J. Cryptology (2000) 13: 107–142
DOI: 10.1007/s001459910005

Journal of
CRYPTOLOGY

© 2000 International Association for Cryptologic Research

Randomness versus Fault-Tolerance*

Ran Canetti
IBM T. J. Watson Research Center,
30 Saw Mill River Road,
Hawthorne, NY 10532, U.S.A.
canetti@watson.ibm.com

Eyal Kushilevitz
Department of Computer Science, Technion,
Haifa, Israel
http://www.cs.technion.ac.il/∼eyalk

Rafail Ostrovsky
Bell Communications Research, MCC-1C365B,
Morristown, NJ 07960-6438, U.S.A.
rafail@bellcore.com

Adi Rosén
Department of Computer Science, University of Toronto,
Toronto, Ontario, Canada
adiro@cs.toronto.edu

Communicated by Oded Goldreich

Received 30 April 1998 and revised 3 January 1999 and 1 July 1999

Abstract. We investigate the relations between two major properties 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 nontrivial 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

* A preliminary version of this paper appeared in the *Proceedings of the* 16*th PODC*, 1997, pp. 35–45. Eyal Kushilevitz was supported by the MANLAM Fund. Part of this work was done while Eyal Kushilevitz and Adi Rosén were visiting Bellcore.

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 $O(poly(t) \cdot \log n + poly(t) \cdot (C(f)/n))$ 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.

Key words. Secure multiparty protocols, Randomness, Limited independence, Composition of protocols.

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 [Y2] and [GMW]) 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) [BGW], [CCD], or it can be limited to efficient (probabilistic polynomial time) computations [Y2], [GMW]. 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. To simplify the presentation, we concentrate in this work on the *static* case although the results (and techniques) carry onto the adaptive case as well.

We mention some known results: In [Y2] and [GMW] 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 [BGW] and [CCD] 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. A considerable amount of work has been done in this area (e.g., [BB], [B1], [BDPV], [BDV], [CFGN], [CK1], [FKN], [FY], [K], [KM4], [KMO], [KOR1], [KR], and [RB]); in what follows 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 a linear number of faults requires a linear number of rounds deterministically [FL] and a constant number of rounds if randomization is allowed [FM]; reaching a consensus in an asynchronous distributed system with faults is impossible with deterministic protocols [FLP], but is possible with the use of randomized protocols (see [CD]). Various techniques to minimize the amount of randomness needed were extensively studied in computer science (e.g., [AGHP], [BGG], [BM], [CG1], [IZ], [KK], [KM1], [KM2], [KM3], [KM4], [KY], [N], [NN], [S2], [Y1], and [Z]) and tradeoffs between randomness and other resources were found (e.g., [BDPV], [BGS], [BDV], [CG2], [CK2], [CRS], [KM4], [KOR1], [KPU],[KR], and [RS]).

Security versus randomness. It is not hard to show that, except for degenerate cases, *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 number of multiplication gates in 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 [KM4]. 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 [BDPV].
2. For the special case of 1-privacy, any linear-size circuit can be computed 1-privately with a constant number of random bits [KOR1]. More generally, every circuit of m boolean gates can be computed 1-privately with $O(m/n)$ random bits [KOR1].

Our results. We generalize both of the above results. That is, we show that for both passive and active adversaries (even adaptive ones), and for *any* value of t for which secure computation is possible, any circuit of m boolean gates can be securely evaluated using only $O(poly(t) \cdot (\log n + m/n))$ random bits overall. While these results do

[1] When the adversary is limited to probabilistic polynomial time and intractability assumptions are used, as in [Y2] and [GMW], then by the results of [BM], [H], and [ILL] we may as well assume the existence of a pseudorandom generator. In this case, if a party needs "many" random bits, it can always choose only a "small" seed of truly random bits, and expand the seed into a "long" sequence of pseudorandom bits and use them. Therefore, in the case of computationally bounded adversaries the quantification of the "amount of randomness needed" is not meaningful (and, in particular, the amount of randomness needed inherently depends on a security parameter).

not substantially improve on [BGW] and [CCD] for $t = \Theta(n)$, they constitute a big improvement for smaller values of t. In particular, for $t = \mathit{polylog}(n)$, circuits with a quasi-linear (i.e., $m = O(n \cdot \mathit{polylog}(n))$) number of gates can be securely evaluated using only $\mathit{polylog}(n)$ random bits. For $t = 1$, we are only $O(\log n)$ away from the specialized (to passive adversaries only) result of [KOR1].

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 "partial 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 "partial decentralization" solution is considerably inferior to a fully distributed computation: while our solution needs only $O(\mathit{poly}(t) \cdot (\log n + m/n))$ randomness, the above "partial decentralization" solution (according to presently known methods) requires $O(\mathit{poly}(t) \cdot m)$ 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 [BGW], and combine it with ideas from [KOR1] 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 [BGW]. (In particular, we use the [BGW] modules for secret sharing and evaluating individual gates as building blocks.) However, as in [KOR1], 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 the technique of [KOR1] 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 derandomization techniques.

Interestingly, we show that not only can we 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 [KM4] 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 protocol composition technique. To show the security of our protocols, we use general definitions of secure multiparty protocols. In particular, we use the formalization of [C2], which allows modular composition of secure protocols. (This formalization is based on the approach in [B2] and [B3].) That is, in order to avoid reproving the security of the [BGW] construction from scratch, we separately prove the security of the overall design of our protocol, assuming that the [BGW] modules for secret-sharing and for evaluating individual gates are secure. We then conclude, using the [C2] composition theorem, that the composition of our "overall design" with the [BGW] modules is secure. (For self-containment we also sketch a proof of security of our protocol for *passive* adversaries, without relying on [C2].) We remark that a formal proof of security for [BGW] was never published. (It can be inferred, say, from the security proof of [BCG] as it appears in [C1].) The modular proof technique used here can be applied also to proving the security of the [BGW] protocol itself and it has the advantage that it extends to the adaptive case as well.

Organization. In Section 2 we provide some necessary definitions, including those of privacy and randomness. In Section 3 we review the solution of [BGW] 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 [BGW] 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 Appendix 6.2 we describe a simple extension of the results of [S2] and [KM4] for sample spaces with limited independence; we use this extension in our constructions. In Appendix 6.2 we sketch a proof of security of our protocol for *passive* adversaries, without relying on [C2].

2. Preliminaries

In Section 2.1 we review the notion of secure protocols, using the formalization of [C2]. In Section 2.2 we review the notion of *modular composition* of protocols, introduced in [MR], and restate the composition theorem from [C2]. Modular composition plays a central role in the security proofs of our protocols. In Section 2.3 we define other notions used within the paper. With the exception of Section 2.3, the material in this section is a summary of the corresponding sections in [C2], and is included here for the sake of self-containment.[2]

Multiparty functions. The functions to be evaluated by the parties are formalized as follows. An n-party function (for some $n \in \mathbf{N}$) is a probabilistic function $f\colon (D)^n \times \{0,1\}^* \to (D)^n$, for some finite domain D, where the last input is taken to be the random input.

2.1. *Secure Protocols*

We specify 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., [GL], [MR], [B3], and [C2]). In this

[2] One difference from the formalization of [C2] is that there the complexity measures, and the security requirement, are stated in terms of a *security parameter* that tends to infinity. Here we deal with a simpler case where the inputs are taken from a finite set, and the security is perfect (i.e., no computational restrictions are made on the adversary and no "negligible probabilities of error" are allowed). Consequently, the security parameter is not necessary. In fact, the definitions here can be regarded as a statement of the definitions of [C2] for a specific value of the security parameter.

work we use the definition of [C2] which we sketch below. We concentrate on the "secure channels" setting of [BGW] and [CCD], where the adversary is computationally unbounded but has no access to the communication between nonfaulty parties. Also, for simplicity of exposition we concentrate on the case of static (nonadaptive) adversaries. Nevertheless, all the protocols presented in this paper maintain their security even in the presence of adaptive adversaries. The definitions for the passive and active cases are very similar; we develop them together, noting the differences as we go.

In a nutshell, secure protocols are protocols that "emulate" an ideal model where all parties privately hand their inputs to a centralized trusted party who computes the results, hands them back to the parties, and vanishes. The definition is described in three stages: First the "real-life" model of computation is formalized; next the ideal model is formalized; finally the notion of "emulation" and the definition are presented.

The real-life model. An n-party protocol π is a collection of n interactive, probabilistic algorithms. Formally, each algorithm is an Interactive Turing machine, as defined in [GMR]. We use the term party P_i to refer to the ith algorithm. Each party P_i starts with input $x_i \in D$, and random input $r_i \in \{0, 1\}^*$. Informally, we envision each two parties as connected via a private communication channel. A more complete description of the communication among parties is presented below. A t-limited real-life adversary, $\mathcal{A}$, is another interactive (computationally unbounded) Turing machine describing the behavior of the corrupted parties. Adversary $\mathcal{A}$ starts off with input that contains the identities of the corrupted parties (some subset $C \subseteq \{1, \ldots, n\}$), together with their inputs and random inputs. In addition, $\mathcal{A}$ receives auxiliary input z. (The auxiliary input is a standard tool that allows proving the composition theorem. Intuitively, the auxiliary input captures information gathered by the adversary from other interactions occurring before the current interaction. Auxiliary inputs were first introduced in [GO], in the context of Zero-Knowledge proofs; for discussion see [GO] and [G].)

The computation proceeds in rounds, where each round proceeds as follows. First the uncorrupted parties generate their messages of this round, as described in the protocol. (That is, these messages appear on the outgoing communication tapes of the uncorrupted parties.) The messages addressed to the corrupted parties become known to the adversary (i.e., they appear on the adversary's incoming communication tape). Next the adversary generates the messages to be sent by the corrupted parties in this round. If the adversary is passive, then these messages are determined by the protocol. An active adversary determines the messages sent by the corrupted parties in an arbitrary way. Finally each uncorrupted party receives all the messages addressed to it in this round (i.e., the messages addressed to P_i appear on P_i's incoming communication tape).

At the end of the computation all parties locally generate their outputs. The uncorrupted parties output whatever is specified in the protocol. The corrupted parties output a special symbol, $\perp$, specifying that they are corrupted. In addition, the adversary outputs some arbitrary function of its view of the computation. The adversary view consists of its auxiliary input and random input, followed by the corrupted parties' inputs, random inputs, and all the messages sent and received by the corrupted parties during the computation. Without loss of generality, we can imagine that the real-life adversary's output consists of its entire view.

Let $\text{ADVR}_{\pi,\mathcal{A}}(\vec{x}, z, \vec{r})$ denote the output of real-life adversary $\mathcal{A}$ with auxiliary input

z and when interacting with parties running protocol π on input $\vec{x} = x_1, \ldots, x_n$ and random input $\vec{r} = r_{\mathcal{A}}, r_1, \ldots, r_n$ as described above ($r_{\mathcal{A}}$ for $\mathcal{A}$, x_i and r_i for party P_i). Let $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, z, \vec{r})_i$ denote the output of party P_i from this execution. Recall that if P_i is uncorrupted, then this is the output specified by the protocol; if P_i is corrupted, then $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, z, \vec{r})_i = \perp$. Let

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

Let $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, z)$ denote the probability distribution of $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, z, \vec{r})$ where $\vec{r}$ is uniformly chosen.

The ideal process. The ideal process is parameterized by the function to be evaluated. This is an n-party function $f\colon (D)^n \times \{0, 1\}^* \to (D)^n$, as defined above. Each party P_i has input $x_i \in D$; no random input is needed for the parties in the ideal process (if f is a probabilistic function, then the needed randomness will be chosen by the trusted party). Recall that the parties wish to compute $f(\vec{x}, r_f)_1, \ldots, f(\vec{x}, r_f)_n$, where r_f is an appropriately long random string, and P_i learns $f(\vec{x}, r_f)_i$ (where $f(\vec{x}, r_f)_i$ denotes the ith component of $f(\vec{x}, r_f)$). An ideal-process-adversary $\mathcal{S}$ is an interactive (computationally unbounded) Turing machine describing the behavior of the corrupted parties. Adversary $\mathcal{S}$ starts off with the identities and inputs of the corrupted parties (parties P_i for $i \in C$), random input, and auxiliary input. In addition, there is an (incorruptible) trusted party, T. The ideal process proceeds as follows.

INPUT SUBSTITUTION: The ideal-process-adversary $\mathcal{S}$ sees the inputs of the corrupted parties. If $\mathcal{S}$ is active, then it may also alter these inputs. Let $\vec{b}$ be the $|C|$-vector of the altered inputs of the corrupted parties, and let $\vec{y}$ be the n-vector constructed from the input $\vec{x}$ by substituting the entries of the corrupted parties by the corresponding entries in $\vec{b}$. If $\mathcal{S}$ is passive, then no substitution is made and $\vec{y} = \vec{x}$.

COMPUTATION: Each party P_i hands its (possibly modified) input value, y_i, to the trusted party T. Next, T chooses a value r_f randomly from $\mathcal{R}_f$, and hands each P_i the value $f(\vec{y}, r_f)_i$.

OUTPUT: Each uncorrupted party P_i outputs $f(\vec{y}, r_f)_i$, and the corrupted parties output $\perp$. In addition, the adversary outputs some arbitrary function of the information gathered during the computation in the ideal process. This information consists of the adversary's random input, the corrupted parties' inputs, and the resulting function values $\{f(\vec{y}, r_f)_i\colon\ P_i \text{ is corrupted}\}$.

Let $\text{ADVR}_{f,\mathcal{S}}(\vec{x}, z, \vec{r})$, where $\vec{r} = (r_f, r)$, denote the output of ideal process adversary $\mathcal{S}$ on random input r and auxiliary input z, when interacting with parties having input $\vec{x} = x_1, \ldots, x_n$, and with a trusted party for computing f with random input r_f. Let the $(n + 1)$-vector

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

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

Definition of security. We require that protocol π emulates the ideal process for evaluating f, in the following sense. For any real-life adversary $\mathcal{A}$ there should exist an ideal-process adversary $\mathcal{S}$, such that, for any input vector $\vec{x}$ and any auxiliary input z, the global outputs $\text{IDEAL}_{f,\mathcal{S}}(\vec{x}, z)$ and $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, z)$ are identically distributed. Furthermore, we require that the complexity of the ideal-process adversary $\mathcal{S}$ be comparable with (i.e., polynomial in) the computational complexity of the real-life adversary $\mathcal{A}$. (See [C2] for motivation and discussion of this requirement.)

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

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

where $\overset{\text{d}}{=}$ denotes equality between two distributions. If $\mathcal{A}$ and $\mathcal{S}$ are passive adversaries, then we say that π t-privately evaluates g.

2.2. *Composition of Secure Protocols*

In what follows we use the fact that the security of protocols is preserved under a natural composition operation. For a full exposition and a proof see [C2]. 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 resulting 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 [MR]. 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.

The hybrid model. 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 hybrid model with ideal access to f (or in short the f-hybrid 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 interact with T_f in a way that is similar to the ideal process for evaluating f. That is, the parties hand their f-inputs to T_f (party P_i hands ξ_i), and are handed back their respective outputs (P_i learns

$f(\xi_1, \ldots, \xi_n, r_f)_i$). The values ξ_i that correspond to corrupted parties are decided by the adversary, who also learns the values handed by T_f to the corrupted parties. 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.

Let $\text{EXEC}^{f_1,\ldots,f_m}_{\pi,\mathcal{A}}(\vec{x}, z)$ denote the random variable describing the output of the computation in the $(f_1, \ldots, f_m)$-hybrid model with protocol π, adversary $\mathcal{A}$, inputs $\vec{x}$, and auxiliary input z for the adversary, analogously to the definition of $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, z)$ in Section 2.1. (We stress that here π is a hybrid of a real-life protocol with ideal evaluation calls to T.)

Security in the hybrid model. Protocols for securely computing a function g in the $(f_1, \ldots, f_k)$-hybrid model are defined in the usual way:

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

$$\text{IDEAL}_{g,\mathcal{S}}(\vec{x}, z) \overset{\text{d}}{=} \text{EXEC}^{f_1,\ldots,f_m}_{\pi,\mathcal{A}}(\vec{x}, z). \tag{2}$$

If $\mathcal{A}$ and $\mathcal{S}$ are passive adversaries, then we say that π t-privately evaluates g in the $(f_1, \ldots, f_m)$-hybrid model.

Replacing ideal evaluation with a subroutine. 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 $\pi^{\rho_1,\ldots,\rho_m}$ denote protocol π where each ideal evaluation call to f_i is replaced by an invocation of protocol ρ_i.

Theorem 1 [C2]. *Let $f_1, \ldots, f_m$ and g be n-party functions. Let π be an n-party protocol that t-securely (resp., t-privately) computes g in the $(f_1, \ldots, f_m)$-hybrid model, in a way that no more than one ideal evaluation call is made at each round. Let $\rho_1, \ldots, \rho_m$ be n-party protocols that t-securely (resp., t-privately) compute $f_1, \ldots, f_m$, respectively. Then the protocol $\pi^{\rho_1,\ldots,\rho_m}$ t-securely (resp., t-privately) computes g.*

2.3. *Other Definitions*

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.[3]

Definition 3. A d-random protocol is a protocol such that, for every input assignment $\vec{x}$ and every auxiliary input 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 what follows 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 (although, to measure the complexity of our protocols, m could be taken as the number of multiplication gates only). 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 NOT $a \Rightarrow (1-a)$; a AND $b \Rightarrow a \cdot b$; and a OR $b \Rightarrow 1-((1-a)(1-b))$.

For simplicity of presentation and analysis we concentrate on secure evaluation of *deterministic* functions. Still, as a side-remark we sketch a way for dealing with probabilistic functions. The idea is to "share" each random input to the circuit among the parties in a way that prevents the adversary from influencing the chosen value, and guarantees that the adversary gathers no information on this value on top of the information leaked by the function value. More precisely, let A be a circuit that has $t+1$ input wires, and a single output wire whose value equals the sum of the inputs (mod p). Given a randomized circuit C with random input wires $r_1, \ldots, r_k$, construct a circuit C' that is identical to C except that in C' each r_i is replaces by an A circuit, denoted A_i. Each party P_j with $j \leq t+1$ is assigned to the jth input of each A_i. (This is in addition to the other, regular input wires assigned to P_j.) P_j chooses a random value in $GF[p]$ for each one of the A-inputs assigned to it, and from this point on treats each such input wire as a regular input wire. The parties now proceed to evaluate C'.

[3] 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.

3. An Overview of the Protocol of [BGW] for Passive Adversaries

Our construction for passive adversaries, described in the next section, uses components used in the general construction of [BGW] 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 [BGW]. The construction (and its proof) is presented in a modular way, using the formalism from the previous section. This form of presentation enables us to use the components of [BGW] without reproving their security from scratch.

In the [BGW] protocol the parties first agree on an arithmetic circuit for the function f to be computed. In particular, the parties agree on a prime $p > n$ (all the arithmetic in what follows is done modulo p) and on n distinct elements $\mu_1, \ldots, \mu_n$ in $GF[p]$ (all polynomials in the protocol will be evaluated at these n evaluation points; for example, we can choose $\mu_1 = 1, \ldots, \mu_n = n$). Each party is assigned to some of the input wires. The party's input consists of a value for each of the input wires assigned to it. 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 among the parties the value of each input wire assigned to it. 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 [BGW] protocol consists of a "high-level" protocol for evaluating the circuit; this protocol uses as "subroutines" protocols for secure evaluation of the following n-party functions:

Secret sharing. $\text{SHARE}_n(s, \varepsilon, \ldots, \varepsilon) = F(\mu_1), \ldots, F(\mu_n)$, where $s \in GF[p]$ is the "secret" to be shared, ε denotes the empty input, and $F()$ is a random polynomial of degree t in $GF[p]$ with $F(0) = s$. Let $\text{SHARE}_{n,i}$ denote the function SHARE_n where the dealer (i.e., the party with nonempty input) is P_i. Note that we do not specify how the coefficients of F are chosen; this is regarded as the "intrinsic randomness" of the function SHARE.

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. $\text{MULT}_n(a_1|b_1, \ldots, a_n|b_n) = C(\mu_1), \ldots, C(\mu_n)$, 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 $A(\mu_i) = a_i$ (resp., $B(\mu_i) = b_i$) for all i. (Also here we do not specify how the coefficients of C are chosen; this is 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 $\alpha_i = \varepsilon$ otherwise. In the high-level protocol the parties in W will interpolate a (degree t) polynomial A satisfying $A(\mu_i) = a_i$ for all i, and will output $A(0)$.[4]

[4] An apparently simpler formalization of function RECONS would be to let $\alpha_i = s$ if $i \in W$, where s is the

Theorem 2 [BGW]. *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 protocols for computing $\text{SHARE}_{n,i}$ and $\text{RECONS}_{n,W}$ are just Shamir's secret sharing and reconstruction protocols [S1]. The secret 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 party locally summing its two inputs. Below we sketch Rabin's simplification of the protocol for securely computing MULT_n, as it appears in [GRR]. This protocol requires each participating party to choose $O(t \log p)$ random bits (hence a total of $O(nt \log p)$ random bits in each invocation of the multiplication protocol).

The multiplication step of [*GRR*]. 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 $2t$ (for instance, D cannot be irreducible). We overcome these problems as follows. We show below that there is a linear combination

$$D(0) = \sum_{i=1}^{2t+1} \gamma_i D(\mu_i), \tag{3}$$

where the γ_i's are known coefficients. Once this is established, the parties can proceed as follows: Each party P_i ($1 \le i \le 2t+1$) chooses a random polynomial $\Delta_i(x)$ of degree t whose free coefficient is d_i. It then sends $\Delta_i(\mu_j)$ to P_j. Each party P_j computes $\alpha_j = \sum_{i=1}^{2t+1} \gamma_i \alpha_{i,j}$, where $\alpha_{i,j}$ is the value that P_j receives from P_i. It holds that α_j is P_j's share for the polynomial $\Delta(x) = \sum_{i=1}^{2t+1} \gamma_i \Delta_i(x)$ which is a random, degree t, polynomial whose free coefficient is $\sum_{i=1}^{2t+1} \gamma_i D(\mu_i) = D(0)$.

It remains to show γ_i's that satisfy (3). 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 $\vec{D} = (D(\mu_1), D(\mu_2), \ldots, D(\mu_{2t+1}))$. With this notation we get that $\vec{D} = V \cdot \vec{d}$. Since V is nonsingular (see, e.g., [vLW]), 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 $D(0) = d_0 = \sum_{i=1}^{2t+1} V_{1,i}^{-1} \cdot D(\mu_i)$ (where V^{-1} is a fixed matrix).

For completeness, we state the following theorem:

Theorem 3 [BGW]. *Let $t < n/2$. Then, given an arithmetic circuit for computing an n-party function f, there exists a protocol for t-securely computing f in the hybrid*

free coefficient of the polynomial $A()$ satisfying $A(\mu_i) = a_i$ for all i. However, this formalization imposes an additional (and unnecessary) secrecy requirement, namely, that even the parties in W do not learn the inputs of the other parties *to the reconstruction protocol*. Meeting this additional requirement would require unnecessarily complex protocols.

model with passive adversaries and with ideal access to the functions $\text{SHARE}_{n,i}$, 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-party function f in the presence of passive adversaries. The number of random bits used by these protocols is $O(mnt \log p)$ (where m is the size of the circuit for f).

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* $f\colon \{0,1\}^n \to \{0,1\}^n$ *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 present 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. Formally, we present and analyze the protocol in the hybrid model, with ideal access to a function that takes no input, and generates outputs from a distribution to be determined in what follows. At the onset of the protocol the parties first evaluate this function (denoted RAND_n), and use the local outputs as their random inputs for the rest of the protocol. In Section 4.3.3 we present a simple protocol that securely evaluates RAND_n.

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 wires from their shares. Our approach can be applied to any protocol that follows this outline. For concreteness, however, we concentrate on the construction of [BGW] (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 [S1] and [BGW] 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 $\approx n/s$ 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 procedure of [BGW] 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 (in a shared manner), the corresponding teams provide the specified parties with the information needed to reconstruct these values.

The random input of the parties (needed for sharing their inputs and for the gate evaluations) is provided by the trusted dealer (i.e., by the function RAND_n), in a way that guarantees that the view of each subset of at most t parties depends only on a "small" fraction of the overall random input of the system. Thus, the random inputs dealt to the parties may have only limited independence, which leads to saving in overall randomness.

4.2. *Detailed Description* (*with Trusted Dealer*)

We now state the protocol in detail, in the RAND_n-hybrid model. Let the team size be $s = 2t + 1$ and the number of teams be $k = n/s$ (we assume for convenience that n is divisible by s; see Remark 1 below). Next, partition the n parties into k teams of parties of size s each. Each team will evaluate (at most) $\ell = \lceil m/k \rceil$ gates. We also specify an enumeration of the parties in each team. Denote by $P_{T,j}$ the jth party in team T. Let $p > s$ be a prime, and let $\mu_1, \ldots, \mu_s$ be s evaluation points, as before. All the computations described below are over $GF[p]$. First the parties perform an ideal evaluation call to RAND_n, and use the outcome as their random input for the rest. Next, we describe the "high-level" protocol in the hybrid model with access to ideal evaluation of the functions $\text{SHARE}_{s,i}$, 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 $\text{SHARE}_{s,j}$ with dealer $P_{T,j}$. The dealer's input to $\text{SHARE}_{s,j}$ is its 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).[5] 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.

[5] T, T_x, and T_y need not be disjoint, or even distinct.

- *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 team 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 $\mathrm{RECONS}_{s,W}$ where $W \subseteq [n]$ is the set of parties that are assigned to this wire. Next, each party in W interpolates a (degree t) polynomial A satisfying $A(\mu_i) = a_i$ 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 we use the subroutines of [BGW], sketched in the previous section.

The function RAND_n. We now turn to describing the distribution provided by RAND_n. (Figuratively, this is the distribution provided by the trusted dealer.) The output of RAND_n consists of M random elements in $GF[p]$, denoted $Z_1, \ldots, Z_M$, where each coordinate out of $1, \ldots, M$ is assigned to exactly one party. Each party receives the elements whose coordinates are assigned to it. The elements will have only limited independence; specifically, they will be only β-wise independent. The values M and β, as well as the number of field elements received by each party, are determined below.

We count the number of random elements in $GF[p]$ required by the protocol. In the INPUT SHARING stage the dealer distributes coefficients of n degree-t polynomials (one polynomial to each party). Then the dealer distributes additional s polynomials per each multiplication 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 $D(\cdot)$). Altogether, a set of t parties gets $O(t^2 \ell)$ 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 $O(t^2 \ell s) = O(t^3 \ell) = O(t^4 m/n)$ numbers from the distribution. Hence the dealer generates 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.

We describe two ways for computing the desired distribution. A straightforward method proceeds as follows. For the purpose of this method, we assume that the prime p satisfies $p > M$ (this strengthens the assumption made before that $p > s$). The dealer chooses a random polynomial $R(x)$ in $GF[p]$ of degree β and then generates the M numbers $Z_1 = R(1), Z_2 = R(2), \ldots, Z_M = R(M)$. 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 a simple extension of the results of [S2] and [KM4] (which, for self-containment, appears in Appendix 6.2). The dealer will uniformly sample a space of M-tuples over $GF[p]$, which is constructed to suit the specific $\binom{n}{t}$ subsets (we emphasize that, for the purpose of this method, the requirement that $p > s$ suffices). By [S2] and [KM4], 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.[6] To sample this space, $O(t \log(n/t) + (m/n)t^4 \log p) = O(t \log(n/t) + (m/n)t^4 \log t)$ random bits are needed.

4.3. *Proof of Security*

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. 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 parties use truly random inputs for the protocol, instead of using the output of RAND$_n$. (That is, π_R is a protocol in the real-life model, whereas π is a protocol in the RAND$_n$-hybrid model.)

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 any input vector $\vec{x}$ and any auxiliary input z,*

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

[6] Both [S2] and [KM4] 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*$(\binom{n}{t})$. This is polynomial only for $t = O(1)$ but can be done "off-line" and can be hard-wired into the protocol.

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

Claim 3. *There exists an $O(t^2 \log n + (m/n)t^5 \log t)$-random protocol that t-privately evaluates RAND_n.*

The above claims (to be proven below) imply Theorem 4. □

4.3.1. *Proof of Claim* 1

Let $\hat{\pi}$ denote the high-level protocol that corresponds to protocol π_R in the hybrid model with ideal evaluation access to the functions $\text{SHARE}_{s,i}$, ADD_s, MULT_s, and $\text{RECONS}_{s,W}$. It suffices to show that $\hat{\pi}$ is t-secure in the hybrid model. Theorems 1 and 2 then imply that protocol π_R t-privately evaluates f in the real-life model.

Given a real-life adversary $\mathcal{A}$, the ideal-model adversary $\mathcal{S}$ proceeds via (black-box) simulation of $\mathcal{A}$. That is, given a set C of corrupted parties, inputs $\{x_i \mid P_i \in C\}$, and auxiliary input z, adversary $\mathcal{S}$ proceeds as follows. First, $\mathcal{S}$ provides $\mathcal{A}$ with C, $\{x_i \mid P_i \in C\}$, z. Next, $\mathcal{S}$ generates simulated values sent by the uncorrupted parties, and simulated values given by the trusted parties for the evaluated functions. These values are set to random elements in $GF[p]$. In the reconstruction stage, $\mathcal{S}$ provides $\mathcal{A}$ with random field elements that "interpolate" to the function value. A more complete description of simulator $\mathcal{S}$ appears in Fig. 1.

Analysis of simulator $\mathcal{S}$. Fix some input vector $\vec{x}$ and auxiliary input z. We show that

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

Recall that each one of the random variables $\text{IDEAL}_{f,\mathcal{S}}(\vec{x}, z)$ and $\text{EXEC}^{\text{SHARE},\text{ADD},\text{MULT},\text{RECONS}}_{\hat{\pi},\mathcal{A}}(\vec{x}, z)$ consists of the outputs of the parties plus the adversary output. The analysis consists of two steps:

1. Show that $\mathcal{A}$'s view of a simulated execution is distributed identically to its view of a real execution. (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.)
2. Fix some possible value v for $\mathcal{A}$'s view. Let E_v denote the output values of the uncorrupted parties in a real-life execution of $\hat{\pi}$ in which $\mathcal{A}$ has view v. Let I_v denote the outputs of the uncorrupted parties in an execution of the ideal process with $\mathcal{S}$, in which the simulated $\mathcal{A}$'s view is v. (Note that both E_v and I_v are uniquely determined given $\vec{x}$ and v.) Then $E_v = I_v$.[7]

Showing step 2 is straightforward: the value I_v is the function value (provided by the trusted party) at input $\vec{x}$. It follows from the description of $\hat{\pi}$ that E_v equals the value of the circuit on inputs $\vec{x}$. (This follows from the fact that the value of each wire in the

[7] If the computed function is randomized, then E_v and I_v are random variables having the same distribution.

Simulator $\mathcal{S}$

Initial input: A set C of corrupted parties, inputs $\{x_i \mid P_i \in C\}$, auxiliary input z, and random input r. In addition, $\mathcal{S}$ has access to a trusted party in the ideal model for evaluating f.

1. Invoke a copy of $\mathcal{A}$, on set C of corrupted parties, inputs $\{x_i \mid P_i \in C\}$, auxiliary input z, and a sufficiently long portion of r.
2. For each party $P_{T,i}$, simulate an interaction of team T with the trusted party for computing SHARE$_{s,i}$. That is, for each $P_{T,i}$ and for each corrupted party P_j in team T, provide $\mathcal{A}$ with a random number in $GF[p]$ as the value given by the trusted party in the evaluation of SHARE$_{s,i}$. If $P_{T,i}$ is corrupted, then $\mathcal{A}$ hands an input value, denoted $y_{T,i}$, to the trusted party. Record this value.
3. For each gate g in the circuit, simulate the "baton hand-off" step of the shares of the input wires 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 wires 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 hand $\mathcal{A}$ a random number in $GF[p]$. (If both $P_{T,i}$ and $P_{T_1,i}$, resp., $P_{T_2,i}$, are corrupted, then $\mathcal{A}$ already knows the corresponding share and no action is needed.)
4. Once the "baton hand-off" step of a gate g is simulated, simulate 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 hand $\mathcal{A}$ 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 hand $\mathcal{A}$ a random number in $GF[p]$ as the value given by the trusted party to each corrupted party in team T.
5. When the simulation of a gate leading to an output wire of the circuit is complete, simulate an interaction with the trusted party for computing RECONS$_{s,W}$, where W is the set of parties that are to learn the value of this wire. This is done as follows. If no corrupted party is in W, then no action is needed. Otherwise, invoke the trusted party for the output value of the main function, f. Let v be the value received from the trusted party. Let T be the team that holds the value of this wire, and let a_i denote the share that $\mathcal{A}$ hands its trusted party for RECONS$_{s,W}$, in the name of each corrupted party $P_{T,i}$ in T. Then choose a random polynomial A of degree t such that $A(0) = v$ and $A(\mu_i) = a_i$ for each corrupted party $P_{T,i}$. (Note that this can always be done since $\mathcal{A}$ corrupts at most t parties.) Next, for each corrupted $P_{T,i}$, hand $\mathcal{A}$ the vector $(A(\mu_1), \ldots, A(\mu_s))$.
6. When $\mathcal{A}$ generates its output, output whatever $\mathcal{A}$ does and halt.

Fig. 1. Description of the simulator for protocol $\hat{\pi}$.

circuit equals the value at point 0 of the polynomial that the parties associate with this wire.) Since the circuit computes the function we have $E_v = I_v$.

We complete the proof by showing step 1. That is we show, by induction on the number of rounds, that the adversary views of the real and simulated executions are identically distributed. To see this, fix a prefix p of the adversary view up to some round, and consider the probability of some continuation c of this prefix to the next round. We claim that the probability that continuation c occurs, given prefix p, is identical in the real and simulated interactions. To see that, consider the three possible types of components of the adversary view at a given round:

1. Messages arriving from the trusted party, regarding an evaluation of either SHARE or MULT. In an interaction in the hybrid 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, the corresponding (at most t) values received by $\mathcal{A}$ are independently chosen random numbers in $GF[p]$. However, these 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 matches the values held by the corrupted parties.
3. Messages arriving from uncorrupted parties in a "baton hand-off' stage. These messages are completely determined by the prefix p.

This completes the proof of Claim 1. □

4.3.2. *Proof of Claim* 2

Fix some values of the inputs $\vec{x}$ for the parties, and auxiliary input z for the adversary. We show that

$$\text{EXEC}_{\pi_R,\mathcal{A}}(\vec{x}, z) \overset{\text{d}}{=} \text{EXEC}^{\text{RAND}_n}_{\pi,\mathcal{A}}(\vec{x}, z). \tag{4}$$

Recall that each side of (4) consists of the outputs of the uncorrupted parties, concatenated with the output of the adversary. It can be readily seen that the outputs of the uncorrupted parties in the execution of π and in the execution of π_R are identical. (In both cases, these are the corresponding output values of the evaluated *deterministic* function f on input $\vec{x}$.) It remains to be demonstrated that the adversary's view is equally distributed in the two cases.

To see this, 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) consists of:

1. Random numbers, obtained from the ideal evaluation of RAND$_n$, 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) multiplication-gate evaluations in which P_i participates it receives additional t coefficients to be used in the evaluation. Altogether, at most $t(\ell + 1) = O(t \cdot \ell)$ elements in $GF[p]$ are received, 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 $Q(\mu_r)$ 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 $Q(\mu_r) = \sum_{m=1}^{t} Z_{k_m}\mu_r^m + S$, 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 $Q(\mu_r) = \sum_{m=1}^{t} Z_{k_m}\mu_r^m + S$, 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 $Q(\mu_r) = \sum_{j=1}^{s}(\sum_{m=1}^{t} Z_{k_{j,m}}\mu_r^m + S_j)$, 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 $s + \ell(s + 2)$ elements in $GF[p]$.

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 dth 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. Consequently, the distribution of the adversary's view in the case where the dealer deals totally independent numbers is the same as in the case where it chooses them according to our scheme. $\square$

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).

4.3.3. *Proof of Claim* 3 (*the Protocol without Trusted Dealer*)

The protocol of Section 4.2 assumes a trusted dealer whose role is restricted to choosing random integers in $GF[p]$ and distributing them to the parties. Equivalently, we think of that protocol as running in a hybrid model with ideal access to the function RAND$_n$ described above. (Recall that function RAND$_n$ takes empty input and generates an M-tuple $Z = Z_1, \ldots, Z_M$ according to a distribution $\mathcal{Z}$, which is either the distribution from Appendix 6.2 or a β-independent distribution. Each party receives the appropriate subset of Z.)

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 securely evaluating 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 "double up" as the trusted dealer. That is, each of the designated parties (called dealers) generates $M = n \cdot t + m \cdot s \cdot t$ values in $GF[p]$ and distributes the values to the n parties as described for the trusted dealer. Next, each of the n parties locally outputs the sum (over $GF[p]$) of the values received from $P_1, \ldots, P_{t+1}$. Let ρ denote this protocol.

Analysis of protocol ρ. First note that the amount of randomness used in ρ is larger by a factor of $t+1$ than the amount of randomness used to generate a single M-tuple from the above distribution. Consequently, the protocol is $O(t^2 \log n + (m/n)t^5 \log t)$-random.

We show that the protocol t-privately evaluates RAND_n. Informally, as long as at most t dealers are corrupted, the random choices of the (at least one) uncorrupted dealers make sure that the output of each party, being the sum of the values received from the dealers, is uniformly distributed. Furthermore, the outputs of the parties are β-independent.

A rigorous proof requires a bit more care. Recall that for each real-life adversary $\mathcal{A}$ that interacts with the protocol we need to construct an ideal-process adversary $\mathcal{S}_{\mathrm{RAND}}$ that causes the global output of the ideal process to be distributed identically to the global output of running ρ. For this purpose, $\mathcal{S}_{\mathrm{RAND}}$ will first invoke the trusted party for RAND_n and will obtain the outputs of the corrupted parties in the ideal process. Next $\mathcal{S}_{\mathrm{RAND}}$ will generate a view of $\mathcal{A}$ that has the "right distribution", conditioned on the event that the outputs of the corrupted parties are identical to the values received from the trusted party. A more complete description of $\mathcal{S}_{\mathrm{RAND}}$ appears in Fig. 2.

Simulator $\mathcal{S}_{\mathrm{RAND}}$

Initial input: A set C of corrupted parties, auxiliary input z, and random input r. In addition, $\mathcal{S}_{\mathrm{RAND}}$ has access to a trusted party in the ideal process for evaluating RAND_n.

1. Invoke the trusted party for RAND_n, and obtain the output values of the corrupted parties $\{y_i \mid P_i \in C\}$. Recall that each output value y_i consists of a sequence of elements $Z_{s_1}, \ldots, Z_{s_l}$ in $GF[p]$.
2. Invoke $\mathcal{A}$ on the set C of corrupted parties, auxiliary input z, and random input r. Next, determine the messages to be sent from the uncorrupted dealers to the corrupted parties, as follows:

   Recall that each dealer P_i generates M elements in $GF[p]$. Denote these elements by $Z_{i,1}, \ldots, Z_{i,M}$. Each coordinate $s \in [M]$ is assigned to one of the parties, and the corresponding field element is sent to that party. Let s be assigned to some corrupted party P_j. If P_i is corrupted, then $Z_{i,s}$ is determined by the protocol and the adversary's random input r. (Recall that the adversary is passive, thus even corrupted parties follow the protocol.) So it remains to determine $Z_{i,s}$ for uncorrupted dealers P_i. These values are chosen at random from $GF[p]$, under the restriction that $Z_s = \sum_{\text{dealers } P_i} Z_{i,s}$.

   Once the $Z_{i,s}$'s are determined, group them into messages sent from uncorrupted dealers to corrupted parties, and hand these messages to $\mathcal{A}$.
3. When $\mathcal{A}$ halts, output whatever $\mathcal{A}$ outputs and halt.

Fig. 2. Description of the simulator for protocol ρ.

Analysis of simulator $\mathcal{S}_{\text{RAND}}$. Fix some value for the auxiliary input z. We show that

$$\text{IDEAL}_{\text{RAND}_n,\mathcal{S}_{\text{RAND}}}(z) \overset{\text{d}}{=} \text{EXEC}_{\rho,\mathcal{A}}(z). \qquad (5)$$

Assume, without loss of generality, that the (real-life) adversary $\mathcal{A}$ outputs its entire view of the interaction. This view consists of the set C, the auxiliary input z, some random input r, and the values received from the uncorrupted dealers. Let $\bar{\tau}$ denote the number of *un*corrupted dealers. To see that (5) holds, we observe that the distributions in both sides of (5) are obtained in the same way from a single distribution, $\mathcal{Z}^{(t+1)}$. This distribution is obtained by choosing l M-tuples independently from $\mathcal{Z}$ and summing them coordinatewise, modulo p. (As a side remark we note that distributions $\mathcal{Z}$ and $\mathcal{Z}^{(t+1)}$ are in fact identical.)

It is readily seen from the protocol and from the construction of $\mathcal{S}_{\text{RAND}}$ that the distributions $\text{IDEAL}_{\text{RAND}_n,\mathcal{S}_{\text{RAND}}}(z)$ and $\text{EXEC}_{\rho,\mathcal{A}}(z)$ are obtained from distribution $\mathcal{Z}^{(t+1)}$ in the same way, as follows. Choose an M-tuple $Z_1, \ldots, Z_M$ from $\mathcal{Z}^{(t+1)}$. Let v_i denote the collection of the values out of $Z_1, \ldots, Z_M$ whose coordinates are assigned to P_i. (That is, $v_i = Z_{s_1}, \ldots, Z_{s_l}$ for some predefined value of l.) The output of each uncorrupted party P_i is set to v_i. The view of the adversary (either $\mathcal{A}$ or $\mathcal{S}_{\text{RAND}}$) is obtained as follows. First, include the set C of corrupted parties, the auxiliary input z, and the random input r. It remains to determine the values received from the uncorrupted dealers. These values are chosen randomly from $GF[p]$, under the constraint that the output of each corrupted party P_i matches v_i. That is, let s be assigned to some corrupted party P_j. For each dealer P_i, let $Z_{i,s}$ denote the value that P_j receives from P_i. If P_i is corrupted, then $Z_{i,s}$ is determined by the protocol and the adversary's random input r. The elements $Z_{i,s}$ for uncorrupted dealers P_i are chosen at random from $GF[p]$, under the restriction that $Z_s = \sum_{\text{dealers } P_i} Z_{i,s}$.

5. An Overview of the Protocol of [BGW] for Active Adversaries

The general outline of the construction of [BGW] for the case of active (Byzantine) adversaries is very similar to the case of passive adversaries. Yet, the definitions of the four "building blocks," SHARE, ADD, MULT, RECONS, have to be modified to reflect the additional power of the adversary. As before, let $p > n$ be a prime and let $\mu_1, \ldots, \mu_n$ be n evaluation points. We now define the following n-party functions:

Verifiable secret sharing. $\text{VSS}_n(s|F(\cdot), \varepsilon, \ldots, \varepsilon) = \alpha_1, \ldots, \alpha_n$, where $s \in GF[p] \cup \{\varepsilon\}$, and $F(\cdot)$ is either ε or a polynomial of degree t over $GF[p]$. If $s \neq \varepsilon$, then $\alpha_i = E(\mu_i)$, where $E()$ is a random polynomial of degree t with $E(0) = s$. (This case represents a sharing by an uncorrupted dealer.) If $s = \varepsilon$ and $F() = \varepsilon$, then $\alpha_i = \varepsilon$. (This case represents an unsuccessful sharing by a corrupted dealer.) Otherwise ($s = \varepsilon$ and $F() \neq \varepsilon$), then $\alpha_i = F(\mu_i)$. (This case represents a successful sharing by a corrupted dealer; here the adversary can determine the outputs of all parties.)[8]

[8] This formalization captures VSS schemes where the uncorrupted parties know at the end of the sharing phase whether the sharing of a secret was successful. Schemes where this information becomes known only later (such as some of the schemes in [GRR]) should be formalized differently.

Looking ahead, we note that uncorrupted parties will invoke VSS with $s \neq \varepsilon$.

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. $\text{ACT-MULT}_n(a_1|b_1|c_1, \ldots, a_n|b_n|c_n) = C(\mu_1), \ldots, C(\mu_n)$, where each $a_i, b_i \in GF[p]$, $c_i \in GF[p] \cup \{\varepsilon\}$, and C is a polynomial 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 $A(\mu_i) = a_i$ (resp., $B(\mu_i) = b_i$) for at least $n - t$ of the parties. Then $C(0) = A(0) \cdot B(0)$.

(II) If $c_i \neq \varepsilon$, then $C(\mu_i) = c_i$. 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 $c_i = \varepsilon$. 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 multiplication step of [BGW] allows the adversary to have such harmless influence.)

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

An additional change from the passive case is that here error correction is required for obtaining the value of an output line of the circuit. That is, each party P_i with $i \in W$ for some invocation of $\text{RECONS}_{n,W}(a_1, \ldots, a_n)$ receives the values $a_1, \ldots, a_n$; these values constitute a perturbed code-word of a Generalized Reed–Solomon code. The value of this line is the free coefficient of the (unique) degree-t polynomial defined by $a_1, \ldots, a_n$. This polynomial can be computed using the Berlekamp–Welch algorithm. (See, for instance, [MS] and [S3].)

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

We do not prove this theorem here. Yet we sketch below the constructions of [BGW] for computing VSS and ACT-MULT.

The VSS protocol of [BGW]. Here a Verifiable Secret Sharing (VSS) scheme is used instead of Shamir's secret sharing. (VSS was introduced in [CGMA]; different VSS schemes are described in [CGMA], [GMW], [BGW], [FM], [CCD], [BCG], and [GRR].) In general, 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. A popular methodology (followed by [BGW] and used in this paper) for constructing a VSS scheme is to design protocols for secure evaluation of the functions VSS and RECONS. We sketch a VSS scheme described in [BGW], 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 $f_i(\cdot) = H(\mu_i, \cdot)$ and $g_i(\cdot) = H(\cdot, \mu_i)$ to each P_i. Then each P_i sends $f_i(\mu_j)$ to each P_j, and verifies that the value received from P_j equals $g_i(\mu_j)$. (Note that $f_i(\mu_j) = H(\mu_i, \mu_j) = g_j(\mu_i)$.) If any of its verifications fails, the party requests the dealer to make the corresponding value (i.e., $H(\mu_i, \mu_j)$) public. Next, each party P_i inspects all publicized values. If any of these values does not match P_i's private share (i.e., $f_i()$ and $g_i()$), then P_i requests the dealer to make $f_i()$ and $g_i()$ 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 and sets its share to a default 0. Otherwise, it sets its share of the secret to be $f_0(\mu_i) = g_i(0)$.

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 the unique polynomial $f_0(\cdot) = H(0, \cdot)$ will be reconstructed, using error correcting techniques of Generalized Reed–Solomon codes. The reconstructed secret is $f_0(0) = H(0, 0)$.

The ACT-MULT *protocol of* [BGW]. Several methods for evaluating a multiplication step are sketched in [BGW]. An additional, simpler method is described in [GRR]. Here we only sketch a simple method that works when the fraction of corrupted parties is less than a *fourth*. (This method combines techniques from [BGW] with the passive multiplication step of [GRR].) The method here requires a total of $O(nt^2 \log p)$ random bits per multiplication gate. For the case of $4t \geq n > 3t$ a total of $O(nt^3 \log p)$ random bits are required per multiplication gate.

Recall that, in the case of passive adversaries, evaluating a multiplication gate consists of each party resharing a locally computed value, followed by local evaluation of a linear combination of the newly received shares. The same method is followed here, with two modifications:

- Each party reshares the locally computed value using the VSS scheme described above. It should be noted that the local evaluation of the linear combination of the newly received shares can still be done, since it is guaranteed that the share of each party is a value $f_0(\mu_i)$ of a random polynomial whose free coefficient is the secret.
- For each party P_i, the parties verify that the value d_i that P_i reshares is indeed the product of P_i's 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 $2t$. (This polynomial is the product of the polynomials associated with the input wires to the gate.) Thus the set of values that were reshared 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. As long as $n > 4t$, the code-word can be used to identify uniquely and correct up to t erroneous entries. Note that no party knows the entire code-word. 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 (the efficiency of this computation builds upon the specific choice of evaluation points μ_i). These shares are not used in the computation of the linear combination (3).

For completeness, we also state the following theorem:

Theorem 6 [BGW]. *Let $t < n/3$. Given an arithmetic circuit for computing an n-party function f, there exists a protocol for t-securely computing f in the hybrid 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 $i \in [n]$ and $W \subseteq [n]$.*

Using the composition theorem (Theorem 1), we get that there exist protocols for t-securely computing (in the real-life model) any n-party function f in the presence of active adversaries for any $t < n/3$. The number of random bits used by these protocols is $O(mnt^3 \log p)$ (where m is the size of the circuit for f).

6. Our Protocol for Active Adversaries

In Section 4 we showed how to compute any function t-privately with an $O(t^2 \log n + (m/n)t^5 \log t)$-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^2 more randomness than before. We show:

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

Proof. 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 [BGW] protocol are replaced by their Byzantine counterparts, for securely computing the functions VSS, ADD, ACT-MULT, RECONS described in the previous section. As in Section 4, it suffices to choose $p > s$ and s evaluation points $\mu_1, \ldots, \mu_s$.

The protocol for jointly generating the randomness for the computation remains unchanged, except for the appropriate increase in the amount of randomness generated. That is, each one of $t + 1$ designated dealers will sample the distribution and send the appropriate subset of the obtained M-tuple to each party; each party will sum, coordinatewise, the tuples received from the dealers. The security guarantees provided by this protocol are a bit weaker than in the passive case. We capture these guarantees via a somewhat weaker formalization of the function representing the trusted dealer. We call this function ACT-RAND_n.

The function ACT-RAND_n. We describe the function, denoted ACT-RAND_n, that represents the requirements from the randomness-generating protocol in the Byzantine case. There are two differences from the passive case (i.e., from function RAND_n). First, ACT-RAND_n has to supply the parties with sufficiently many elements of $GF[p]$ to support the new protocols. In addition, it has to accommodate the fact that an active adversary can modify the outputs of the corrupted parties, as well as to influence the outputs of the uncorrupted parties (to a limited extent).[9]

We start by counting the number of field elements that ACT-RAND$_n$ should output, and bound the required level of independence. Each invocation of VSS requires $O(t^2)$ values in $GF[p]$. Evaluating a multiplication gate requires $O(t)$ invocations of VSS for each party in the corresponding team. Furthermore, the adversary's view of the computation now depends on at most $\beta = O((m/n) \cdot t^6)$ elements in $GF[p]$. Thus, ACT-RAND$_n$ will use the distribution $\mathcal{Z}$ which is either the distribution described in Appendix 6.2, with $M = O(n \cdot t^2 + m \cdot s \cdot t^3)$ and the appropriate independence guarantees, or simply a β-independent distribution with the above value of M.

We proceed to formalize ACT-RAND$_n$. This is a distribution of M-tuples of elements $Z_1, \ldots, Z_M$ in $GF[p]$, where each Z_i is assigned to a party. Distribution $\mathcal{Z}^{(t+1)}$ is obtained by choosing l M-tuples independently from $\mathcal{Z}$ and summing them coordinatewise, modulo p. Function ACT-RAND$_n$ takes inputs $v_1, \ldots, v_n$ where v_i, the input of P_i, is either ε or is interpreted as a sequence of elements from $GF[p]$. (Uncorrupted parties will invoke ACT-RAND$_n$ with input ε.) The function value is $y_1, \ldots, y_n$ (party P_i gets y_i), where each y_i is a sequence of numbers in $GF[p]$. It is helpful to regard the concatenation of $y_1, \ldots, y_n$ as an M-tuple $Z_1, \ldots, Z_M$ of elements in $GF[p]$, where y_i consists of the elements whose coordinates are assigned to P_i. The M-tuple $Z_1, \ldots, Z_M$ is computed via the following procedure:

1. For each dealer P_i, if $v_i \neq \varepsilon$, then v_i is interpreted as a pair $v_i = (v_i', v_i'')$, where v_i' is interpreted as values for the elements of $GF[p]$ whose coordinates are assigned to P_i, and v_i'' is interpreted as an M-tuple of elements in $GF[p]$.
   If P_i is not a dealer and $v_i \neq \varepsilon$, then v_i is interpreted as v_i' described above.
2. Let $\hat{Z} = \hat{Z}_1, \ldots, \hat{Z}_M$ be an M-tuple that is chosen from the distribution $\mathcal{Z}^{(t+1)}$, under the constraint that for each coordinate s that is assigned to a party P_i where $v_i \neq \varepsilon$, the value $\hat{Z}_s$ equals the value specified in v_i'. (We remark that this conditional distribution is efficiently samplable.)
3. The output M-tuple $Z_1, \ldots, Z_M$ is the coordinatewise sum, modulo p, of $\hat{Z}$ with all the M-tuples v_i'' that are not ε.

Intuitively, the value v_i' allows a corrupted party P_i to influence its own local output. The value v_i'' allows a corrupted dealer P_i to influence the output distribution of the uncorrupted parties. Nevertheless, the definition of ACT-RAND$_n$ guarantees that the outputs of the uncorrupted parties will be uniformly distributed in $GF[p]$ and will have at least the amount of independence guaranteed by distribution $\mathcal{Z}$. Furthermore, the adversary's view in the ideal process for evaluating ACT-RAND$_n$ consists only of the projection of Z on the coordinates assigned to the corrupted parties, plus some independently distributed information.

Analysis of the protocol. The analysis is very similar to the passive case. Let $t < n/3$, and let f be the computed function. Fix an arithmetic circuit for f and a large enough

[9] For instance, the adversary may have the corrupted parties send totally random values, thus making the output of ACT-RAND$_n$ totally independent. Intuitively, however, such deviations are "harmless." This intuition is made rigorous in the formalization of ACT-RAND$_n$.

prime p. Let π be the protocol described above with respect to that circuit. (Protocol π is designed in the ACT-RAND_n-hybrid model.) We show that protocol π satisfies the conditions of Definition 1 via three claims, similar to Claims 1–3. Let π_R be identical to protocol π with the exception that the parties use totally random elements in $GF[p]$ for the protocol, instead of the output of ACT-RAND_n. (That is, π_R is a protocol in the real-life model, and does not use calls to ACT-RAND_n.)

Claim 4. *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 z,*

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

Claim 5. *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, $\text{EXEC}_{\pi_R,\mathcal{A}}(\vec{x}, z) \stackrel{\text{d}}{=} \text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, z)$.*

Claim 6. *There exists an $O(t^3 \log n + (m/n)t^7 \log t)$-random protocol that t-securely evaluates ACT-RAND_n.*

Note that, unlike Claim 2, in Claim 5 both random variables are a result of interaction with an *active* adversary. Still, the proof of Claim 5 is almost identical to the proof of Claim 2, and is therefore omitted. Claims 4 and 6 are proven below. This completes the proof of Theorem 7. □

6.1. *Proof of Claim* 4

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 hybrid model with ideal evaluation access to functions $\text{VSS}_{s,i}$, ADD_s, ACT-MULT_s, and $\text{RECONS}_{s,W}$. It suffices to show that $\hat{\pi}$ is t-secure in the hybrid model. Theorems 1 and 5 then 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 $\mathcal{A}$. Simulator $\mathcal{S}$ starts running $\mathcal{A}$ on its auxiliary input z_0 and random input $r_{\mathcal{A}}$. Next, $\mathcal{A}$ may corrupt parties, and will expect to see the internal data and the messages received by the corrupted parties. Simulator $\mathcal{S}$ proceeds as described in Fig. 3.

The analysis of $\mathcal{S}$ (i.e., the proof that for all inputs $\vec{x}$ and all auxiliary inputs z we have $\text{IDEAL}_{f,\mathcal{S}}(\vec{x}, z) \stackrel{\text{d}}{=} \text{EXEC}^{\text{VSS,ADD,ACT-MULT,RECONS}}_{\hat{\pi},\mathcal{A}}(\vec{x}, z)$) is identical to the corresponding part of the proof of Claim 1, except for the following point that relates to step 2 of the analysis there. Contrary to the passive case, in the active case the inputs that a corrupted P_i hands the trusted party (in the ideal model) may be different than x_i. Yet, it still holds that the inputs that the corrupted parties give the trusted party are uniquely determined given a view v of the simulated $\mathcal{A}$. Furthermore, let $\vec{y}$ denote the modified input vector; then it can be verified that both I_v and E_v equal the value of the circuit on inputs $\vec{y}$.[10] □

[10] The introduction of active adversaries raises an additional apparent difficulty. When an uncorrupted party

Simulator $\mathcal{S}$

Initial input: A set C of corrupted parties, inputs $\{x_i \mid P_i \in C\}$, auxiliary input z, and random input r. In addition, $\mathcal{S}$ has access to a trusted party in the ideal process for evaluating f.

1. Invoke a copy of $\mathcal{A}$, on set C of corrupted parties, inputs $\{x_i \mid P_i \in C\}$, auxiliary input z, and a sufficiently long portion of r.
2. For each party $P_{T,i}$, simulate an interaction of team T with the trusted party for computing $\text{VSS}_{s,i}$. That is, if $P_{T,i}$ is corrupted, then record the value $s|F()$ handed by $\mathcal{A}$ to its trusted party, and hand $\mathcal{A}$ the output value for each corrupted party $P_{T,j}$. (This value is determined by $s|F()$, as described in the definition of function $\text{VSS}_{s,i}$.) If $P_{T,i}$ is not corrupted, then, for each corrupted party in team T, hand $\mathcal{A}$ 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 provide the trusted party for f with $P_{T,i}$'s input for f, computed as follows: If $s \neq \varepsilon$, then $P_{T,i}$'s input is set to s. Else, if $F()$ is a polynomial of degree t, then the input value of $P_{T,i}$ is set to $F(0)$. Otherwise ($s = F() = \varepsilon$), the input of $P_{T,i}$ is set to a default value, say 0.
3. (This part is identical to the simulator for the passive case, see Fig. 1.) For each gate g in the circuit, simulate the "baton hand-off" step of the shares of the input wires 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 wires 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 hand $\mathcal{A}$ a random number in $GF[p]$. If both $P_{T,i}$ and $P_{T_1,i}$ (resp., $P_{T_2,i}$) are corrupted, then $\mathcal{A}$ already knows the corresponding share and no action is needed.
4. Once the "baton hand-off" step of a gate g is completed, simulate 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). This is done as follows.

   If the gate g is an addition gate, then hand $\mathcal{A}$ 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 hand $\mathcal{A}$ 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 ($c_i = \varepsilon$), v_i is set to a random number in $GF[p]$.
5. When the simulation of a gate leading to an output wire of the circuit is complete, simulate an interaction with the trusted party for computing $\text{RECONS}_{s,W}$, where $W \subseteq [n]$ is the set of parties that are to learn the value of this wire. If no corrupted party is in W, then no action is needed. Otherwise, invoke the trusted party for 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 choose 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 $B(\mu_i) = a_i$ for each *conforming* corrupted party $P_{T,i}$. Next, hand each corrupted $P_{T,i}$ the vector $(B(\mu_1), \ldots, B(\mu_s))$. (Note that this can always be done since $\mathcal{A}$ corrupts at most t parties.)
6. Once $\mathcal{A}$ halts, output whatever $\mathcal{A}$ outputs and halt.

Fig. 3. The simulator for protocol $\hat{\pi}$, Byzantine case.

$P_{T_i,j}$ 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 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.

6.2. *Proof of Claim* 6

The protocol for securely evaluating ACT-RAND$_n$ is very similar to the protocol for the passive case (for computing RAND$_n$). We designate $t+1$ parties (say, $P_1, \ldots, P_{t+1}$) who, in addition to their other roles in the protocol, will "double up" as dealers. Each one of the dealers generates $M = O(n \cdot t^2 + m \cdot s \cdot t^2)$ values in $GF[p]$ according to the distribution $\mathcal{Z}$. Let $v_{i,j}$ be the vector consisting of all the elements in $GF[p]$ that are chosen by P_i and are assigned to P_j. Then P_i sends $v_{i,j}$ to each party P_j. Each party P_j locally outputs the coordinatewise sum, modulo p, of the $t+1$ vectors received from the dealers. Let ρ_{A} denote this protocol for the active case.

Analysis of protocol ρ_{A}. First note that the amount of randomness used in ρ_{A} is larger by a factor of $t+1$ than the amount of randomness used to generate a single M-tuple from the above distribution. Sampling from the distribution of Appendix 6.2 takes $O(t^2 \log n + (m/n)t^6 \log t)$ random bits. Consequently, the protocol is $O(t^3 \log n + (m/n)t^7 \log t)$-random. (Instead we can use an $O((m/n)t^7)$-wise independent distribution and pay $O((m/n)t^7 \log m)$ random bits.)

We show that the protocol t-securely evaluates ACT-RAND$_n$. As in the passive case, the intuition is that, as long as at least one dealer remains uncorrupted, the random choices of the uncorrupted dealers make sure that the outputs of the parties, being the sum of the values received from the dealers, have the desired independence structure. (Still, it should be noted that in the active case the adversary can somewhat influence the distribution of the outputs of the uncorrupted parties.)

Also here, a rigorous proof is a bit more involved. Recall that for each active real-life adversary $\mathcal{A}$ that interacts with the protocol we need to construct an ideal-process adversary $\mathcal{S}_{\text{ACT-RAND}}$ that causes the global output of the ideal process to be distributed identically to the global output of running ρ. In the passive case, this was done by making sure that the messages generated by $\mathcal{S}_{\text{RAND}}$ (representing the messages sent by the uncorrupted parties) "match" the values provided by the trusted party for RAND$_n$. This was possible since the messages generated by $\mathcal{A}$ depended only on the random input of $\mathcal{A}$.

In the active case the messages generated by $\mathcal{A}$ (representing the messages sent by the corrupted parties) may depend on the messages generated by $\mathcal{S}_{\text{ACT-RAND}}$. Consequently, in this case we do not know how to generate a view of $\mathcal{A}$ that is consistent with values that are chosen by the trusted party. Instead, we use the fact that function ACT-RAND$_n$ allows $\mathcal{S}_{\text{ACT-RAND}}$ to influence somewhat the outputs of the parties. Simulator $\mathcal{S}_{\text{ACT-RAND}}$ is presented in Fig. 4.

Analysis of simulator $\mathcal{S}_{\text{ACT-RAND}}$. Fix some input vector $\vec{x}$ and auxiliary inputs z. We show that

$$\text{IDEAL}_{\text{ACT-RAND}_n, \mathcal{S}_{\text{ACT-RAND}}}(z) \overset{\text{d}}{=} \text{EXEC}_{\rho_{\mathrm{A}}, \mathcal{A}}(z). \tag{6}$$

The analysis is very similar to the passive case (Claim 3). Assume, without loss of generality, that $\mathcal{A}$ outputs its entire view of the interaction. This view consists of the set C, the auxiliary input z, some random input r, and the values received from the

Simulator $\mathcal{S}_{\text{ACT-RAND}}$

Initial input: A set C of corrupted parties, auxiliary input z, and random input r. In addition, $\mathcal{S}_{\text{ACT-RAND}}$ has access to a trusted party in the ideal process for evaluating ACT-RAND$_n$.

1. Invoke $\mathcal{A}$ on the set C of corrupted parties, auxiliary input z, and random input r. For each uncorrupted dealer P_i and each corrupted party P_j, provide $\mathcal{A}$ with a message $v_{i,j}$ that consists of uniformly distributed elements in $GF[p]$. (The number of field elements in $v_{i,j}$ equals the number of field elements in P_i's output in the specification of function ACT-RAND$_n$.)
2. For each corrupted dealer P_i and uncorrupted party P_j, adversary $\mathcal{A}$ generates a message $v_{i,j}$ to be sent from P_i to P_j. (Each $v_{i,j}$ is interpreted as a vector of elements from $GF[p]$.) Record those messages.
3. Prepare the input values of the corrupted parties in the ideal process, as follows. (These values will be handed to the trusted party for ACT-RAND$_n$.) Recall that the input of each corrupted party P_i is v_i; if P_i is a dealer, then $v_i = v_i', v_i''$; otherwise $v_i = v_i'$. Then:
   (a) Each v_i' is the coordinatewise sum modulo p of the values $v_{j,i}$ that P_i received from the uncorrupted dealers. (These values were handed to $\mathcal{A}$ in step 1 above.)
   (b) Let v_i''' be the coordinatewise sum modulo p of the values $v_{i,j}$ that dealer P_i sent to the uncorrupted parties. (These values were generated by $\mathcal{A}$ in step 2 above.) Let v_i'' be the completion of v_i''' to an M-tuple, computed by placing the value 0 in all the missing coordinates. (These locations correspond to the coordinates that are assigned to corrupted parties.)

   Hand these inputs to the trusted party for ACT-RAND$_n$. (Recall that the inputs of the uncorrupted parties are ε.) The outputs provided by the trusted party to the corrupted parties can be ignored; they are equal to the inputs v_i'.
4. Once $\mathcal{A}$ halts, output whatever $\mathcal{A}$ outputs and halt.

Fig. 4. Description of the simulator for protocol ρ_{A}.

uncorrupted dealers. Let $\bar{\tau}$ denote the number of *un*corrupted dealers. To see that (6) holds, we observe that the distributions in both sides of (6) are equal to a distribution that is generated as follows:

1. Invoke adversary $\mathcal{A}$ with uniformly chosen random input r, auxiliary input z, and set C of corrupted parties.
2. Independently choose $\bar{\tau}$ M-tuples from distribution $\mathcal{Z}$. Hand all $\mathcal{A}$ the elements in $GF[p]$ whose coordinates are assigned to corrupted parties.
3. Adversary $\mathcal{A}$ generates the messages to be sent by the corrupted dealers to the uncorrupted parties. Let $w_i = w_{i,1}, \ldots, w_{i,M}$ denote the M-tuple of elements in $GF[p]$ that represents the messages sent by each corrupted dealer P_i to all uncorrupted parties. (The elements $w_{i,s}$ such that s is assigned to a corrupted party are set to a default 0.)
4. Let $Z = Z_1, \ldots, Z_M$ denote the coordinatewise sum modulo p of the $\bar{\tau}$ M-tuples from step 2 and the $t + 1 - \bar{\tau}$ M-tuples w_i from step 3.

   The output of an uncorrupted party P_i consists of the elements in Z whose coordinates are assigned to P_i. The output of the corrupted parties is $\perp$. The output of the adversary consists of k, r, z, C and the values handed to $\mathcal{A}$ in step 2.

This completes the proof of Claim 6. □

Acknowledgments

We thank Oded Goldreich for his insightful comments which greatly improved the presentation of our paper, as well as for suggesting a way to eliminate some more unnecessary randomness by using [S2] and [KM4]. We also thank Madhu Sudan for discussions regarding error-correcting codes. Finally, we would like to thank the anonymous referees for carefully reading the paper and providing us with very useful comments.

Appendix A. An Extension of [S2] and [KM4]

In this appendix we describe a straightforward extension of results from [S2] and [KM4]. (All arithmetic 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 multiset[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 multiset of size p^ℓ:

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

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

Claim 7. *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 a vector y is chosen 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. Recall that a randomly chosen vector y in *space*(M) is just the product $M \cdot v$, for a randomly chosen $v \in (GF[p])^\ell$; in particular, $y_i = M_i \cdot v$. Then

$$\sum_{i \in S_j} w_i \cdot y_i = \sum_{i \in S_j} (w_i \cdot M_i \cdot v) = \sum_{i \in S_j} (w_i \cdot M_i) \cdot v.$$

Since $\sum_{i \in S_j} w_i M_i \neq \vec{0}$ then the above is a product of a nonzero vector with a uniformly distributed vector in $(GF[p])^\ell$ which is just a uniformly distributed element of $GF[p]$. $\square$

The next claim easily follows from Claim 7:

Claim 8. *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.*

[11] By *"multiset"* 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 multiset.

Proof. By Claim 7, 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 $GF[p]$. It is well known that the only probability distribution that is uniform with respect to all "linear tests" is the uniform distribution.[12] $\square$

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 8 with respect to each of the t sets.[13] 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 $\log t / \log p + d + 1$. 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.

Remark (*on the efficiency of the algorithm*). Since the algorithm runs in time $poly(n, t, p^d)$, if the size of any S_j is $\omega(\log n)$, then the running time is superpolynomial. (Clearly, any set that is uniform over S_j must be of size at least $p^{|S_j|}$.) However, even in this case the size of the matrix M is much smaller and so *sampling* in the space remains efficient. Finally, as mentioned in Section 4 (see footnote 6) this construction is used only for the final saving in randomness and one can stick to the (computationally more efficient) solution based on β-wise independent distributions.

Appendix B. A Direct Proof of Claim 1

In this appendix we describe a direct proof for Claim 1; a proof that does not rely on Theorem 1 but rather shows, specifically for our case, how to compose the subsimulators for the various components, as guaranteed by [BGW], into a simulator for the entire protocol. We hope that the reader will get some insight regarding how the composition technique works, avoiding many of the technical details required for the full composition theorem. For this purpose, we concentrate on the passive case.

Given a real-life adversary $\mathcal{A}$, we will construct a simulator $\mathcal{S}$. For this, we define four adversaries $\mathcal{A}_{\text{SHARE}}$, $\mathcal{A}_{\text{ADD}}$, $\mathcal{A}_{\text{MULT}}$, and $\mathcal{A}_{\text{RECONS}}$ that essentially determine the behavior of $\mathcal{A}$ when each of the four subroutines SHARE, ADD, MULT, and RECONS (respectively) are

[12] More formally, since the functions of the form $r^{\Sigma w_i \cdot y_i}$, where r is a root of unity of order p, are just the Fourier basis for $GF[p]$, 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 every such function.

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

executed. We note, however, that the adversary $\mathcal{A}$ need not use the same strategy against all invocations of some subroutine; his decisions may be influenced by the execution of the protocol so far. To overcome this difficulty, we will provide each of the four adversaries with the transcript of the protocol up to the point where the execution of the subroutine starts. This is technically done by including $\mathcal{A}$'s view of the execution of the protocol up to this point in the auxiliary input of the current subadversary. So, for example, adversary $\mathcal{A}_{\text{MULT}}$ behaves exactly as $\mathcal{A}$ behaves in a certain execution of MULT when the history is as provided to $\mathcal{A}_{\text{MULT}}$ via the auxiliary inputs z. As in [C2], we assume that the subroutines are executed sequentially (and not in parallel to each other). This makes the above adversaries well defined.

We can now use Theorem 2 to conclude the existence of four simulators $\mathcal{S}_{\text{SHARE}}$, $\mathcal{S}_{\text{ADD}}$, $\mathcal{S}_{\text{MULT}}$, and $\mathcal{S}_{\text{RECONS}}$, satisfying the definition of security with respect to the four protocols and the four adversaries. Next we show how to construct a simulator $\mathcal{S}$ for our protocol using these four simulators. Roughly speaking, we will explicitly describe how to simulate all messages sent outside of the four subroutines and we use the simulators to simulate all the communication inside the execution of the subroutines. We first describe the procedure for running each of these four simulators. When we run a simulator, say for MULT, we do the following: $\mathcal{S}$ runs the corresponding simulator $\mathcal{S}_{\text{MULT}}$ as it is. (By the definition of $\mathcal{A}_{\text{MULT}}$, this in particular implies that $\mathcal{S}_{\text{MULT}}$ controls the same set of corrupted parties, and that at the end $\mathcal{S}_{\text{MULT}}$ outputs the entire simulated view.) Once $\mathcal{S}_{\text{MULT}}$ generates its output (which is a simulated view of an interaction of $\mathcal{A}_{\text{MULT}}$), simulator $\mathcal{S}$ continues the simulated run of $\mathcal{A}$ on this output. Our choice of the auxiliary inputs to $\mathcal{S}_{\text{MULT}}$ and to the corrupted parties guarantees that the output of $\mathcal{S}_{\text{MULT}}$ is consistent with the prefix of $\mathcal{A}$'s run so far.

We now describe how the simulator $\mathcal{S}$ works, given the above procedure for running a simulator for a subroutine. As common in such protocols, this relies heavily on the properties of degree t polynomials.[14] In detail, messages are simulated as follows:

- For each party $P_{T,i}$, simulator $\mathcal{S}$ simulates an interaction of the parties in team T in the subroutine SHARE$_{s,i}$ where $P_{T,i}$ shares its input $x_{T,i}$. That is, if $P_{T,i}$ is corrupted, then $\mathcal{S}$ executes the simulator $\mathcal{S}_{\text{SHARE}}$ with input $x_{T,i}$ (which is already known to him), while if $P_{T,i}$ is not corrupted, then $\mathcal{S}$ executes the simulator $\mathcal{S}_{\text{SHARE}}$ with a random number in $GF[p]$ as the input.
- For each gate g in the circuit, $\mathcal{S}$ simulates the "baton hand-off" step. That is, let T be the team that computes gate g, and let T_1 and T_2 be the teams that hold the values of the input wires 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 $\mathcal{A}$ a random number in $GF[p]$. If both $P_{T,i}$ and $P_{T_1,i}$ (resp., $P_{T_2,i}$) are corrupted, then $\mathcal{A}$ already knows the corresponding share.
- Once the "baton hand-off" step of a gate g is simulated, $\mathcal{S}$ simulates an interaction of team T in the corresponding subroutine (either ADD$_s$ or MULT$_s$) for computing the function that corresponds to gate g. In each case, $\mathcal{S}$ uses the subsimulator (either

[14] More specifically, it depends on the fact that if we choose such a polynomial at random, or we first choose some $t' \leq t$ random points and then choose a random polynomial out of those polynomials that pass through the chosen t' points, then in both cases we get the same distribution.

$\mathcal{S}_{\text{ADD}}$ or $\mathcal{S}_{\text{MULT}}$) with input to the corrupted players as already known to it, and with (uniformly and independently chosen) random numbers in $GF[p]$ as the inputs of noncorrupted parties.

- At some point during the computation (before any of the invocations of $\text{RECONS}_{s,W}$ is to be simulated), $\mathcal{S}$ hands its trusted party the input values of the corrupted parties, and receives the output values assigned to them.
- When the simulation of a gate leading to an output wire of the circuit is complete, $\mathcal{S}$ simulates the reconstruction by the subroutine $\text{RECONS}_{s,W}$, 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 the corresponding simulator $\mathcal{S}_{\text{RECONS}}$. For this, let v be the value that should correspond to this output wire (the value v was received from the trusted party), 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 holds. Then $\mathcal{S}$ chooses a random polynomial A of degree t such that $A(0) = v$ and $A(\mu_i) = a_i$ for each corrupted party $P_{T,i}$. The simulator $\mathcal{S}$ uses $\mathcal{S}_{\text{RECONS}}$ with input values $(A(\mu_1), \ldots, A(\mu_s))$. $\square$

Analysis of simulator $\mathcal{S}$ (sketch). To show that the output of the simulator is distributed identically to the distribution of the ideal-model adversary, we repeat the analysis made in the proof of Claim 1. The only difference is that in Claim 1 we assume ideal invocations of the four subroutines whereas here the actual subroutines are called. However, based on the properties of the four subsimulators $\mathcal{S}_{\text{SHARE}}$, $\mathcal{S}_{\text{ADD}}$, $\mathcal{S}_{\text{MULT}}$, and $\mathcal{S}_{\text{RECONS}}$ (that is, the fact that the output generated by each such simulator, given the appropriate inputs and auxiliary inputs, is identical to the distribution of output by the corresponding adversary) the analysis still goes through.

References

[AGHP] 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. 3, 1992, pp. 289–304. (Addendum: Vol. 4, 1993, pp. 119–120).

[B1] D. Beaver, Perfect Privacy for Two-Party Protocols, TR-11-89, Harvard University, 1989.

[B2] D. Beaver, Foundations of Secure Interactive Computing, *Proc. CRYPTO*, 1991, pp. 377–391.

[B3] D. Beaver, Secure Multiparty Protocols and Zero-Knowledge Proof Systems Tolerating a Faulty Minority, *J. Cryptology*, Vol. 4, 1991, pp. 75–122.

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

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

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

[BDV] C. Blundo, A. De-Santis, and U. Vaccaro, Randomness in Distribution Protocols, *Proc. ICALP*, LNCS 820, Springer-Verlag, Berlin, 1994, pp. 568–579.

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

[BGS] C. Blundo, A. Giorgio Gaggia, and D. R. Stinson, On the Dealer's Randomness Required in Secret Sharing Schemes, EuroCrypt94 and *Designs Codes Cryptography*, Vol. 11, No. 2, 1997, pp. 235–259.

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

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

[C1] R. Canetti, Studies in Secure Multi-Party Computation and Applications, Ph.D. Thesis, Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, June 1995.

[C2] R. Canetti, Security and Composition of Multiparty Cryptographic Protocols, *J. Cryptology*, this issue.

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

[CD] B. Chor and C. Dwork, Randomization in Byzantine Agreement, *Adv. Comput*. *Res*., Vol. 5, 1989, pp. 443–497.

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

[CG1] B. Chor and O. Goldreich, Unbiased Bits from Sources of Weak Randomness and Probabilistic Communication Complexity, FOCS 85 and *SICOMP*, Vol. 17, 1988, pp. 230–261.

[CG2] R. Canetti and O. Goldreich, Bounds on Tradeoffs between Randomness and Communication Complexity, FOCS 90 and *Comput*. *Complexity*, Vol. 3, 1993, pp. 141–167.

[CGMA] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch, Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults, *Proc*. *FOCS*, 1985, pp. 383–395.

[CK1] B. Chor and E. Kushilevitz, A Zero-One Law for Boolean Privacy, STOC 89 and *SIDMA*, Vol. 4, 1991, pp. 36–47.

[CK2] B. Chor and E. Kushilevitz, A Communication-Privacy Tradeoff for Modular Addition, *Inform*. *Process*. *Lett*., Vol. 45, 1993, pp. 205–210.

[CRS] S. Chari, P. Rohatgi, and A. Srinivasan, Randomness-Optimal Unique Element Isolation, with Application to Perfect Matching and Related Problems, *Proc*. *STOC*, 1993, pp. 458–467.

[FKN] U. Feige, J. Kilian, and M. Naor, A Minimal Model for Secure Computation, *Proc*. *STOC*, 1994, pp. 554–563.

[FL] M. Fischer and N. Lynch, A Lower Bound for the Time to Assure Interactive Consistency, *IPL*, Vol. 14, No. 4, 1982, pp. 183–186.

[FLP] M. J. Fischer, N. A. Lynch, and M. S. Paterson, Impossibility of Distributed Consensus with One Faulty Process, *J. Assoc*. *Comput*. *Mach*., Vol. 32, No. 2, 1985, pp. 374–382.

[FM] P. Feldman and S. Micali, An Optimal Algorithm for Synchronous Byzantine Agreement, STOC 88 and *SIAM J. Comput*., Vol. 26, No. 4, 1997, pp. 873–933.

[FY] M. Franklin, and M. Yung, Communication Complexity of Secure Computation, *Proc*. *STOC*, 1992, pp. 699–710.

[G] O. Goldreich, *Foundations of Cryptography* (*Fragments of a Book*), Weizmann Institute of Science, 1995. (Avaliable at http://philby.ucsd.edu.)

[GMW] O. Goldreich, S. Micali, and A. Wigderson, How to Play Any Mental Game, *Proc*. 19*th STOC*, 1987, pp. 218–229.

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

[GMR] S. Goldwasser, S. Micali, and C. Rackoff, The Knowledge Complexity of Interactive Proof Systems, *SIAM J. Comput*., Vol. 18, No. 1, 1989, pp. 186–208.

[GO] O. Goldreich and Y. Oren, On the Cunning Power of Cheating Verifiers: Some Observations about Zero-Knowledge Proofs, in preparation. Preliminary version by Y. Oren in *Proc*. 28*th FOCS*, 1987.

[GRR] R. Gennaro, T. Rabin, and M. Rabin, Simplified VSS and Fact-Track Multiparty Computations with Applications to Threshold Cryptography, *Proc*. *PODC*, 1998, pp. 101–111.

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

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

[IZ] R. Impagliazzo and D. Zuckerman, How to Recycle Random Bits, *Proc*. 30*th FOCS*, 1989, pp. 248–253.

[K] E. Kushilevitz, Privacy and Communication Complexity, FOCS 89, and *SIAM J. Discrete Math*., Vol. 5, No. 2, 1992, pp. 273–284.

[KK] D. Karger and D. Koller, (De)randomized Construction of Small Sample Spaces in NC, FOCS 94 and *J. Comput*. *System Sci*., Vol. 55, No. 3, 1997, pp. 402–413.

[KM1] D. Koller and N. Megiddo, Constructing Small Sample Spaces Satisfying Given Constraints, STOC 93 and *SIAM J. Discrete Math*., Vol. 7, No. 2, 1994, pp. 260–274.

[KM2] D. Karger and R. Motwani, Derandomization through Approximation: An NC Algorithm for Minimum Cuts, *Proc*. 26*th STOC*, 1994, pp. 497–506.

[KM3] H. Karloff and Y. Mansour, On Construction of k-wise Independent Random Variables, STOC 94 and *Combinatorica*, Vol. 17, No. 1, 1997, pp. 91–107.

[KM4] E. Kushilevitz and Y. Mansour, Randomness in Private Computations, PODC 1996 and *SIAM J. Discrete Math*., Vol. 10, No. 4, 1997, pp. 647–661.

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

[KOR1] E. Kushilevitz, R. Ostrovsky, and A. Rosén, Characterizing Linear Size Circuits in Terms of Privacy, STOC 96 and *J. Comput*. *System Sci*., Vol. 58, 1999, pp. 129–136.

[KOR2] E. Kushilevitz, R. Ostrovsky, and A. Rosén, Amortizing Randomness in Private Multiparty Computations, *Proc*. 17*th PODC*, 1998, pp. 81–90.

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

[KR] E. Kushilevitz and A. Rosén, A Randomness-Rounds Tradeoff in Private Computation, CRYPTO 94 and *SIAM J. Discrete Math*., Vol. 11, No. 1, 1998, pp. 61–80.

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

[MS] F. J. Macwiliams and N. J. A. Sloane, *The Theory of Error Correcting Codes*, North-Holland, Amsterdam, 1977.

[MR] S. Micali and P. Rogaway, Secure Computation, manuscript, 1992 (updated version, 1998). Preliminary version in *Proc*. *CRYPTO* 91, pp. 392–404.

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

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

[RB] T. Rabin and M. Ben-Or, Verifiable Secret Sharing and Multiparty Protocols with Honest Majority, *Proc*. 21*st STOC*, 1989, pp. 73–85.

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

[S1] A. Shamir, How to Share a Secret, *Comm*. *ACM*, Vol. 22, 1979, pp. 612–613.

[S2] L. J. Schulman, Sample Spaces Uniform on Neighborhoods, *Proc*. 24*th STOC*, 1992, pp. 17–25.

[S3] M. Sudan, Algorithmic Issues in Coding Theory, *Proc*. 17*th Conf. on Foundations of Software Technology and Theoretical Computer Science*, Kharapur, India, 1997. Available on-line at theory.lcs.mit.edu/˜madhu/

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

[Y1] A. C. Yao, Theory and Applications of Trapdoor Functions, *Proc*. 23*rd FOCS*, 1982, pp. 80–91.

[Y2] A. Yao, Protocols for Secure Computation, *Proc*. 23*rd FOCS*, 1982, pp. 160–164.

[Z] D. Zuckerman, Simulating BPP Using a General Weak Random Source, FOCS 91 and *Algorithmica*, Vol. 16, 1996, pp. 367–391.