Randomness vs. Fault-Tolerance

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

Abstract

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

Previously, the upper bound on the amount of randomness needed for securely computing any non-trivial function f was polynomial both in n, the total number of parties, and the circuit-size $C(f)$. This was the state of knowledge even for the special case $t = 1$ (i.e., when there is at most one malicious player). In this paper, we show that for any linear-size circuit, and for any value $t < n/2$, $O(poly(t) \cdot \log n)$ randomness is sufficient. More generally, we show that for any function f with circuit-size $C(f)$, we need only $O\left(poly(t) \cdot \log n + poly(t) \cdot \frac{C(f)}{n}\right)$ randomness in order to withstand any coalition of size at most t. Moreover, in our protocol only $t+1$ players flip coins and the rest of the players are deterministic. Our results generalize to the case of *adaptive* adversaries as well.

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

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

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

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

1 Introduction

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

SECURE PROTOCOLS. Secure multiparty protocols (first studied in [51, 27]) are protocols that guarantee the privacy of the inputs and 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 four basic settings were considered. The adversary controlling the corrupted (i.e., faulty) parties can be either computationally unbounded (in which case the communication channels are assumed to be private) [6, 15] or limited to efficient (probabilistic polynomial time) computations [51, 27]. 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 faulty 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.

We mention some known results: In [51, 27] it was shown that under intractability assumptions every poly-time computable function f can be computed securely tolerating a computationally bounded, active adversary that controls up to $t < n/2$ parties. Moreover, in the case of passive adversaries, any number $t \leq n$ of colluding parties is tolerable. In [6, 15] protocols for securely computing any function in the presence of computationally unbounded adversaries are shown. In the case of passive adversaries these protocols withstand up to $t < n/2$ corrupted parties. In the case of active adversaries these protocols withstand up to $t < n/3$ corrupted parties. In both cases this is the maximum attainable resilience. Considerable amount of work has been done in this area since (e.g., [2, 3, 7, 8, 12, 19, 21, 25, 38, 39, 40, 41, 42]). In the sequel we concentrate on works concerning the relation between multiparty security and randomness.

RANDOMNESS. Randomness plays an important role in computer science. In particular, in the context of distributed computing there are important examples of problems where there is a provable gap between the power of randomized algorithms and their deterministic counterparts. For instance, achieving Byzantine agreement with linear number of faults requires linear number of rounds deterministically [23] and constant number of rounds if randomization is allowed [22]; reaching a consensus in an asynchronous distributed system with faults is impossible with deterministic protocols [24], but is possible with the use of randomized protocols (see [16]). Various techniques to minimize the amount of randomness needed were extensively studied in computer science (e.g., [35, 50, 10, 45, 52, 17, 31, 4, 44, 1, 48, 34, 36, 39, 32, 33]) and tradeoffs between randomness and other resources were found (e.g., [13, 46, 37, 14, 20, 9, 8, 42, 7, 41, 39]).

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

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

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 [39]. Additionally, for any function f with *sensitivity* n, if $t \geq n - c$ for some constant c, then $\Omega(n^2)$ random bits are required [7].
2. For the special case of 1-privacy any linear-size circuit can be computed with constant number of random bits [41].

Our results. We generalize both of the above results. That is, we show that for both passive and active adversaries, and for *any* value of t for which secure computation is possible, any circuit of m boolean gates can be securely evaluated using only $O(poly(t) \cdot \log n \cdot \frac{m}{n})$ random bits overall. While these results do not substantially improve on [6, 15] for $t = \Theta(n)$, they constitute big improvement for smaller values of t. In particular for $t = polylog(n)$, circuits with quasi-linear (i.e., $m = O(n \cdot polylog(n))$) number of gates can be securely computed using only $polylog(n)$ random bits. For $t = 1$ and $m = O(n)$ we are only $O(\log n)$ away from the specialized (to passive adversaries only) result of [41]. Our results generalize to *adaptive* adversaries as well (i.e., adversaries that choose the identity of the corrupted parties during the course of the computation, based on the information gathered so far [12, 11]).

[1] When the adversary is limited to probabilistic polynomial time and intractability assumptions are used, as in [51, 27], then by the results of [10, 29, 30] we may as well assume the existence of a pseudorandom generator. In this case players can expand "small" seeds of truly random bits into "long" sequences of pseudo-random bits and use them. Therefore, in this case the quantification of the "amount of randomness needed" is not meaningful.

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 some cases this may be the preferred solution, but the 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 $O(t) > t$ parties, and have the parties in S compute the function and announce the results. This "small decentralization" approach seems especially viable when $t = o(n)$, sine the set S need not be much bigger then t. Our work shows that, with respect to the amount of randomness used, this "small decentralization" extended centralized solution is considerably inferior to a full-blown distributed computation: in contrast to our results (we need only $O(\frac{m}{n} \cdot \log n \cdot poly(t))$ randomness), the above "small decentralization" solution according to presently known methods requires $O(m \cdot poly(t))$ random bits.

Techniques used and additional remarks. Our results build on many previous ideas in the area of privacy as well as on limited independence distributions. In particular, we use the general framework of [6], and combine it with ideas from [41] together with techniques for limited independence, in order to save in randomness. That is, the parties evaluate the given circuit gate by gate; each gate is computed in a manner similar to the construction of [6]. However, as in [41], 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 [41] 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 works on both secure protocol design and de-randomization techniques.

Interestingly, we show that not only we can use a small amount of randomness but also only $t+1$ players need to be randomized, and the rest of the players can be deterministic. This is nearly optimal against coalitions of size t, since it was shown in [39] that t-private computations of simple functions require at least t players to use randomness, and that in some cases, such as the XOR function, t is sufficient.

Organization: In Section 2 we provide some necessary definitions, including those of privacy and randomness. In Section 3 we provide our solution for the case of passive adversaries. In Section 4 we extend it to the case of active (i.e., Byzantine) adversaries. In the Appendix we review, for self containment, the construction of [6].

2 Preliminaries

Secure Protocols. We specify the requirements from a protocol for securely computing a function whose inputs are partitioned among several parties. Some definitions of multiparty secure computation have been proposed in the past [43, 28, 3, 5, 11, 12]. Here we outline the [11, 12] approach and definition. We concentrate on the 'information-theoretic' case; that is, the adversary is computationally unbounded. We formulate the definition for the case of *active* adversaries. The definition for *passive* adversaries can be inferred in a straightforward way. We also consider *adaptive* adversaries.

Figuratively, secure protocols "emulate" an ideal setting where all parties privately hand their inputs to a centralized trusted party who computes the results, hands them back to the parties, and vanishes. First the 'real-life' computation is modeled as follows. The parties interact via their private communication links; an adversary *corrupts* parties, during the course of the computation based on the information seen so far. Once a party is corrupted it hands all the information seen in the past to the adversary. From this point on it follows the adversary's instructions. (If the adversary is passive then the party follows its original protocol.) We call this adversary a *real-life adversary.* At the end of the computation each uncorrupted party locally outputs whatever is specified in its protocol; only the output of corrupted parties is known to the adversary.

We use the following notation. Let $\text{VIEW}_{\pi,\mathcal{A}}(\vec{x},\vec{r})$ denote the view of adversary $\mathcal{A}$ when interacting with parties running protocol π on input $\vec{x}$ and random input $\vec{r}$ (x_i and r_i for party P_i; the adversary has also a random input r_{n+1}), as described above. Let $\text{VIEW}_{\pi,\mathcal{A}}(\vec{x})$ denote the random variable describing the distribution of $\text{VIEW}_{\pi,\mathcal{A}}(\vec{x},\vec{r})$ when $\vec{r}$ is randomly chosen. Let $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x},\vec{r})_i$ denote the output of party P_i after running protocol π on input $\vec{x}$ and random input $\vec{r}$, and with a real life adversary $\mathcal{A}$. If P_i is uncorrupted this is the output specified by the protocol; if P_i is corrupted then $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x},\vec{r})_i = \text{VIEW}_{\pi,\mathcal{A}}(\vec{x},\vec{r})$. Let $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x},\vec{r}) = \text{EXEC}_{\pi,\mathcal{A}}(\vec{x},\vec{r})_1 \ldots \text{EXEC}_{\pi,\mathcal{A}}(\vec{x},\vec{r})_n$. Let $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x})$ denote the random variable describing $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x},\vec{r})$ where $\vec{r}$ is uniformly chosen.

Next an *ideal model* for multiparty function evaluation is formulated. This ideal model captures "the highest level of security we can hope to get from such evaluation". A computation in this ideal model proceeds roughly as follows. First an *ideal-model adversary* corrupts a set of parties, one by one in an adaptive way[2]. Once a party is corrupted the adversary learns the party's input; if the adversary is active then it may also modify the party's input. Next all parties hand their (possibly modified) inputs to an incorruptible *trusted party.* The trusted party computes the outputs (i.e., it evaluates the given function at the given inputs) and hands each party its designated output. Finally, the uncorrupted parties output whatever they receive from the trusted party; the corrupted parties output some function of their joint identities, inputs, values received from the trusted party, and the random input of the adversary. We let the n-vector $\text{IDEAL}_{f,\mathcal{S}}(\vec{x},r) = \text{IDEAL}_{f,\mathcal{S}}(\vec{x},r)_1 \ldots \text{IDEAL}_{f,\mathcal{S}}(\vec{x},r)_n$ denote the outputs of the parties for computing f on input $\vec{x}$, adversary $\mathcal{S}$, and random input (to $\mathcal{S}$), r (party P_i outputs $\text{IDEAL}_{f,\mathcal{S}}(\vec{x},r)_i$). $\text{IDEAL}_{f,\mathcal{S}}(\vec{x})$ is the random variable describing $\text{IDEAL}_{f,\mathcal{S}}(\vec{x},r)$ where r is uniformly chosen.

Finally a protocol π for computing a function is secure if executing π in the 'real-life' setting is "equivalent" to evaluating the function in the ideal model, in the sense that any effect on the computation achieved by a real-life adversary can already be achieved by an ideal-model adversary. More precisely, an adversary is called *t-limited* if it corrupts no more than t parties at each execution. A protocol π for computing function f is t-secure if for *any* t-limited real-life adversary $\mathcal{A}$ attacking π there *exists* a t-limited ideal-model adversary $\mathcal{S}$ that has the same effect on the computation (of f) as $\mathcal{A}$, *even though $\mathcal{S}$ operates in the ideal model.* The effect of an adversary on a computation is defined as follows. On any inputs for the parties, the random variable describing the outputs of all parties in the ideal model with adversary $\mathcal{S}$ should be distributed identically to the random variable describing the outputs of all parties in the real-life model with adversary $\mathcal{A}$. That is:

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

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

where $\stackrel{d}{=}$ denotes "identically distributed". If only passive adversaries are considered then we say that π t-privately computes f.

Note that this definition captures the intuitive notion of "not being able to learn any information from the communication" since the output of the corrupted parties includes the communication that the corrupted parties see.

Black-box simulation. To show that secure protocols can be 'composed' (i.e., if one combines secure protocols in an appropriate manner then the resulting protocol is also secure), a more restricted notion of equivalence of computations is used. Specifically, the ideal-model adversary is limited to black-box simulation of the real-life adversary. That is, there should exist a *single* ideal-model adversary $\mathcal{S}$ with oracle (or black-box) access to a real-life adversary. (This black-box represents the input-output relations of the real-life adversary described above.) The ideal-model adversary $\mathcal{S}$ will henceforth be called a simulator. We let $\mathcal{S}^{\mathcal{A}}$ denote the simulator $\mathcal{S}$ with black-box access to a real-life adversary $\mathcal{A}$. For concreteness, we describe the 'mechanics' of this black box. It has a random tape, and an input-output tape. Once a special `start` input is given on the input-output tape, the interaction on this tape proceeds in iterations, as follows. Initially, no party is corrupted. In each iteration l, the black-box expects to receive the messages sent by the uncorrupted parties to the corrupted parties in the lth round. Next, the black-box outputs the messages to be sent by the corrupted parties in the lth round. (If the adversary is passive then these messages are specified by the protocol; otherwise they are arbitrary.) Next, the black-box may issue several '`corrupt` P_i' requests. Such a request should be answered by P_i's view of the computation so far (i.e., P_i's input, random input, and messages received and sent). Also, from this point on P_i is corrupted.

The simulator is restricted to probabilistic polynomial time (where each invocation of the black-box is counted as one operation)[3]. Furthermore, we limit the operation of the

[2] We remark that the adaptivity in the ideal model is only in the sense that the adversary can choose who to corrupt next based on what he gets from the previously corrupted players. This is *weaker* than the adaptive real-life adversary who also sees partial history of the computation.

[3] For simplicity, we assume that the computed function is polynomially computable. Alternatively, the simulator is polynomial in the complexity of the function.

simulator as follows. We require that the `start` message is sent only once, and that no party is corrupted in the ideal model unless a request to corrupt this party is issued by the black-box. We call protocols that satisfy this stronger notion of security *simulatable.* We stress that our protocols (as well as those of [6]) are simulatable.

Measuring randomness. We measure the amount of randomness used by a protocol as follows. We provide each player P_i with a random tape (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 player P_i reads. In this case we say that player P_i used $d_i \cdot \lceil \log p \rceil$ random bits.[4]

Definition 2: *A d-random protocol is a protocol such that for any input assignment, the total number of random bits used by all players in* any *execution is at most d.*

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

Circuits. In the sequel we represent the functions computed by the parties as circuits. These circuits have as input n bits $x_1, \ldots, x_n$, and their negations $\bar{x}_1, \ldots, \bar{x}_n$. The circuits consist of AND and OR gates of fan-in 2, and arbitrary fan-out. The size of a circuit, denoted m, is the number of gates in the circuit.

3 The Protocol for Passive Adversaries

We prove the following theorem:

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

We first describe our protocols assuming the existence of a trusted dealer whose role is restricted to tossing coins, and distributing (random) values to the players. The trusted dealer does not receive any messages and has no input. Later, in Section 3.2.1, we eliminate this assumption by using $t+1$ players who will perform the task of the trusted dealer (in addition to their other tasks).

3.1 Overview

All known generic constructions of protocols for securely computing any function have the following structure. First each party shares its input; next the parties evaluate the given circuit in a gate-by-gate manner from inputs to outputs, maintaining the property that the value of each wire in the circuit is shared among the parties. Finally the parties reconstruct the value of the output wire from their shares. Our approach is general and can be applied to any protocol that follows this outline. For concreteness, however, we concentrate on the [6] construction (briefly reviewed in Appendix A).

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 players 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 [47, 6] 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 players in the corresponding team. Each team will be assigned to roughly $m/(n/s) = m \cdot s/n$ gates.

To evaluate a gate, each player of the corresponding team T first receives a share of the value of each of the two input wires to the gate. These shares are communicated by the players of the teams that evaluated the gates leading to those wires. Now team T invokes the [6] procedure for evaluating this gate. (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 value of the output wire of the circuit is known, the corresponding team reconstructs this value and announces it to all other parties.

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

3.2 Detailed Description

We now state the protocol in detail, assuming a trusted dealer. Let $s = 2t+1$ and let $n = k \cdot s$ (we assume for convenience that n is divisible by s; see Remark 1 below). We partition the parties into k teams of players of size s each. Each team will evaluate (at most) $\ell = \lceil \frac{m}{k} \rceil$ gates. We also specify an enumeration of the parties in each team. Denote by $P_{T,j}$ the j'th player in team T. Let $p > s$ be a prime (to be determined later). All the computations described below are over $GF[p]$. Since we want to evaluate a boolean circuit, we will use the transformation a AND $b \Rightarrow a \cdot b$ and a OR $b \Rightarrow 1 - ((1-a)(1-b))$. That is, we now have an arithmetic circuit with addition and multiplication gates.

1. (Input Sharing)
   Each player P_i invokes the [6] input sharing procedure (Shamir's secret sharing scheme) with respect to the parties in its team. That is, at the end of this stage there exists a (random) polynomial Q_i over $GF[p]$ and each member P_j in P_i's team has its share $s_i^j = Q_i(j)$. Note also that for this procedure P_i needs t random elements in $GF[p]$.

2. (Computation)
   The gates of the circuit are evaluated one by one from

[4] 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.

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:*
Let x and y be the input wires of gate g, and let T_x and T_y denote the teams that evaluated the corresponding gates. 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 input wire to gate g the parties in T hold shares of a polynomial of degree t whose free coefficient is the value of that wire.

- *Compute shares for the output of the gate:*
Once the parties in T receive their shares of the input wires, they evaluate the gate by invoking the appropriate procedure of [6] (depending on the type of the gate). At the end of this step the parties in T hold shares of a polynomial of degree t (over $GF[p]$) whose free coefficient is the value of the gate. We observe that the randomness needed in this step is at most t values in $GF[p]$ per player in T (see Appendix).

3. (Output)
Let T be the team that computes the output gate of the circuit. Then the parties in T invoke the [6] output reconstruction procedure. Recall that this procedure consists of broadcasting the shares of the output wire and interpolating the shared value. Finally $P_{T,1}$ announces the result to all n parties.

We now turn to describe how the dealer distributes the random numbers (in $GF[p]$) to the players. In the Input Sharing stage the dealer distributes coefficients of n polynomials (one polynomial to each player). Then, the dealer distributes additional s polynomials per each gate to be computed (one polynomial for each player in the simulating team). Each polynomial is defined by t coefficients in $GF[p]$. Therefore, the dealer produces 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 produce these M numbers independently. Instead, we observe that the view of each subset (of size at most t) of players depends on a "relatively small" set of numbers: at most $\beta = \Theta((m/n)\cdot t^4)$ such numbers. In brief, note that the total number of shares that any set of t players sees is $t\cdot(s+\ell\cdot(2s+s)) = O(t\cdot\ell\cdot s)$, thus depending on at most $O(t^2\cdot\ell\cdot s)$ numbers that the dealer distributes as coefficients. In addition, a set of players of size t receives directly from the dealer a total of $O(t^2\ell)$ numbers.[5] Hence the dealer can produce M numbers $Z_1, Z_2, \ldots, Z_M$ in $GF[p]$ which are uniformly distributed and are $\beta = \Theta((m/n)\cdot t^4)$-wise independent. This may be done as follows. Set $p > M$. The dealer chooses a random polynomial $R(x)$ in $GF[p]$ of degree β and then produces 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 produced by the dealer. The view of any subset of size t of players indeed depends on at most $\beta = \Theta((m/n)\cdot t^4)$ numbers produced by the dealer. However, we do not need that *any* subset of size β be uniformly distributed. It suffices that the $\binom{n}{t}$ subsets of size β, defined by the $\binom{n}{t}$ subsets of t players, be uniformly distributed. To achieve the relaxed requirement we use an extension of the results of [48, 39]: Set $p > 2t+1$. The dealer will uniformly sample a space of M-tuples over $GF[p]$, which is constructed to suit the specific $\binom{n}{t}$ subsets. By [48, 39] there is a sample space of size $\binom{n}{t}p^\beta$, such that if we sample it uniformly, then the projection of the chosen vector on any of the $\binom{n}{t}$ subsets is uniformly distributed.[6] To sample this space the dealer will need randomness of $O(t\log n + (m/n)t^4\log p)$.

We now outline the analysis of the above protocol (still under the assumption of a trusted dealer). Full details are omitted from this abstract. In the proof we use the fact that the [6] protocol is secure. In fact, we use a stronger property: each of the procedures of [6] (i.e., the input sharing procedure, the addition and multiplication procedures, and the output reconstruction procedure) is separately t-private, when applied with a set of $s = 2t+1$ players. For instance, consider the (probabilistic) function f that describes the sharing of P_i's input. This function depends only on P_i's input (the other parties have empty inputs). The parties' (secret) outputs of f are shares of a random polynomial of degree t, with free coefficient equal to P_i's input. Then the [6] protocol t-securely computes this function, in a simulatable way. Similar statements can be made with respect to the other components of [6].

Proof of Theorem 1 [Trusted dealer case] (outline). We show that the above protocol satisfies the conditions of Theorem 1. Let $t < n/2$. As explained, the protocol (in its trusted dealer version) is $O(t\log n + (m/n)t^4\log t)$-random.

To show t-privacy, we describe an ideal-model adversary $\mathcal{S}$. That is, a *simulator* having black-box access to an arbitrary real-life adversary $\mathcal{A}$. The simulator $\mathcal{S}^{\mathcal{A}}$ is interacting with a trusted party for computing f as described in Section 2 and for each input $\vec{x}$ for the parties, the parties' outputs as generated by $\mathcal{S}^{\mathcal{A}}$ is identically distributed as the parties' outputs in real-life, interacting with $\mathcal{A}$.

The simulator $\mathcal{S}^{\mathcal{A}}$ uses as subroutines the simulators guaranteed to exist for the different [6] procedures invoked in our protocol. (Namely, the simulators for the input sharing procedure, for the addition and multiplication procedures, and for the output reconstruction procedure.) Each one of these *sub-simulators* guarantees the following property: As long as the random inputs of the parties that participate in the corresponding procedure (in the real-life execution) are independent, and assuming that the sub-simulator receives the information it expects in its ideal model, the parties' output of the simulated execution are identically distributed as the parties' outputs of the real-life execution with adversary $\mathcal{A}$.

Basically, simulator $\mathcal{S}$ invokes the appropriate sub-simulators as dictated by the structure of the evaluated circuit. Note that the different invocations are *not* independent (e.g., the set of corrupted parties over *all* sub-simulations contains at

[5] To get β, we use $s = \Theta(t)$, and $\ell = \Theta(m/k) = \Theta(m\cdot t/n)$.

[6] Both [48, 39] deal with the field $GF[2]$ but can be extended to $GF[p]$. We note that the time-complexity of sampling in the sample space is $poly(n, \log\binom{n}{t}, \beta)$ but the complexity of the known algorithms that find such a space is $poly(\binom{n}{t})$. Note however that this can be done "off-line" and can be hard-wired into the protocol.

most t parties). Therefore, the main issue of the proof is to show that $\mathcal{S}$ can indeed invoke the sub-simulators and provide them with the (consistent) information that they need (particularly, the answers that they get from the corresponding trusted parties). Once the simulation ends, $\mathcal{S}$ outputs the entire interaction with $\mathcal{A}$. That is, $\mathcal{S}$ outputs $\mathcal{A}$'s view of the simulated execution.

The crux of the proof is the observation that in the real-life execution the adversary's view depends only on a small number of random elements in $GF(p)$; thus, although the random inputs of the parties have only limited independence, the adversary's view is distributed identically to a situation where the random inputs are totally independent. Consequently, it is distributed identically to its view in the simulated execution (i.e., in the interaction with $\mathcal{S}$). It follows that the parties' outputs in the ideal scenario is distributed identically to their outputs in the real-life execution. (The uncorrupted parties output the function value and the corrupted parties output their view.)

More precisely, $\mathcal{S}$ first hits $\mathcal{A}$'s `start` button. Next it proceeds as follows:

- Whenever the real-life adversary $\mathcal{A}$ issues a "`corrupt` P_i" command (during the invocation of any of the sub-simulators), the ideal-model adversary $\mathcal{S}$ also corrupts P_i (in the ideal model) and receives P_i's input, x_i. Then, $\mathcal{S}$ hands $\mathcal{A}$ a possible view of P_i in the (real-life) computation so far, which is consistent with x_i and the information known to $\mathcal{A}$ so far. (We omit details from this outline.) Next $\mathcal{A}$ issues a "`corrupt` P_i" command to all the sub-simulators that are currently running (and all the sub-simulators that will be invoked later). Whenever any sub-simulator wishes to corrupt P_i in its ideal model, $\mathcal{A}$ hands back x_i.
- For each party P_i, simulator $\mathcal{S}$ invokes a sub-simulator, $\mathcal{S}_{I,i}$, for the [6] input sharing procedure of P_i's input among the parties in P_i's team. Any message sent by $\mathcal{S}_{I,i}$ to its simulated real-life adversary is being forwarded by $\mathcal{S}$ to $\mathcal{A}$. Each message issued by $\mathcal{A}$ in name of a corrupted party is being forwarded by $\mathcal{S}$ to the appropriate sub-simulator.
- When $\mathcal{S}_{I,i}$ asks its trusted party for the output values (of parties in P_i's team) of the input sharing function, it is answered (by $\mathcal{S}$) as follows: If P_i is corrupted then these values are computable from $\mathcal{A}$'s messages. Otherwise these values are picked at random (in $GF[p]$).
- When $\mathcal{S}_{I,i}$ terminates, $\mathcal{S}$ simulates the "baton hand-off" step of the shares of P_i's input. That is, let T be a team that is supposed to receive the shares of P_i's input wire. Then, for each corrupted party P_j in T, simulator $\mathcal{S}$ hands $\mathcal{A}$ the appropriate share: If the share is to be sent from a corrupted party P_i, then this share is already known to $\mathcal{A}$. Otherwise $\mathcal{S}$ hands $\mathcal{A}$ a random value in $GF[p]$.
- For each gate, $\mathcal{S}$ invokes a sub-simulator for the [6] corresponding procedure. This sub-simulator is being handled by $\mathcal{S}$ similarly to the sub-simulators for the input sharing procedure. In particular, the "baton hand-off" step between each two gates is being handled similarly to the "baton hand-off" step described above.
- When the simulation of the output gate of the circuit is complete, $\mathcal{S}$ invokes the sub-simulator $\mathcal{S}_O$ for the output reconstruction procedure. $\mathcal{S}_O$ is given the identities of the corrupted parties; their inputs to the simulated output reconstruction procedure are set to be their outputs of the simulated evaluation of the output gate. When $\mathcal{S}_O$ asks its trusted party for the output value, $\mathcal{S}$ proceeds as follows. First $\mathcal{S}$ hands its trusted party (in the ideal model) the inputs of the corrupted parties. In our case $\mathcal{A}$ is a passive adversary, therefore these inputs are just the unchanged x_i values. Now $\mathcal{S}$ receives the function value, y, from its trusted party, and forwards this value to $\mathcal{S}_O$. Then, $\mathcal{S}_O$ can complete its simulation.

This completes the (outline) description of the operation of $\mathcal{S}$. Since the sub-simulators for the various [6] components run in polynomial time, so does $\mathcal{S}$. The validity of $\mathcal{S}$ is shown via the following two claims. Denote the above protocol (in which the dealer uses a small probability space with limited independence) by π, and let π_R denote that protocol when the parties are given totally independent random inputs.

Claim 1: *Assume that in the above protocol (for computing function f) the parties are given totally random inputs. Then, the protocol satisfies Definition 1 with respect to simulator $\mathcal{S}$. That is, for any t-limited adversary $\mathcal{A}$ and any input $\vec{x}$ we have* $\text{IDEAL}_{f,\mathcal{S}^{\mathcal{A}}}(\vec{x}) \stackrel{d}{=} \text{EXEC}_{\pi_R,\mathcal{A}}(\vec{x})$, *where π_R is our protocol with a trusted dealer that deals random inputs to the parties uniformly and independently. (Proof omitted.)*

Claim 2: *Let π be the protocol we described, and π_R the same protocol with the exception that the trusted dealer deals all the numbers uniformly and independently. Then, for any input $\vec{x}$, the vector of output of the parties in π and π_R is identically distributed. That is, for any t-limited (real life) adversary $\mathcal{A}$ and any input $\vec{x}$ for the parties, we have* $\text{EXEC}_{\pi_R,\mathcal{A}}(\vec{x}) \stackrel{d}{=} \text{EXEC}_{\pi,\mathcal{A}}(\vec{x})$.

Proof: To prove the claim, we first observe the structure of the information of a particular player P_i. This information (in addition of the player's input x_i) consists of:

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

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

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

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

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

Altogether, P_i receives a total of $s + \ell(s+2)$ shares. For simplicity of the proof, we will assume that when a player sends the result of the computation of a gate, it does not send the sum of the shares as defined by the protocol, but instead sends the list of summands separately (in case of an OR gate this also include a constant). There are s such summands.[7] Therefore, in this modified setting each player receives at most $s + \ell(2s+s) = O(\ell s)$ shares. Clearly, if this modified protocol is secure, so is the original one.

Next we examine the messages that an arbitrary subset (coalition) of players of size at most t can see. Each of the messages of type 1 received by these players is just a different number Z_k in the space produced by the dealer. Altogether, the messages of type 1 seen by the players 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 player, say, P_j (this is not necessarily the player that sends the message; the player that sends the message may only relay on it). For a player P_j we denote by $\delta_{j,d}$ the d'th polynomial that this player 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 or players under consideration, these numbers are also distinct (by definition) from any of the Z_k's directly received by any corrupted player.

The total number of shares a subset of at most t players 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 players sees directly from the dealer (as messages of type 1), we have that the communication seen by the set of players 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 players depends on at most $O((m/n)t^4)$ values Z_k. To produce 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 players depends on, is uniformly distributed.

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

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

This completes the proof-outline of Theorem 1. □

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 players share their inputs among the players of the first team and then these r players do not further participate in the protocol. When one of these inputs is required then the first team will provide the corresponding shares. This implies that the view of players in the first team contains slightly more messages and hence requires slightly increasing the value of β (by a constant factor).*

3.2.1 The Protocol without Trusted Dealer

The above description assumes a trusted dealer whose role is restricted to choosing random integers in $GF[p]$ and distributing them to the players. To eliminate this assumption, we designate $t+1$ players who, in addition to their others roles in the protocol, will play the role of the trusted dealer. To do that, each one of these $t+1$ players will produce $M = n \cdot t + m \cdot s \cdot t$ integers in $GF[p]$ as described for the single dealer. Each of these players also distributes the numbers to the n players as described for the trusted dealer. Each player adds the $t+1$ corresponding numbers (over $GF[p]$), and considers this sum as the number provided by the "trusted dealer", for use in the protocol. Note that the amount of randomness used by this protocol is larger by a factor of $t+1$ than the amount used by the protocol with the trusted dealer.

We now outline the proof of security. We build a simulator for this protocol based on the simulator for the case of a trusted dealer. This simulator works as follows: It first

[7] Alternatively, the protocol can be defined in a way that the evaluation of each gates ends after each of the players in the team computes its local value, and gets t numbers from dealer. The players do not distribute shares of these values to the players in their team. When the value "computed" is needed by another gate, then the players distribute the shares to the players that need this value, which in turn, make the summation of the shares (and possibly subtract it from the constant 1). However, this description is less modular.

draws $t+1$ vectors of numbers over $GF[p]$, in the same way as is done by the $t+1$ dealers. Then it invokes the simulator for the case of a trusted dealer, with the following two differences: (1) If the simulator has to corrupt one of the dealers, then the information passed to the adversary includes also the corresponding drawn vector. In this case this vector is also added to the simulation's output. (2) When the simulator has to give a random number to one of the sub-simulators, it gives it the sum of the corresponding values in the $t+1$ vectors, in the same way that the players do it in the protocol. Now observe that since the adversary can corrupt at most t players, the random inputs to the sub-simulator have the distribution of the numbers in one of the vectors, and using the same arguments as in the proof of Claim 2, the random inputs to the sub-simulators can be considered uniformly distributed and independent. Therefore the simulator will produce an output with the same distribution as for the case of a trusted dealer and random inputs which are totally independent (except for the additional vectors of numbers).

When the real-life adversary interacts with the actual protocol with no trusted dealer the same arguments show that its output will be distributed the same way as if the random numbers used by the players were provided by an outside source of independent numbers. This is true except for the additional random vectors of numbers each dealer draws. The distribution of those is the same as the distribution of the vectors drawn by the simulator (as the simulator uses exactly the same procedure). Therefore the simulator can indeed simulate the real-life adversary.

4 Byzantine Adversaries

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

Theorem 2: *Let $t < n/3$. Then, any function $f : \{0,1\}^n \to \{0,1\}$ that has a circuit of size m, can be t-securely computed by a $O(t^3 \log n + (m/n)t^6 \log t)$-random protocol.*

The protocol for active adversaries is identical to the one for passive adversaries, with the exceptions that the size of teams is increased to $s = 3t+1$, and the various components of the [6] protocol are replaced by their Byzantine counterparts. That is:

- The secret sharing procedure is replaced by a *verifiable secret sharing (VSS)*. VSS schemes are described, for instance, in [18, 27, 6, 22, 15]. We sketch a VSS scheme in Appendix A.
- The multiplication procedure is replaced by its Byzantine counterpart, and the addition procedure remains (trivial and) unchanged.
- The output reconstruction procedure is replaced by the reconstruction procedure of the VSS scheme in use.

In addition, the trusted dealer now has to supply the parties with enough random values to support the VSS scheme and the evaluation scheme of [6]. Inspecting these schemes (see Appendix A), it turns out that each invocation of VSS requires from the dealer $O(t^2)$ values in $GF[p]$, and evaluating a gate requires $O(t^2)$ values in $GF[p]$ for each party in the team. Furthermore, the adversary's view of the computation now depends on at most $\beta = O((m/n) \cdot t^5)$ elements in $GF(p)$. Thus the dealer can still use the same method used for passive adversaries, with the appropriate value of β.

Emulating the trusted dealer is done in the same way as in the basic protocol.

Analyzing the Byzantine protocol. The analysis follows the same lines as the proof of Theorem 1, with the exception that the simulator $\mathcal{S}$ now invokes the sub-simulators for the Byzantine versions of the [6] components.

Yet, the introduction of Byzantine adversaries raises an additional apparent difficulty that is not addressed above. That is, in the extended protocol, when an uncorrupted party $P_{T_i,j}$ in team T_i receives a share of a value a from a *corrupted* party $P_{T_u,j}$ in team T_u, it may well be the case that $P_{T_i,j}$ will receive a bad share (or perhaps no share at all). If too many uncorrupted parties in T_i will start off the computation with wrong shares then the evaluation will be incorrect. This difficulty is answered as follows. Since there are at most t corrupted players altogether, and each corrupted player can give a bad share to at most one party in T_i, it follows that at most t players in T_i are either corrupted or start off with an erroneous share. Thus we are guaranteed that the corresponding sub-simulator will meet its requirements.

Acknowledgments We thank Oded Goldreich for his insightful comments which greatly improved the presentation of our paper, as well as for suggesting us a way to get rid of some more unnecessary randomness by using [48, 39].

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[22] P. Feldman and S. Micali, "An Optimal Algorithm For Synchronous Byzantine Agreement", STOC, 1988, pp. 148-161.

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

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

[25] M. Franklin, and M. Yung, "Communication Complexity of Secure Computation", STOC, pp. 699–710, 1992.

[26] R. Gennaro, T. Rabin and M. Rabin, "VSS and Secure Multiparty Computations Made Simple", manuscript.

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

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

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

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

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

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

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

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

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

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

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

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

[39] E. Kushilevitz, and Y. Mansour, "Randomness in Private Computations", PODC 1996, pp. 181-190.

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

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

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

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

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

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

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

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

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

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

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

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

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

A An Overview of [6]

A protocol for honest but curious adversaries.

The protocol starts with a secret sharing stage in which each player uses Shamir's secret-sharing scheme to share its input among the players. Then, the players simulate an arithmetic circuit for the function in a gate-by-gate fashion. The evaluation of each gate starts with the players holding shares of the two inputs of the gate, a and b. That is, the shares corresponding to a are produced using a random polynomial $A(x)$ (over $GF[p]$) of degree t whose free coefficient is a. The share of each player P_i is the value $A(i)$. The value b is shared in the same manner using a polynomial $B(x)$. The goal of the players is to compute shares of the value c, which is the output of the gate, using a degree t polynomial $C(x)$ whose free coefficient is c (and the share of player P_i is $C(i)$).

If the gate computes the value $c = a + b$ then the desired polynomial $C(x)$ is just $A(x) + B(x)$. Hence, each player can locally compute (without any communication) the value $C(i) = A(i) + B(i)$. If the gate computes the value $c = a \cdot b$ some interaction between the players is required. Here we describe Rabin's simplification [26] of the technique of [6]. First, each player P_i locally computes the value $D(i) = A(i) \cdot B(i)$. These values define a polynomial $D(x)$ whose free coefficient is indeed the value $a \cdot b$. However, the degree of D is $2t$ (and not t) which may lead to problems in revealing the output at the end. In addition, D is not even a random polynomial of degree $2t$ (as D cannot be irreducible) which may lead to the violation of privacy (as shares that will be produced later may not be uniformly distributed). We overcome these problems as follows. We show below that there is a linear combination

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

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

It remains to show γ_i's that satisfy (1). 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 $1 \leq i,j \leq 2t+1$) contains the value i^{j-1}. Also denote, $\vec{D} = (D(1), D(2), \ldots, D(2t+1))$. With this notation we get that $\vec{D} = V \cdot \vec{d}$. Since V is non-singular (see, e.g., [49]), 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 $D(0) = d_0 = \sum_{i=1}^{2t+1} V^{-1}_{1,i} \cdot D(i)$ (where V^{-1} is a known matrix).

Dealing with Byzantine adversaries.

The general structure of the protocol remains the same, while the components are modified to withstand Byzantine adversaries. The components that need to be modified are two:

1. Sharing of a secret: Here a Verifiable Secret Sharing (VSS) scheme is used instead of Shamir's secret sharing. (VSS was introduced by [18].) Essentially, a VSS scheme makes sure that an honest dealer can successfully share a secret in a recoverable way, while guaranteeing that even if the dealer is corrupted, at the end of the sharing phase the uncorrupted parties hold shares of a well defined and reconstructible value. Several VSS schemes exist. We sketch one of the schemes described in [6], that withstands $t < n/3$ faults. The dealer, sharing a secret s, chooses a random bivariate polynomial H of degree t in each variable, whose free coefficient is s. (That is, $H(x,y) = \sum_{i,j=0}^{t} h_{i,j} x^i y^j$, where $h_{0,0} = s$ and the other coefficients are random.). Next the dealer sends the polynomials $f_i(\cdot) = H(i, \cdot)$ and $g_i(\cdot) = H(\cdot, i)$ to each P_i. Next each P_i sends $f_i(j)$ to each P_j, and verifies that the value received from P_j equals $g_i(j)$. (Note that $f_i(j) = H(i,j) = g_j(i)$.). If any of his verifications fails, the party requests the dealer to make the corresponding value (i.e., $H(i,j)$) public. Next, each party P inspects all publicized values. If any of these values doesn't match P's private shares, then P requests the dealer to make P's share public. The parties now again inspect the public shares. If a party P_i finds any inconsistency with its private share then it decides to abort this sharing. Otherwise it sets its share of the secret to be $f_0(i)$.

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

2. Evaluating a multiplication gate: In the honest but curious case evaluating a multiplication gate consisted of each party re-sharing a locally computed value, followed by local evaluation of a linear combination of the newly received shares. Here we follow this method, with two modifications.

First, each party re-shares the locally computed value using a VSS scheme. It should be noted that the local evaluation of the linear combination on the newly received shares can still be done, since the share of each party is still a value $f_0(i)$ of a random polynomial whose free coefficient is the secret.

Next the parties have to prove that the re-shared values are indeed the product of their shares of the input wires to the gate. This is done as follows. Note that all the values that were properly shared 'sit on a polynomial' of degree t. Thus the set of values shared by the parties can be regarded as a perturbed code-word of a Reed-Solomon code, where the erroneous entries correspond to the parties that shared incorrect values. Note that no party knows all these values. Still, the parties hold shares of these values. The parties use their shares to reconstruct the *syndrome* vector of this code-word. This syndrome, while revealing no information on the values that were honestly shared, identifies the parties that shared incorrect values. These shares are sieved out in the computation of the linear combination.