots, A(s))$. (Note that this can always be done since $\mathcal{A}$ corrupts at most t parties.)

Once the simulation of $\mathcal{A}$ is complete, the ideal model adversary $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

Analysis of simulator $\mathcal{S}$. We show that for all inputs $\vec{x}$ and auxiliary inputs $\vec{z}$ we have

$$\text{IDEAL}_{f,\mathcal{S}}(\vec{x},\vec{z}) \stackrel{\text{d}}{=} \text{EXEC}^{\text{SEC-SHAR,ADD,MULT,RECONS}}_{\hat{\pi},\mathcal{A}}(\vec{x},\vec{z}).$$

Recall that each one of the random variables $\text{IDEAL}_{f,\mathcal{S}}(\vec{x},\vec{z})$ and $\text{EXEC}^{\text{SEC-SHAR,ADD,MULT,RECONS}}_{\hat{\pi},\mathcal{A}}(\vec{x},\vec{z})$ consists of the outputs of the parties plus the adversary output. In both random variables the uncorrupted parties have the same outputs, determined by the fixed inputs. (In the ideal model execution, this value is given by the trusted party. In the run of π this follows from the definition of the functions SEC-SHAR, ADD, MULT, RECONS, and from the fact that the circuit computes the function.) It remains to argue that the adversary outputs are identically distributed, given the fixed inputs and outputs of the computed function.

We show, by induction on the number of rounds, that the adversary views of the real and simulated executions are identically distributed.[8] To see this, first note that the support sets of the two distributions are identical (that is, each view that is possible in the real interaction is possible in the simulated interaction

[8]The adversary view consists of its random and auxiliary inputs, followed by the internal data of the corrupted parties and the messages received by them.

and vice versa). Then, fix a prefix of the adversary view up to some round, and consider the probability of any continuation of this prefix to the next round in the real and simulated interactions.

To see that any continuation is equally probable in the two interactions, we consider the three possible types of components of the adversary view in a given round:

1. Messages arriving from the trusted party, regarding an evaluation of either SEC-SHAR of MULT. In an interaction in the semi-ideal model, $\mathcal{A}$ receives up to t shares of a random polynomial of degree t with fixed and unknown free coefficient. In the simulated interaction, $\mathcal{A}$ receives independently chosen random numbers in $GF[p]$. However, the two distributions are identical.

2. Messages arriving from the trusted party, regarding an evaluation of RECONS. In both interactions these are values of a polynomial that is uniformly distributed among all degree t polynomials whose free coefficient is equal to the value v of the corresponding output wire of the circuit on inputs $\vec{x}$, and who match the values held by the corrupted parties.

3. Internal data of a corrupted party, P_i. These are of four types:

   (a) Internal data regarding an invocation of SEC-SHAR. If P_i is not the dealer in this invocations then its local data is only the function value, which is a random value in $GF[p]$.

   If the party is the dealer in this invocation then this data consists of the shared value (i.e., the input x_i of the party), plus the (random) coefficients of the degree-t polynomial $p_i()$ used to share x_i. In both interactions the inputs are the same x_i and the coefficients are uniformly distributed, given the values known to $\mathcal{A}$ (i.e., the values $p_i(j)$ for corrupted parties P_j), and given that the free coefficient is x_i.

   (b) Internal data regarding an invocation of MULT. In both interactions, P_i's inputs and outputs of this invocation are random values in $GF[p]$.

   (c) Internal data regarding an invocation of ADD. In both interactions, P_i's inputs of this invocation are two random values in $GF[p]$, and the output values are the sum of the input values.

   (d) Internal data regarding an invocation of RECONS. Recall that here the output is the concatenation of the inputs of all parties. Thus, in both interactions, P_i's input is implied by the public output.

This completes the proof of Claim 1. □

Proof of Claim 2. To prove the claim, we first observe the structure of the information of a particular party P_i. This information (in addition to the party's input x_i and auxiliary input z_i) consists of:

1. Random numbers, received from the dealer, to be used by P_i as coefficients of polynomials (using which P_i will share its information). These include t coefficients of the polynomial Q_i that P_i receives in the INPUT SHARING stage (to be used to share its input); and, during the COMPUTATION stage, for each of the (at most ℓ) gate-evaluation in which P_i participates it receives additional t coefficients to be used in the evaluation. Altogether, at most $t(\ell+1) = O(t \cdot \ell)$ numbers to be used as coefficients.

2. Shares of values, sent to $P_i = P_{T_u,r}$ by other parties (each such share is the value of $Q(r)$ for some polynomial Q of degree t whose free coefficient is some information S). Specifically, these are:

   - For each party P_j in P_i's team, T_u, a share of P_j's input; this message can be written as $Q(r) = \sum_{m=1}^{t} Z_{k_m} r^m + S$, for some numbers Z_{k_m} provided by the dealer and $S = x_j$.

- For each of the (at most) ℓ multiplication gates in the evaluation of which P_i participates, P_i receives during the computation from each party P_j of its team a share of a value S that P_j computes locally. This message can be written as $Q(r) = \sum_{m=1}^{t} Z_{k_m} r^m + S$, for some Z_{k_m} provided by the dealer.

- For each such gate, P_i also receives shares of the two inputs on which the computation is to be performed (unless it already has these shares). Each such message can be written as $Q(r) = \sum_{j=1}^{s} (\sum_{m=1}^{t} Z_{k_{j,m}} r^m + S_j)$, where each summand is a share generated by one of the parties in the team that evaluated the previous gate, during the evaluation. The $Z_{k_{j,m}}$ are numbers provided by the dealer to these parties. (The case where the input to the gate is one of the x_j's is slightly simpler.)

Altogether, P_i receives a total of $s + \ell(s+2)$ shares.

Next, we examine the messages that an arbitrary subset (coalition) of parties of size at most t can see. Each of the messages of type 1 received by these parties is just a different number Z_k in the space generated by the dealer. Altogether, the messages of type 1 seen by the parties in the subset are just $O(t^2\ell)$ of the Z_k's.

For messages of type 2, observe that each message can be associated with a polynomial as discussed above. Each polynomial is defined by a free coefficient S, and t numbers Z_k used as coefficients and provided by the dealer to party, say, P_j (this is not necessarily the party that sends the message; the party that sends the message may only relay on it). For a party P_j, we denote by $\delta_{j,d}$ the d'th polynomial that this party creates and uses. Considering the sets of numbers Z_k used in each polynomial in the protocol, we observe that the sets are pairwise disjoint. Furthermore, for each message which is associated with a polynomial $\delta_{j,d}$, for P_j which is *not* in the set of parties under consideration, these numbers are also distinct (by definition) from any of the Z_k's directly received by any corrupted party.

The total number of shares a subset of at most t parties can see is $t \cdot (s + \ell \cdot (2s+s)) = O(t \cdot \ell \cdot s)$. Each of these shares can be, in the worst case, of a different polynomial; therefore, these shares may depend on at most $O(t^2 \cdot \ell \cdot s)$ of the Z_k's. Together with the $O(t^2\ell)$ numbers Z_k that the set of parties sees directly from the dealer (as messages of type 1), we have that the communication seen by the set of parties depends on at most $O(t^2 \cdot \ell \cdot s)$ of the Z_k's. Using $s = \Theta(t)$, and $\ell = \Theta(m/k) = \Theta(m \cdot t/n)$ we have that the communication seen by the set of parties depends on at most $O((m/n)t^4)$ values Z_k. To generate these numbers, the dealer sampled a sample space of vectors over $GF[p]$ such that the projection of the chosen vector on specific subsets, including the subset of numbers that the view of the present subset of parties depends on, is uniformly distributed.

If the adversary is not adaptive (that is, the set of corrupted parties is chosen in advance), the above discussion immediately gives us the desired claim. To see that, observe that the output of the adversary is the full communication seen by the corrupted parties. Therefore, given $\vec{x}$, the distribution would be the same if the dealer deals totally independent numbers, or if it chooses them according to our scheme.

If the adversary is adaptive then a slightly more careful argument is needed, since we cannot argue that the distribution of the adversary's output depends on some *specific* subset of the Z_k's. To prove the claim for this case, construct a tree. The leaves of the tree will be labeled by the communication seen by the adversary in a certain execution. The tree is defined inductively, on the rounds of the computation. At each round there are two levels. The first level spans the different options the adversary may take at this point. The second level spans the different options the communication (as seen by the adversary) may take; that is, if the communication seen by the corrupted parties at the round in question depends on a new Z_k, then the node would span the different values Z_k may take. The probability of a particular leaf of the tree is clearly the product of the probabilities along the path leading to it. Looking at such a path, the above arguments on the dependence of the communication on a small subset of the Z_k's ensure that the distribution of any particular branch is uniform, even given its location in the tree. Therefore, there is no difference between the probabilities of a given leaf in the tree describing the computation with real independent numbers, and numbers with limited independence. □

Remark 1: In the above we assume that n is divisible by s. If not, then $n = ks + r$ for some $0 < r < s$. In this case we let the last r parties share their inputs among the parties of the first team and then these r parties do not further participate in the protocol. When one of their inputs is required then the first team will provide the corresponding shares. This implies that the view of parties in the first team contains slightly more messages and hence requires slightly increasing the value of β (by a constant factor).

4.3 The Protocol Without Trusted Dealer

The above description assumes a trusted dealer whose role is restricted to choosing random integers in $GF[p]$ and distributing them to the parties. This can be best viewed, in a modular way, if we think of the above protocol as a protocol in the semi-ideal model with ideal access to a function RAND_n. This is a function to which each party has the empty input ϵ returns a subset of $Z_1, \ldots, Z_M$, distributed as specified by the above protocol. (Note that the function RAND is called only once, when the protocol starts.)

To eliminate the trusted-dealer assumption, we describe the following simple protocol for securely evaluating RAND_n. Applying the composition theorem once more, we obtain a protocol that securely evaluates any function without a trusted dealer.

The protocol for computing RAND_n proceeds as follows. We designate $t+1$ parties (say, $P_1, \ldots, P_{t+1}$) who, in addition to their other roles in the protocol, will play the role of the trusted dealer. Each one of the designated $t+1$ parties generates $M = n \cdot t + m \cdot s \cdot t$ integers in $GF[p]$, as described for the trusted dealer. Each of the designated parties then distributes the numbers to the n parties as described for the trusted dealer. Each of the n parties sums the $t+1$ corresponding numbers (over $GF[p]$), and considers this sum as the output of RAND_n (i.e., as the number provided by the "trusted dealer", for use in the protocol).

First, note that the amount of randomness used by this protocol is larger by a factor of $t+1$ than the amount of randomness used by the protocol with the trusted dealer. To see the intuition why this works, think of a coalition of $t+1$ parties in the non-adaptive scenario. Then, even if the adversary controls t of the $t+1$ designated parties, then for every choice of values by these parties the choice of the good party makes the sum of all $t+1$ vectors a β-independent uniformly distributed M-tuple

For a more precise proof of security, we make the following observations. We note that since the function RAND is given no real input the security requirements are quite simple. The simulator in this case works as follows: (1) If the adversary corrupts a non-dealer party, then the simulator passes to the adversary $t+1$ numbers in $GF[p]$ (as the $t+1$ messages received from the dealers). If any of the dealers was already corrupted then the corresponding number was already fixed by the simulator and is already known to the adversary; for every dealer which is not-corrupted (so far) the corresponding number is chosen as a random number in $GF[p]$, The output of such a party is the sum of these $t+1$ numbers. (2) If the adversary corrupts one of the dealers, then the simulator has to choose the M numbers that that dealer will send in the protocol. This choice has to be consistent with the numbers that were already given to the (at most t) parties corrupted so far. Recall that the M numbers are generated as the values $R(1), \ldots, R(M)$ of some degree β polynomial R. The adversary was already given some values on this polynomial, as handed to him by the at most t parties it has already corrupted. However, as proved before these t parties together receive no more than β such values. Therefore, the simulator can pick a random degree β polynomial consistent with the values handed to the adversary so far.[9] Finally, the simulator also has to give the

[9] It can be shown that the [40] construction (see Appendix) also has the property that for a partial (legal) M-tuple one can find a random consistent M-tuple.

adversary values for the numbers that this dealer is supposed to receive from the other t dealers. This is done in the same way as above.

Theorem 1 and the trusted dealer version of Theorem 4 then imply that the modified protocol is t-secure in the real-life model as well.

5 An Overview of the [6] Protocol for Active Adversaries

The general outline of the [6] construction for the case of active (Byzantine) adversaries is very similar to the case of passive adversaries. Yet, the definitions of the four 'building blocks', SEC-SHAR, ADD, MULT, RECONS, have to be modified to reflect the additional power of the adversary. That is, we now define the following n-argument functions:

Verifiable Secret Sharing. $\text{VSS}_n(F(\cdot), \epsilon, \ldots, \epsilon) = \alpha_1, \ldots, \alpha_n$, where $\alpha_i = F(i)$ if $F(\cdot)$ is a degree t polynomial over $GF[p]$, and $\alpha_i = \perp$ otherwise. (The value $\perp$ denotes the fact that the secret sharing has failed, and no secret is being shared.) We let $\text{VSS}_{n,i}$ denote the function VSS_n where the dealer (i.e., the party with non-empty input) is P_i. As in the case of passive adversaries, in the high-level protocol uncorrupted dealers will evaluate VSS with polynomials F that are distributed uniformly over all polynomials of degree t in $GF[p]$ with some fixed free coefficient.

Evaluating an addition gate. The function for evaluating an addition gate remains unchanged: $\text{ADD}_n(a_1|b_1, \ldots, a_n|b_n) = a_1 + b_1, \ldots, a_n + b_n$.

Evaluating a multiplication gate. $\text{ACT-MULT}_n(a_1|b_1|c_1, \ldots, a_n|b_n|c_n) = C(1), \ldots, C(n)$, where C is distributed uniformly over all polynomials of degree t in $GF[p]$ that meet the following requirements: **(I).** Let A (resp., B) be the lowest degree polynomial such that $A(i) = a_i$ (resp., $B(i) = b_i$) for at least $n - t$ of the parties. Then, $C(0) = A(0) \cdot B(0)$. **(II).** If $c_i \neq \epsilon$ then $C(i) = c_i$. As in the case of passive adversaries, we do not specify how the (random) coefficients of C are determined; this is regarded as the 'intrinsic randomness' of the function ACT-MULT.

Uncorrupted parties P_i will evaluate ACT-MULT_n with $c_i = \epsilon$. We introduce the c_i's in order to capture the fact that an active adversary may be able to fix (or influence) its own shares of the polynomial C. Yet, this capability of the adversary does not interfere with the secure evaluation of the function. (In particular, the [6] multiplication step allows the adversary to have such harmless influence.)

Reconstruction. The reconstruction function also remains unchanged: $\text{RECONS}_{n,W}(a_1, \ldots, a_n) = \alpha_1, \ldots, \alpha_n$, where $W \subseteq [n]$, and $\alpha_i = (a_1, \ldots, a_n)$ if $i \in W$, and $\alpha_i = \epsilon$ otherwise. In the high-level protocol the parties in W will interpolate a (degree t) polynomial A satisfying $A(i) = a_i$ for at least $n - t$ values i, and will output $A(0)$.

Theorem 5: [6]. Let $t < n/3$. Then there exist protocols for t-securely computing the above four functions in the presence of active adversaries, for all $i \in [n]$, $W \subseteq [n]$.

We do not prove this theorem here. Yet we sketch below the [6] constructions for computing VSS and ACT-MULT. For completeness, we state also the following theorem:

Theorem 6: [6] Let $t < n/3$. Given an arithmetic circuit for computing an n-argument function f, there exists a protocol for t-securely computing f in the semi-ideal model with active adversaries and with ideal access to functions $\text{VSS}_{n,i}$, ADD_n, ACT-MULT_n and $\text{RECONS}_{n,W}$, for all $i \in [n]$, $W \subseteq [n]$.

Using the composition theorem (Theorem 1), we get that there exist protocols for t-securely computing any n-argument function in the presence of active adversaries for any $t < n/3$.

The [6] VSS protocol. Here a Verifiable Secret Sharing (VSS) scheme is used instead of Shamir's secret sharing. (VSS was introduced by [19]; different VSS schemes are described in [19, 28, 6, 23, 16, 5].) Essentially, a VSS scheme makes sure that an honest dealer can successfully share a secret in a recoverable way, while guaranteeing that even if the dealer is corrupted, at the end of the sharing protocol the uncorrupted parties hold shares of a well defined and reconstructible value. Several VSS schemes exist. We sketch one of the schemes described in [6], that withstands $t < n/3$ faults. The dealer, sharing a secret s, chooses a random bivariate polynomial H of degree t in each variable, whose free coefficient is s. That is, $H(x,y) = \sum_{i,j=0}^{t} h_{i,j}x^i y^j$, where $h_{0,0} = s$ and the other coefficients are random. Next, the dealer sends the polynomials $f_i(\cdot) = H(i,\cdot)$ and $g_i(\cdot) = H(\cdot,i)$ to each P_i. Then, each P_i sends $f_i(j)$ to each P_j, and verifies that the value received from P_j equals $g_i(j)$. (Note that $f_i(j) = H(i,j) = g_j(i)$.). If any of its verifications fails, the party requests the dealer to make the corresponding value (i.e., $H(i,j)$) public. Next, each party P_i inspects all publicized values. If any of these values doesn't match P_i's private shares, then P_i requests the dealer to make P_i's share public. Again, the parties inspect the public shares. If a party P_i finds any inconsistency with its private share then it decides to abort this sharing. Otherwise, it sets its share of the secret to be $f_0(i)$.

The reconstruction protocol (i.e., the protocol for computing RECONS) is simple: all parties broadcast their shares. It is guaranteed that if the sharing protocol completed successfully then a unique polynomial $H(\cdot,\cdot)$ will be reconstructed (using error correcting techniques of Generalized Reed-Solomon codes.)

The [6] ACT-MULT protocol. In the passive adversaries case evaluating a multiplication gate consisted of each party re-sharing a locally computed value, followed by local evaluation of a linear combination of the newly received shares. Here we follow this method, with two modifications.

First, each party re-shares the locally computed value using a VSS scheme. It should be noted that the local evaluation of the linear combination on the newly received shares can still be done, since the share of each party is still a value $f_0(i)$ of a random polynomial whose free coefficient is the secret. Next, the parties have to prove that the re-shared values are indeed the product of their shares of the input wires to the gate. This is done as follows. Note that all the values that were properly shared 'sit on a polynomial' of degree t. Thus the set of values shared by the parties can be regarded as a perturbed code-word of a Reed-Solomon code, where the erroneous entries correspond to the parties that shared incorrect values. Note that no party knows all these values. Still, the parties hold shares of these values. The parties use their shares to reconstruct the *syndrome* vector of this code-word. This syndrome, while revealing no information on the values that were honestly shared, identifies the parties that shared incorrect values. These shares are sieved out in the computation of the linear combination.

6 Our Protocol for Active Adversaries

In previous sections we showed how to compute any function t-privately with a $O(t^2 \log n + (m/n)t^5 \log t)$-random protocol. In this section we extend the result to the case of active ("Byzantine") adversaries. For this, we will need a factor of t more randomness than before. We show:

Theorem 7: Let $t < n/3$. Then, any function $f : \{0,1\}^n \to \{0,1\}^n$ that has a circuit of size m, can be t-securely computed by a $O(t^3 \log n + (m/n)t^6 \log t)$-random protocol.

The protocol for active adversaries is identical to the one for passive adversaries, with the exceptions that the size of teams is increased to $s = 3t+1$, and that the various components of the [6] protocol are replaced by their Byzantine counterparts, for securely computing the functions VSS, ADD, ACT-MULT, RECONS described in the previous section. In addition, the trusted dealer now has to supply the parties with enough random values to support the new protocols. Inspecting these schemes (see previous section) it turns out that each invocation of VSS requires from the dealer $O(t^2)$ values in $GF[p]$, and evaluating a gate requires $O(t^2)$ values in $GF[p]$ for each party in the team. Furthermore, the adversary's view of the computation now depends on at most $\beta = O((m/n) \cdot t^5)$ elements in $GF[p]$. Thus, the dealer can still use the same method used for passive adversaries, with the appropriate value of β. Emulating the trusted dealer is done in the same way as in the basic protocol.

Proof of Theorem 7 [Trusted dealer case]. The proof is very similar to the proof of Theorem 4. Let $t < n/3$, and let f be the computed function. Fix an arithmetic circuit for f and a prime p. Let π be the (full-fledged) protocol described above with respect to that circuit. As explained, protocol π (in its trusted dealer version) is $O(t \log n + (m/n)t^5 \log t)$-random. We show that protocol π satisfies the conditions of Definition 1 via two claims, similar to Claims 1 and 2. Let π_R be identical to protocol π with the exception that the trusted dealer deals totally random values to the parties.

Claim 3: Protocol π_R t-securely computes f. That is, for any t-limited (active) real-life adversary $\mathcal{A}$, there exists an ideal-model adversary $\mathcal{S}$ such that for all inputs $\vec{x}$ and all auxiliary inputs $\vec{z}$

$$\textsc{ideal}_{f,\mathcal{S}}(\vec{x},\vec{z}) \stackrel{\rm d}{=} \textsc{exec}_{\pi_R,\mathcal{A}}(\vec{x},\vec{z}).$$

Claim 4: For any input $\vec{x}$ and auxiliary input $\vec{z}$, and for any (active) real-life adversary $\mathcal{A}$, the global output of the parties in π and the global output of the parties in π_R are identically distributed. That is, $\textsc{exec}_{\pi_R,\mathcal{A}}(\vec{x},\vec{z}) \stackrel{\rm d}{=} \textsc{exec}_{\pi,\mathcal{A}}(\vec{x},\vec{z})$.

Note that, unlike Claim 2, in Claim 4 both random variables are a result of interaction with an *active* adversary. Yet, the proof of Claim 4 is very similar to the proof of Claim 2, and is therefore omitted. Claim 3 is proven below. This completes the proof of Theorem 7. □

Proof of Claim 3. The proof is very similar to the proof of Claim 1. Let $\hat{\pi}$ denote the high level protocol that corresponds to protocol π_R in the semi-ideal model with ideal evaluation access to functions $\textsc{vss}_{s,i}$, $\textsc{add}_s$, $\textsc{act-mult}_s$ and $\textsc{recons}_{s,W}$. It suffices to show that $\hat{\pi}$ is t-secure in the semi-ideal model. Theorems 1 and 5 imply that protocol π is t-secure in the real-life model.

Given a real-life adversary $\mathcal{A}$, the ideal-model adversary $\mathcal{S}$ proceeds via a simulation of $\mathcal{A}$. Adversary $\mathcal{S}$ starts running $\mathcal{A}$ on its auxiliary input z_0 and random input r_0. Next, $\mathcal{A}$ may corrupt parties, and will expect to see the internal data and the messages received by the corrupted parties. $\mathcal{S}$ proceeds as follows. (As in the case of passive adversaries, $\mathcal{S}$ has to simulate the internal data of parties upon corruption and the messages sent from uncorrupted parties to corrupted parties.)

- (This part is identical to the proof of Claim 1.) Whenever the real-life adversary $\mathcal{A}$ issues a "corrupt P_i" command, the ideal-model adversary $\mathcal{S}$ also corrupts P_i (in the ideal model) and receives P_i's input, x_i, and auxiliary input z_i. Then, $\mathcal{S}$ hands $\mathcal{A}$ the values x_i, z_i, and the following completion of P_i's internal data. P_i's internal data consists of P_i's local input and output values from the already evaluated ideal calls. $\mathcal{S}$ sets these values to be consistent with the data already known to the adversary; whenever a value is not determined it is chosen at random from its domain. That is: The input to the evaluation of $\textsc{sec-shar}_{n,i}$ is set to a random polynomial of degree t with free coefficient x_i. The outputs of all other evaluations of $\textsc{sec-shar}$ are set to random numbers in $GF[p]$. For each gate that P_i's team has already evaluated, the simulator sets the shares that P_i received from uncorrupted parties to be random numbers in $GF[p]$. If the gate is an addition gate then P_i's output value is determined by its inputs. If the gate is a multiplication gate then P_i's output value is set to a random number in $GF[p]$. We show how to simulate the outputs of $\textsc{recons}$ below.

- For each party $P_{T,i}$, the ideal-model adversary $\mathcal{S}$ simulates an interaction of team T with the trusted party for computing $\text{SEC-SHAR}_{s,i}$. That is, if $P_{T,i}$ is corrupted then $\mathcal{S}$ receives the polynomial F handed by $\mathcal{A}$ to the trusted party, and hands $\mathcal{A}$ the value (either $F(\cdot)$ or ϵ, depending on the degree of F) for each corrupted party $P_{T,j}$. If $P_{T,i}$ is not corrupted then, for each corrupted party in team T, the ideal-model adversary $\mathcal{S}$ hands $\mathcal{A}$ a random number in $GF[p]$ as the value given by the trusted party. In addition, if the dealer, $P_{T,i}$, is corrupted then $\mathcal{S}$ hands to its trusted party an input value (for the main function f), computed as follows: If $F(\cdot)$ is a polynomial of degree t then the input value of $P_{T,i}$ is set to $F(0)$. Otherwise, the input of $P_{T,i}$ is set to a default value, say 0.
- (This part is identical to the proof of Claim 1.) For each gate g in the circuit, $\mathcal{S}$ simulates the "baton hand-off" step of the shares of the input lines to the gate. That is, let T be the team that computes gate g, and let T_1, T_2 be the teams that hold the values of the input lines to the gate. Then, for each i, if $P_{T,i}$ is corrupted and $P_{T_1,i}$ (resp., $P_{T_2,i}$) is not corrupted, then $\mathcal{S}$ hands $\mathcal{A}$
- Once the "baton hand-off" step of a gate g is simulated, $\mathcal{S}$ simulates an interaction of team T with the trusted party for computing the function that corresponds to gate g (i.e., either ADD_s or ACT-MULT_s). If the gate g is an addition gate then $\mathcal{S}$ hands $\mathcal{A}$ the sum of the two input values given by each corrupted party in team T to the trusted party. If the gate is a multiplication gate then $\mathcal{S}$ hands $\mathcal{A}$ a value v_i determined as follows. If the value c_i that $P_{T,i}$ handed to its trusted party is different than ϵ then $v_i = c_i$. Otherwise ($c_i = \epsilon$), v_i is set to a random number in $GF[p]$. [10]
- When the simulation of a gate leading to an output wire of the circuit is complete, $\mathcal{S}$ simulates an interaction with the trusted party for computing $\text{RECONS}_{s,W}$, where W is the set of parties that are to learn the value of this wire. If no corrupted party is in W then $\mathcal{S}$ need do nothing. Otherwise, $\mathcal{S}$ invokes its own trusted party for the output value of the main function, f. Let v be the output value that corresponds to this output wire, let T be the team that holds the value of this wire, and let a_i be the share that each corrupted party $P_{T,i}$ in T hands the trusted party for $\text{RECONS}_{s,W}$. Then, $\mathcal{S}$ chooses a polynomial B as follows. Say that a corrupted party $P_{T,i}$ is *conforming* if the value a_i that $P_{T,i}$ hands to the trusted party for $\text{RECONS}_{s,W}$ equals $P_{T,i}$'s output of the gate leading to the output wire. (Note that $\mathcal{S}$ can verify whether a party is conforming.) Then, B is chosen uniformly out of all degree t polynomials such that $B(0) = v$ and $B(i) = a_i$ for each *conforming* corrupted party $P_{T,i}$. Next, $\mathcal{S}$ hands each corrupted $P_{T,i}$ the vector $(B(1), \ldots, B(s))$. (Note that this can always be done since $\mathcal{A}$ corrupts at most t parties.)

Once the simulation of $\mathcal{A}$ is complete, the ideal model adversary $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

The proof that for all inputs $\vec{x}$ and auxiliary inputs $\vec{z}$ we have

$$\text{IDEAL}_{f,\mathcal{S}}(\vec{x}, \vec{z}) \stackrel{\text{d}}{=} \text{EXEC}^{\text{VSS,ADD,ACT-MULT,RECONS}}_{\hat{\pi},\mathcal{A}}(\vec{x}, \vec{z})$$

is identical to the corresponding part of the proof of Claim 1, and is omitted. □

Finally, to eliminate the use of a trusted dealer, we use the same idea as in Section 4.3. The difference here is that if the adversary corrupts one of the $t+1$ dealers then it may choose the M-tuple produced by this dealer in an arbitrary way. However, since at least one of the $t+1$ dealers is not corrupted then the sum still has the required independence properties. The security proof of RAND remains as before.

Acknowledgments We thank Oded Goldreich for his insightful comments which greatly improved the presentation of our paper, as well as for suggesting us a way to get rid of some more unnecessary randomness

[10] The introduction of active adversaries raises an additional apparent difficulty. When an uncorrupted party $P_{T_i,j}$ in team T_i receives a share of a value a from a *corrupted* party $P_{T_u,j}$ in team T_u, it may well be the case that $P_{T_i,j}$ will receive a bad share (or perhaps no share at all). If too many uncorrupted parties in T_i will start off the computation with wrong shares then the evaluation will be incorrect. This difficulty is answered as follows. Since there are at most t corrupted parties altogether, and each corrupted party can give a bad share to at most one party in T_i, it follows that at most t parties in T_i are either corrupted or start off with an erroneous share.

by using [50, 40].

References

[1] N. Alon, O. Goldreich, J. Hastad, and R. Peralta, "Simple Constructions of Almost k-wise Independent Random Variables", FOCS 90 and *Random Structures & Algorithms*, Vol. 4, 1993.

[2] J. Bar-Ilan, and D. Beaver, "Non-Cryptographic Fault-Tolerant Computing in a Constant Number of Rounds", Proc. of 8th PODC, pp. 201–209, 1989.

[3] D. Beaver, "Perfect Privacy for Two-Party Protocols", TR-11-89, Harvard University, 1989.

[4] M. Bellare, O. Goldreich, and S. Goldwasser, "Randomness in Interactive Proofs", FOCS, 1990, pp. 563–571.

[5] M. Ben-Or, R. Canetti and O. Goldreich, "Asynchronous Secure Computations", *25th STOC,* 1993, pp. 52-61.

[6] M. Ben-or, S. Goldwasser, and A. Wigderson, "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation", STOC, 1988, pp. 1–10.

[7] C. Blundo, A. De-Santis, G. Persiano, and U. Vaccaro, "On the Number of Random Bits in Totally Private Computations", ICALP, LNCS 944, 1995, pp. 171–182.

[8] C. Blundo, A. De-Santis, and U. Vaccaro, "Randomness in Distribution Protocols", ICALP, LNCS 820, 1994, pp. 568–579.

[9] C. Blundo, A. Giorgio Gaggia, and D. R. Stinson, "On the Dealer's Randomness Required in Secret Sharing Schemes", EuroCrypt94, LNCS 950, 1995, pp. 35–46.

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

[11] R. Canetti, "Modular Composition of Multi-party Cryptographic Protocols", Available at the Theory of Cryptography Library, `http://........yr.s.l..tore.rests.mit.edu/~tcryptol`, 1998.

manuscript (available from the author), 1998.

[12] R. Canetti, "Studies in Secure Multi-Party Computation and Applications", Ph.D. Thesis, Department of Computer Science and Applied Math, Weizmann Institute of Science, Rehovot, Israel, June 1995.

[13] R. Canetti, U. Feige, O. Goldreich, and M. Naor, "Adaptively Secure Multi-Party Computation", Proc. of 28th STOC, 1996, pp. 639–648.

[14] R. Canetti, and O. Goldreich, "Bounds on Tradeoffs between Randomness and Communication Complexity", FOCS 90 and *Computational Complexity* Vol. 3, 1993, 141-167.

[15] S. Chari, R. Rogatgi, A. Srinivasan "Randomness-Optimal Unique Element Isolation, with Application to Perfect Matching and Related Problems, STOC 1993, pp. 458-467.

[16] D. Chaum, C. Crepeau, and I. Damgard, "Multiparty Unconditionally Secure Protocols", STOC, pp. 11–19, 1988.

[17] B. Chor, and C. Dwork, “Randomization in Byzantine Agreement”, *Advances in Computing Research*, volume 5, 443-497, 1989.

[18] B. Chor, and O. Goldreich, “Unbiased Bits from Sources of Weak Randomness and Probabilistic Communication Complexity”, FOCS 85 and SICOMP Vol. 17, 1988, 230–261.

[19] B. Chor, S. Goldwasser, S. Micali and B. Awerbuch, “Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults”, FOCS, 1985, pp. 383-395.

[20] B. Chor, and E. Kushilevitz, “A Zero-One Law for Boolean Privacy”, STOC 89 and SIDMA, Vol. 4, 36–47, 1991.

[21] B. Chor, and E. Kushilevitz, “A Communication-Privacy Tradeoff for Modular Addition”, *Information Processing Letters*, Vol. 45, 1993, pp. 205–210.

[22] U. Feige, J. Kilian, and M. Naor, “A Minimal Model for Secure Computation”, STOC, pp. 554-563, 1994.

[23] P. Feldman, and S. Micali, “An Optimal Algorithm For Synchronous Byzantine Agreement”, STOC, 1988 and SIAM J. on Computing, Vol. 26, No. 4, 1997, pp. 873–933.

[24] M. Fischer and N. Lynch. “A Lower Bound for the Time to Assure Interactive Consistency”, *IPL*, 14(4), pp. 183–186, 1982.

[25] M. J. Fischer, N. A. Lynch, and M. S. Paterson. “Impossibility of Distributed Consensus with One Faulty Process”, *Journal of the ACM*, 32(2), pp. 374–382, 1985.

[26] M. Franklin, and M. Yung, “Communication Complexity of Secure Computation”, STOC, pp. 699–710, 1992.

[27] R. Gennaro, T. Rabin, and M. Rabin, “VSS and Secure Multiparty Computations Made Simple”, manuscript.

[28] O. Goldreich, S. Micali, and A. Wigderson, ”How to Play Any Mental Game”, Proc. of 19th STOC, 1987, pp. 218-229.

[29] S. Goldwasser, and L. Levin, “Fair Computation of General Functions in Presence of Immoral Majority”, *CRYPTO,* 1990.

[30] J. Hastad, “Pseudo-Random Generators under Uniform Assumptions”, *STOC 90*.

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

[32] R. Impagliazzo, and D. Zuckerman, “How to Recycle Random Bits”, Proc. of 30th FOCS, 1989, pp. 248–253.

[33] D. Karger, and D. Koller, “(De)randomized Construction of Small Sample Spaces in *NC*”, FOCS, 1994, pp. 252-263.

[34] D. Karger, and R. Motwani, “Derandomization through Approximation: An *NC* Algorithm for Minimum Cuts”, Proc. of 26th STOC, 1994, pp. 497–506.

[35] H. Karloff, and Y. Mansour, "On Construction of k-wise Independent Random Variables", STOC, 1994, pp. 564–573.

[36] D. E. Knuth, and A. C. Yao, "The Complexity of Non-Uniform Random Number Generation", Algorithms and Complexity, pp. 357-428, ed. J. Traub, 1976.

[37] D. Koller, and N. Megiddo, "Constructing Small Sample Spaces Satisfying Given Constraints", STOC 93 and *SIAM J. Disc. Math.* Vol. 7, 1994.

[38] D. Krizanc, D. Peleg, and E. Upfal, "A Time-Randomness Tradeoff for Oblivious Routing", STOC, 1988, pp. 93–102.

[39] E. Kushilevitz, "Privacy and Communication Complexity", FOCS 89, and SIAM Jour. on Disc. Math., Vol. 5, No. 2, 1992, pp. 273–284.

[40] E. Kushilevitz, and Y. Mansour, "Randomness in Private Computations", PODC 1996, and SIAM Jour. on Disc. Math., Vol. 10, No. 4, 1997, pp. 647–661.

[41] E. Kushilevitz, S. Micali, and R. Ostrovsky, "Reducibility and Completeness in Multi-Party Private Computations", Proc. of 35th FOCS, pp. 478-489, 1994.

[42] E. Kushilevitz, R. Ostrovsky, and A. Rosén, "Characterizing Linear Size Circuits in Terms of Privacy", STOC, pp. 541–550, 1996.

[43] E. Kushilevitz, R. Ostrovsky, and A. Rosén, "Amortizing Randomness in Private Multiparty Computations", Proc. of 17th PODC, 1998, to appear.

[44] E. Kushilevitz, and A. Rosén, "A Randomness-Rounds Tradeoff in Private Computation", CRYPTO, LNCS 839, pp. 397-410, 1994.

[45] S. Micali, and P. Rogaway, "Secure Computation", manuscript, 1992. Preliminary version in *CRYPTO 91.*

[46] J. Naor, and M. Naor, "Small-Bias Probability Spaces: Efficient Constructions and Applications", STOC 90, and *SIAM J. on Computing,* Vol 22, No. 4, 1993, pp. 838–856.

[47] N. Nisan, "Pseudorandom Generator for Space Bounded Computation", Proc. of 22nd STOC, 1990, pp. 204–212.

[48] P. Raghavan, and M. Snir, "Memory vs. Randomization in On-Line Algorithms", ICALP, 1989, pp. 687–703.

[49] A. Shamir. "How to Share a Secret", *Communications of the ACM*, 22:612–613, 1979.

[50] L. J. Schulman, "Sample Spaces Uniform on Neighborhoods", Proc. of 24th STOC, 1992, pp. 17–25.

[51] J. H. van Lint, and R. M. Wilson, "A Course in Combinatorics", Cambridge University Press, 1992.

[52] A. C. Yao, "Theory and Applications of Trapdoor Functions", Proc. of 23rd FOCS, 1982, pp. 80–91.

[53] A. C. Yao, "How to Generate and Exchange Secrets", Proc. of 27th FOCS, 1986, pp. 162-167.

[54] D. Zuckerman, "Simulating BPP Using a General Weak Random Source", Proc. of 32nd FOCS, 1991, pp. 79–89.

A An Extension of [50, 40]

In this appendix we describe a straightforward extension of results from [50, 40]. (All operations in this section are over $GF[p]$.) The goal is as follows: Given sets $S_1, \ldots, S_t \subseteq \{1, \ldots, n\}$ we wish to construct a multi-set[11] $\mathcal{D}$ of n-tuples over $GF[p]$ such that if we look at the projection of $\mathcal{D}$ on the coordinates in any of the sets S_j then we get a uniform distribution over all the $p^{|S_j|}$ possible tuples. We start with the following definition: given an $n \times \ell$ matrix M we define the following multi-set of size p^ℓ:

$$space(M) \quad = \quad \{M \cdot v | v \in (GF[p])^\ell\} \quad \subseteq \quad (GF[p])^n.$$

For such a matrix M, denote its rows by $M_1, \ldots, M_n$.

Claim 5: Let $\{w_i\}_{i \in S_j}$ be arbitrary elements of $GF[p]$. If

$$\sum_{i \in S_j} w_i M_i \neq \vec{0}$$

then, when we choose a vector y from the probability distribution defined by $space(M)$, the sum $\sum_{i \in S_j} w_i \cdot y_i$ is uniformly distributed over $GF[p]$.

Proof: Since $\sum_{i \in S_j} w_i M_i \neq \vec{0}$ then for some d we have $\sum_{i \in S_j} w_i M_{i,d} \neq 0$. Partition the vectors in $(GF[p])^\ell$ into sets of size p of vectors that agree in all coordinates except for the d-th coordinate. Then, for each such set the sum $\sum_{i \in S_j} w_i (M \cdot v)_i$ takes all p values each of them exactly once.[12] This implies that, for the corresponding y's, the sum $\sum_{i \in S_j} w_i \cdot y_i$ takes all p values each of them once and so the claim follows. $\square$

Claim 6: If for every choice of $\{w_i\}_{i \in S_j}$, which are not all 0's, we have $\sum_{i \in S_j} w_i M_i \neq \vec{0}$ then the projection of $space(M)$ on S_j is uniformly distributed.

Proof: By Claim 5, we get that for every such choice of $\{w_i\}_{i \in S_j}$, the sum $\sum_{i \in S_j} w_i \cdot y_i$, for $y \in space(M)$, is uniformly distributed in $GF[p]$. Since the Fourier basis for $GF[p]$ consists of all the functions of the form $r^{\sum w_i \cdot y_i}$, where r is the root of unity of order p, then the behavior of a distribution with respect to these functions determines the distribution. Therefore, the uniform distribution is the only distribution which is uniform with respect to any such function. $\square$

[11] by *"multi-set"* we mean that an element may appear more than once in $\mathcal{D}$. We interpret a multiset as a probability distribution in the natural way: each element is drawn with probability proportional to the number of times it appears in the multi-set.

[12] To see this we write $\sum_{i \in S_j} w_i M_i = \vec{z} \neq \vec{0}$ which implies that for some d we have $z_d \neq 0$. Also for $y \in space(M)$ we have $y = M \cdot v$ so $y_i = \sum_{k=1}^{\ell} M_{i,k} \cdot v_k$. Therefore

$$\sum_{i \in S_j} w_i \cdot y_i = \sum_{i \in S_j} w_i \cdot \sum_{k=1}^{\ell} M_{i,k} \cdot v_k = \sum_{i \in S_j} \sum_{k \neq d} w_i M_{i,k} \cdot v_k + \sum_{i \in S_j} w_i M_{i,d} \cdot v_d.$$

If we fix all the elements of v except for the d-th one then we get that the first summand is always the same while the second summand equals $z_d \cdot v_d$. Since $z_d \neq 0$ then over all values of v_d we get all p values in $GF[p]$, as needed.

Algorithm: Let d be a bound on the size of the sets $S_1, \ldots, S_t$. We describe an algorithm that runs in time $poly(n, t, p^d)$ and generates a matrix M such that $space(M)$ satisfies the property required in Claim 6 with respect to each of the t sets.[13] If the size of any S_j is $\omega(\log n)$ then this is super-polynomial. However, even in this case the size of M is much smaller and so *sampling* in the space remains efficient. We need to make sure that, for every set S_j, the corresponding rows of M will be linearly independent. We will construct M in a row-by-row manner. While choosing the row M_i, we will make sure that it satisfies the appropriate linear independence constraints. That is, for every set S_j such that $i \in S_j$, we will have to pick a row M_i which is independent of the rows in $T_j = S_j \cap \{1, 2, \ldots, i-1\}$. Fix the value of ℓ to be $\frac{\log t}{\log p} + d + 1$. This implies that the size of the space is $p^\ell \approx t \cdot p^d$. Therefore, for each T_j we can compute the $p^{|T_j|} \leq p^d$ vectors in the linear space spanned by T_j. We do this for each of the (at most t) sets that contain i and so we compute (at most) $t \cdot p^d$ vectors which cannot serve as M_i. Since we have p^ℓ possible vectors then by the choice of ℓ there exists a vector that can serve as M_i.

[13] One can describe a *randomized* algorithm to do so, but since this algorithm will be run by each of the dealers and the whole issue is saving randomness we will restrict ourselves to deterministic algorithms.