Secure Computation with Honest-Looking Parties: What if nobody is truly honest?

(Extended Abstract)

Ran Canetti* Rafail Ostrovsky†

April 28, 1999

Abstract

In a secure multi-party computation a set of mutually distrustful parties interact in order to evaluate a pre-defined function of their inputs, without revealing the inputs to each other. In this scenario, the trust in other parties should be minimal. In the classic formulation of this problem, most of the parties are trusted to exactly follow the prescribed protocol, except for a limited number of parties that are corrupted by a centralized adversary and are allowed to deviate from the protocol in an arbitrary way. However, an assumption of a totally honest behavior of most parties can not be verified. In particular, if an "honest-looking" party diverges from its protocol in a way that is indistinguishable from a totally honest player, it can do so with "impunity".

In this paper, we consider the situation where *all parties* (even uncorrupted ones) may deviate from their protocol in arbitrary ways, under the sole restriction that most of the parties do not risk being detected by other parties as deviating from the protocol execution.

The question whether secure protocols exist in this scenario was raised in the past, and solutions for very limited deviations from the protocol (i. e., refraining from erasing data) were given. Yet, solving the general problem was believed hard, if at all possible. Contrary to this belief, we show that if secure communication channels are provided (and one-way functions exist) then any polynomial function can be securely computed in this scenario.

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

†Bell Communications Research, MCC-1C365B, 445 South Street, Morristown, New Jersey 07960-6438, e-mail: rafail@bellcore.com

1 Introduction

The discipline of cryptographic protocols is, generally speaking, geared towards finding ways for mutually distrustful parties to perform some joint functionality in a "secure way". Here security usually means some combination of correctness requirements from the outputs, along with secrecy requirements on the local inputs. A classical formulation of the above problem, put forward by Yao, and by Goldreich, Micali and Wigderson [29, 17] proceeds as follows: There are some number, n, of parties, some of which are "honest" in the sense that they follow the prescribed protocol. The rest of the parties behave maliciously in an arbitrary, centrally controlled way. (Figuratively, they are controlled, or corrupted, by an adversary.) In this model it was shown by Goldreich, Micali and Wigderson [17], Ben-Or, Goldwasser and Wigderson [5], Chaum, Crepeau and Damgard [10] and Goldreich [16] how to securely perform any functionality of the parties' inputs when the adversary corrupts up to a constant fraction of the parties. Limited security can be maintained even if the majority of the parties are corrupted as was shown by Beaver and Goldwasser [4] and Goldwasser and Levin [19].

Is it reasonable to assume that (even uncorrupted) parties scrupulously follow their protocol? In a distributed environment where no party is thoroughly trusted it may sometimes be more reasonable to assume that *any party* would deviate from its protocol, if this deviation may carry some gains, and as long as the deviation is guaranteed to remain undetected by its peers. (This is, of-course, in addition to the corrupted parties who are not deterred by being uncovered and may run an arbitrary protocol.) Protocols that guarantee security even in this scenario are the focus of this paper.

Limited types of "externally undetectable" deviation from the prescribed protocol were studied in the past. (Slightly modifying a term from Canetti, Feige, Goldreich and Naor [7], we call parties that carry out such deviations honest-looking.) Refraining from erasing local data is an example of limited honest-looking behavior. Ben-Or, Goldreich and Wigderson [5] and Chaum, Crepeau and Damgard [10] general constructions remains secure even in the presence of such deviation, if secure channels are provided. In Canetti, Feige, Goldreich and Naor [7] it is shown how to maintain this property even when the security of the channels is obtained via encryption, and even when the adversary is *adaptive*, i.e., it corrupts parties during the course of the computation in an adaptive way. Other limited forms of honest-looking behavior, and methods for protecting against such behavior using a trusted dealer, were also discussed in Canetti at al. [7]. However, they argue that protecting against more permissive forms of honest-looking behavior is hard (if at all possible).

In this paper, we study such (more permissive) behavior. Instead of assuming that most of the players are totally honest and follow the prescribed protocol, we only assume that most of the players behave in a manner that guarantees that they will not be caught "cheating" (i.e., caught deviating from a prescribed protocol). We argue that this may be a more realistic assumption from a 'sociological' point of view, since detection that a processor "cheats" usually carries some negative stigma or punishment.

Our results: We consider the setting where secure channels among all parties exist and the adversary is computationally bounded. In this setting, we show how honest-looking behavior can be dealt with. We first define two types of honest-looking behavior. The first is similar to the most permissive notion of Canetti, Feige, Goldreich and Naor work [7]; the other is even more permissive. Next, we show how both types of honest-looking behavior can be dealt with.

Let us first describe the two forms of honest-looking behavior considered in this work. The more "conservative" behavior allows arbitrary "internal" deviation from the protocol, as long as the deviation is undetectable by the joint view of *all* parties running the protocol (as long as the party remains uncorrupted). Figuratively, such a party makes sure to remain undetected even if the entire community joins forces to decide whether it is honest. We call this type of behavior globally honest-looking (GHL). Indeed, GHL behavior becomes potentially harmful only in the presence of an *adaptive* adversary. Such an adversary may gather much more information from corrupting a party that kept additional or different data to that prescribed in the protocol.

An even more "bold" type of honest-looking behavior allows arbitrary deviation as long as the deviation is undetectable by any *single* party until the point of corruption. We call this type of behavior locally honest-looking (LHL). We remark that LHL parties may be harmful even if the adversary is non-adaptive.

Note that both GHL and LHL parties "do not collaborate", in the sense that parties should be unable to tell whether any of their peers is totally honest or not. In addition, honest-looking parties should remain "indistinguishable" from totally honest parties *with the same inputs.* This follows the approach that inputs are set externally and are beyond the control of the parties.

We define protocols that are robust to honest-looking behavior. These are protocols where the functionality of the honest parties (i.e., of the parties that follow the prescribed protocol) remains unchanged even when any number of other parties are only honest-looking. Next we show how any protocol can be transformed into a robust one. In particular, this means that the general constructions of Ben-Or, Goldreich and Wigderson [5] and Chaum, Crepeau and Damgard [10] and Rabin, Ben-Or [27] can be made robust to honest-looking behavior (both GHL and LHL).

Our constructions are far from practical and should be regarded only as a "proof of feasibility", of something conjectured impossible before. In particular, we assume that t, the maximum number of corrupted parties, satisfies $t < n^{\epsilon}$ where $\epsilon > 0$ is some fixed constant (that comes from our proof), and that the number of parties is polynomially related to the security parameter. (Still, it was conjectured in [7] that such a result is impossible.)

We complement our protocols by strengthening an example from Canetti, Feige, Goldreich and Naor [7] into a generic protocol that is not robust even to GHL parties, if claw-free pairs exist. This example can be extended to show that practically all known protocols for general multiparty computation, and in particular all protocols mentioned above, are insecure even against GHL parties (under the same assumption). (Thus, our complier gives the first protocol robust for both HGL and LHL behavior.)

The technical difficulties and our solution. The main difficulty with protecting against general honest-looking parties (even GHL ones) is that such parties may use their internal *randomness* in ways that make a proof of security impossible. For instance, an honest-looking party can "embed" a commitment to its input in any random string that it is instructed to send. While this behavior looks "harmless" (and is undetectable as long as the party remains uncorrupted), it has devastating effects for the proof of security. (In a nutshell, it becomes impossible to "simulate" the information learnt by the adversary when this party gets corrupted later.)

Our solution is based on the observation that this seemingly unavoidable behavior depends on the fact that the (honest) protocol instructs the party to use a large (i.e., polynomial in the security parameter) number of random bits. If the party were instructed to use only few random bits then this behavior would be prevented. In particular, the first step in our proof shows, basically, that

if a protocol is secure in the standard sense (and does not instruct parties to erase local data), and if each party is instructed to use only logarithmic number of random bits, then the protocol is also secure against GHL parties. We stress that this holds even if the honest-looking party tosses polynomially many coins.

Consequently, our transformations require each party to use only a small number of random bits. In fact, each party uses only a *single random bit*[1]. The random bits are then collected, over private channels, at some special parties who input the (polynomially many) collected bits to some standard pseudorandom number-generator, and then send long pseudorandom sequences back to all the parties. The parties now use these pseudorandom sequences as external "random" inputs for the original protocol. This simple idea, however, does not work as such. The problem, in essence, is that the adversary should not be able to distinguish between the case where the parties use real random strings for their random input for the original protocol, and the case where the random inputs come from our construction; this should hold even if the Adversary corrupts the parties that generated and expanded those strings. Solving this seemingly circular problem is one of the main technical issues addressed in this paper. In particular, the solution involves constructing a new evasive protocol for distributed pseudo-random number generation.

Dealing with LHL behavior is achieved by a reduction to GHL behavior. In LHL behavior, the parties only care about being detected by individual parties (rather than by the entire community). Here we use random, public "auditing" of parties to make sure that any deviation that can be detected by the entire community will actually become known to all parties (with some non-negligible probability). That is, first we make sure that each message sent during the protocol is "recorded" within sufficiently many parties. Next, at the end of each execution of the protocol, the parties jointly choose a single party and perform a public "audit" of this party. The audit uses (deterministic) Byzantine agreements in order to determine the messages sent and received by this party during the computation; once the messages are known the party's correct behavior is globally verified. Of course, additional care is taken not to reveal the private input of party that is being audited.

Related work. Canetti, Feige, Goldreich and Naor [7] use the term semi-honest behavior to denote general internal deviation from the protocol. Three types of semi-honest behavior are defined, all different from the ones here. The first type, called non-erasing, deviates by simply failing to erase data. The second type, called honest-looking, deviates arbitrarily as long as it remains undetected by *any external test* made by the entire community. The third type, called weakly honest, remains undetected by the entire community when the actual protocol is being run. In addition to a solution for non-erasing parties, they sketch a solution for honest-looking ones, given a trusted dealer in a pre-processing stage. GHL parties are similar to the weakly honest parties of Canetti at al. [7]. LHL parties allow even more extreme deviations from the protocol, not considered there at all.

Our protocols should be contrasted with the protocols that protect against a *dishonest majority* of corrupted parties of Beaver and Goldwasser [4] and Goldwasser and Levin [19]. There is a crucial difference in the problem formulations, (and the results achieved): we are not concerned with large coalitions of arbitrarily malicious parties. Instead, we show how to deal with a situation where in addition to a modestly sized totally dishonest coalition *all other parties* are only honest-looking.

[1] The fact that we can implement secure function evaluation where each player tosses only a single random bit might be of independent interest.

In particular, our results guarantee correctness of the outputs based on the true inputs of most of the parties, whereas Beaver, Goldwasser [4] and Goldwasser, Levin [4, 19] allow most of the parties to freely change their inputs. Moreover, our protocols deal with adaptive adversaries and do not allow "early stopping".

Our construction uses a wide variety of tools (and introduces several new ones.) In particular, we use pseudo-random generators of Hastad, Impagliazzo, Levin and Luby [21]; deterministic Byzantine Agreement of Garay and Moses [15]; arguments regarding limited randomness for general secure computation of Kushilevitz, Ostrovsky and Rosen [24] and Canetti, Kushilevitz Ostrovsky and Rosen [8]; the ways to recycle random bits in private computation of Kushilevitz and Rosen [22] and Kushilevitz Ostrovsky and Rosen [25]; the use of perfect hash functions of Fredman Komlos and Szemeredi [14] to boost resilience (also used by Fiat and Naor [13]); and the general completeness theorems of Ben-Or, Goldreich and Wigderson [5] and Chaum, Crepeau, and Damgard [10].

We stress that the security of the channels plays a crucial role in our solution. In fact, it is easy to see that in a setting where the adversary sees all communication (and private communication is achieved via probabilistic encryption [20]) each party must locally toss polynomially many coins for encryption. This makes our solution impossible since we use in a crucial way the fact that each party tosses only logarithmically many coins. (Alternatively, probabilistic encryption can be regarded as a separate module over which an honest-looking party has no control).

Organization. Section 2 defines honest-looking parties and robustness of protocols to such parties. General "compilers" of robust protocols are also defined. Section 3 contains two propositions that demonstrate the intimate relations between the amount of randomness used in the protocol and security against honest-looking parties. Section 4 describes how to transform any protocol to one that is robust to GHL behavior. Section 5 extends the protocol from Section 4 to deal with LHL behavior.

2 Definitions

This section defines the two variants of honest-looking parties, as well as robustness of protocols to such parties. General compilers of protocols into robust ones are also defined. Secure function evaluation in the presence of honest-looking parties is then outlined (we defer formal definition to the full version).

We start with a reminder of the notion of ensembles and indistinguishability [20, 28]. A probability ensemble $X = \{X(k,a)\}_{k\in\mathbf{N},a\in\{0,1\}^*}$ is an infinite sequence of probability distributions, where a distribution $X(k,a)$ is associated with each values of $k \in \mathbf{N}$ and $a \in \{0,1\}^*$. The distribution ensembles we consider in the sequel describe computations where the parameter a corresponds to various types of inputs, and the parameter k is taken to be the security parameter.

Definition 1: We say that two distribution ensembles X and Y are computationally indistinguishable (and write $X \stackrel{c}{\approx} Y$) if for any $c > 0$, for every algorithm D that is probabilistic polynomial-time in its first input, for all sufficiently large k, and all a we have that $|\text{Prob}(D(1^k, a, X(k,a)) = 1) - \text{Prob}(D(1^k, a, Y(k,a)) = 1)| < k^{-c}$.[2]

[2] We use the same notation in the case where $X(k,a)$ and $Y(k,a)$ are distributions over $\{0,1\}$. Here $X \stackrel{c}{\approx} Y$ simply means that $|\text{Prob}(X(k,a) = 1) - \text{Prob}(Y(k,a) = 1)| < k^{-c}$.

The model of computation. An n-party protocol π is a collection of n interactive, probabilistic Turing machines, where the ith machine is associated with the ith party, P_i. Each P_i has input x_i and random input r_i, as well as the security parameter k. Each two parties are connected via a communication channel. In this work we assume secure channels. That is, every two parties may communicate so that no other party, nor the adversary, learns the exchanged information. We also assume synchronous communication. A probabilistic polynomial time (PPT) adaptive t-limited real-life adversary $\mathcal{A}$ is another interactive (computationally bounded) Turing machine that starts off with random input and some auxiliary input. It may choose to corrupt parties during the computation based on the information known to it, and as long as at most t parties are corrupted altogether. Once a party is corrupted the party's input, random input, auxiliary input and the entire history of the messages sent and received by the party become known to the adversary. From this point on the party follows the instructions of $\mathcal{A}$, regardless of protocol π.

Honest-looking parties. Intuitively, honest-looking parties are parties that, until the point of corruption, are indistinguishable by the other parties from totally honest parties. Globally honest-looking (GHL) parties may arbitrarily deviate from their protocol as long as they remain undetected given the combined views of all parties and the adversary. Locally honest-looking (LHL) parties deviate from the protocol in an arbitrary way, as long as no *single* party can distinguish them from honest parties.

The following definitions are aimed at capturing these intuitive notions. An execution of a protocol is the process of running the protocol with a given adversary on given inputs, random inputs, and auxiliary input for the adversary. The configuration of an uncorrupted party at some round of an execution consists of the contents of all tapes of this party, the head position and the control state, taken at the end of this round. In particular, the configuration includes all the messages sent to this party at this round. The internal history of a party at some round of an execution is the concatenation of all the past configurations of this party in this execution. The internal history of the adversary is identical to its current configuration. The global history of the system at some round of an execution is the concatenation of the internal histories of the parties and the adversary at this round.

Let $\text{IH}_{\pi,\mathcal{A},l}(k,\vec{x},z,\vec{r})_0$ denote the internal history at round l of adversary $\mathcal{A}$ when interacting with parties running protocol π on inputs $\vec{x}=x_1\ldots x_n$, auxiliary inputs z and random input $\vec{r}=r_0\ldots r_n$ and with security parameter k, as described above (r_0 and z for $\mathcal{A}$, x_i and r_i for party P_i). Let $\text{IH}_{\pi,\mathcal{A},l}(k,\vec{x},z,\vec{r})_i$ denote the internal history of party P_i at round l of this execution. Let $\text{GH}_{\pi,\mathcal{A},l}(k,\vec{x},z,\vec{r}) = \text{IH}_{\pi,\mathcal{A},l}(k,\vec{x},z,\vec{r})_0, \text{IH}_{\pi,\mathcal{A},l}(k,\vec{x},z,\vec{r})_1, \ldots, \text{IH}_{\pi,\mathcal{A},l}(k,\vec{x},z,\vec{r})_n$ be the global history of this run. Given an execution of a protocol, let l_i^* denote the last round in which party P_i is uncorrupted. (If P_i is never corrupted in this execution then l_i^* is the last round of the execution.) Let $\text{GH}^{(i)}_{\pi,\mathcal{A}}(k,\vec{x},z,\vec{r})$ denote $\text{GH}_{\pi,\mathcal{A},l_i^*}(k,\vec{x},z,\vec{r})$ where the internal history of P_i is removed. Let $\text{IH}^{(i,j)}_{\pi,\mathcal{A}}(k,\vec{x},z,\vec{r})$ denote $\text{IH}_{\pi,\mathcal{A},l_i^*}(k,\vec{x},z,\vec{r})_j$. That is, $\text{GH}^{(i)}_{\pi,\mathcal{A}}(k,\vec{x},z,\vec{r})$ describes the global history of the system, except for P_i, right before P_i is corrupted; $\text{IH}^{(i,j)}_{\pi,\mathcal{A}}(k,\vec{x},z,\vec{r})$ denotes the internal history of P_j right before P_i is corrupted.

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

$\{\text{GH}^{(i)}_{\pi,\mathcal{A}}(k,\vec{x},z)\}_{k\in\mathbf{N},\langle\vec{x},z\rangle\in\{0,1\}^*}$.[3] Let $\text{IH}^{(i,j)}_{\pi,\mathcal{A}}$ be analogously defined with respect to $\text{IH}^{(i,j)}_{\pi,\mathcal{A}}(k,\vec{x},z,\vec{r})$. Given n-party protocols π and π' and a subset $I\subseteq[n]$, let $\pi/_{(I,\pi')}$ be the protocol where the parties in I run protocol π' and all other parties run π. Let $\bar{I}=[n]-I$.

Definition 2: Let $n\in\mathbf{N}$ and let π,π' be n-party protocols. Say that protocol π' is globally honest-looking (GHL) for protocol π if for any PPT t-limited adversary $\mathcal{A}$, any $I\subset[n]$, and any party P_i we have

$$\text{GH}^{(i)}_{\pi_{(I,\pi')},\mathcal{A}} \stackrel{c}{\approx} \text{GH}^{(i)}_{\pi,\mathcal{A}}. \tag{1}$$

(Although the quantification is over all $I\subset[n]$, the interesting cases are when $i\in I$.)

Defining LHL parties requires some more care. A first attempt may be to simply replace condition (1) in Definition 2 with the condition that for any two parties $P_i\neq P_j$ we have

$$\text{IH}^{(i,j)}_{\pi_{(I,\pi')},\mathcal{A}} \stackrel{c}{\approx} \text{IH}^{(i,j)}_{\pi,\mathcal{A}}. \tag{2}$$

However, this requirement is too restrictive. To see that, let us reformulate condition (2) as follows: Let T be a PPT test that takes for input the view of P_j and outputs a binary value, interpreted as an "opinion" whether P_i has followed its protocol. Then, condition (2) is equivalent to saying that *any* test T has the same output distribution (up to negligible difference) in the case where P_i runs the original protocol π and in the case where P_i runs the modified protocol π'. Quantifying over *all* tests allows also tests where P_j's opinion is affected by the adversary in an arbitrary way. Such tests may unnecessarily restrict π', and do not capture our intuitive notion that the adversary should be unable to "frame" truly honest parties. We thus relax the definition of LHL parties as follows. First we define valid tests; these are tests that are not affected by the adversary *in the case where the tested party follows its protocol*:

Definition 3: Let $n\in\mathbf{N}$ and let π,π' be n-party protocols. A PPT machine T with binary output is a valid test for protocols π,π' if for any two PPT t-limited adversaries $\mathcal{A}$ and $\mathcal{A}'$, any $I\subset[n]$ with $i\neq I$, and any parties P_i and P_j we have

$$T(\text{GH}^{(i,j)}_{\pi_{(I,\pi')},\mathcal{A}}) \stackrel{c}{\approx} T(\text{GH}^{(i,j)}_{\pi_{(I,\pi')},\mathcal{A}'}) \tag{3}$$

where $T(\text{GH}^{(i,j)}_{\pi_{(I,\pi')},\mathcal{A}})$ denotes the probability ensemble $\{T(\text{GH}^{(i,j)}_{\pi,\mathcal{A}}(k,\vec{x},z))\}_{k\in\mathbf{N},\langle\vec{x},z\rangle\in\{0,1\}^*}$.

A LHL protocol is now required to pass only all *valid* tests:

Definition 4: Let $n\in\mathbf{N}$ and let π,π' be n-party protocols. Let $n\in\mathbf{N}$ and let π,π' be n-party protocols. Say that protocol π' is locally honest-looking (LHL) for protocol π if for any PPT t-limited adversary $\mathcal{A}$, any $I\subset[n]$, any two parties P_i and P_j, any valid test T we have

$$T(\text{GH}^{(i,j)}_{\pi_{(I,\pi')},\mathcal{A}}) \stackrel{c}{\approx} T(\text{GH}^{(i,j)}_{\pi,\mathcal{A}}). \tag{4}$$

[3] Let $\langle\vec{x},z\rangle$ denote some natural encoding of $\vec{x},z$ as a single string.

REMARKS:

(I). It is stressed that the global histories in the two sides of (1) (resp., (2)) are based on the same inputs for the parties. Furthermore, the distinguisher has access to these inputs. In other words, we require an honest-looking party to be indistinguishable from an honest party running the protocol *on the same input.* This models the fact that we regard inputs as fixed and external to the protocol, rather than decidable by the party.

(II). The requirement that an honest-looking party must remain indistinguishable from honest (by all other parties and the adversary) implies that honest-looking parties (both GHL and LHL) do not collaborate. In other words, honest-looking parties do not 'pool' their local information and are not jointly controlled by an adversary.

(III). In [7] two other notions of internal deviation from the protocol are defined: *Honest-looking* parties are those that are indistinguishable from honest ones by *any* external efficient text; *weakly honest* parties need only be indistinguishable from honest by parties running the (honest) protocol, when no adversary is present. Our definition of GHL parties is similar to that of weakly honest parties, except that we require the GHL party to be indistinguishable from honest even in the presence of an adversary. The definition here may better capture the intuitive notion of weakly honest parties.

Robustness of protocols against honest-looking parties. Intuitively, a protocol π is "robust" against honest-looking behavior if for any protocol π' that is honest-looking for π and for any subset I of parties that run π', the functionality of the parties in $\bar{I}$ remains unchanged from the case where all parties run π.

This intuitive notion is formalized roughly as follows. We say that π is t-robust against GHL (resp., LHL) parties if for any protocol π' that is GHL (resp., LHL) for π, for any subset I of parties that run π', and for any adversary $\mathcal{A}'$, there exists an adversary $\mathcal{A}$ such that the outputs of the parties in $\bar{I}$ when running $\pi_{(I,\pi')}$ and interacting with $\mathcal{A}'$, is distributed identically to the outputs of the parties in $\bar{I}$ when all parties are running π and interacting with $\mathcal{A}$.

More precisely, we first formalize the global output of a protocol π. Let $\text{ADV}_{\pi,\mathcal{A}}(k,\vec{x},z,\vec{r})$ 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 $\vec{r} = r_0 \ldots r_n$ and with security parameter k, as described above. Let $\text{EXEC}_{\pi,\mathcal{A}}(k,\vec{x},z,\vec{r})_i$ denote the *output* of party P_i from this execution. (If P_i is uncorrupted then this is the output specified by π; if P_i is corrupted then $\text{EXEC}_{\pi,\mathcal{A}}(k,\vec{x},z,\vec{r})_i = \perp$.) Let $\text{EXEC}_{\pi,\mathcal{A}}(k,\vec{x},z,\vec{r}) = \text{ADV}_{\pi,\mathcal{A}}(k,\vec{x},z,\vec{r}), \text{EXEC}_{\pi,\mathcal{A}}(k,\vec{x},z,\vec{r})_1, \ldots,$ $\text{EXEC}_{\pi,\mathcal{A}}(k,\vec{x},z,\vec{r})_n$. denote the global output of this run. Let $\text{EXEC}_{\pi,\mathcal{A}}(k,\vec{x},z)$ denote the probability distribution of $\text{EXEC}_{\pi,\mathcal{A}}(k,\vec{x},z,\vec{r})$ where $\vec{r}$ is uniformly chosen. Let $\text{EXEC}_{\pi,\mathcal{A}}$ denote the probability ensemble $\{\text{EXEC}_{\pi,\mathcal{A}}(k,\vec{x},z)\}_{k\in\mathbf{N},\langle\vec{x},z\rangle\in\{0,1\}^*}$. Given a set $I \subseteq [n]$ of parties, let $\text{EXEC}_{\pi,\mathcal{A},I}$ denote the vector $\text{EXEC}_{\pi,\mathcal{A}}$ restricted to the output of the adversary and the parties in I.

Definition 5: Let $n \in \mathbf{N}$, let $t < n$, and let π be an n-party protocol. Protocol π is t-robust to GHL parties (resp., t-robust to LHL parties) if for any n-party protocol π' that is GHL (resp., LHL) for π, and any t-limited adversary $\mathcal{A}'$, there exists an adversary $\mathcal{A}$ such that for any subset $I \subset [n]$ we have

$$\text{EXEC}_{\pi_{(I,\pi')},\mathcal{A}',\bar{I}} \stackrel{c}{\approx} \text{EXEC}_{\pi,\mathcal{A},\bar{I}}.$$

Remarks GHL parties are a security concern only in the presence of adaptive adversaries. (It follows from the definition of GHL parties that if the identities of the corrupted parties are fixed in advance then any protocol is t-robust to GHL parties, for any t.) However, LHL parties are, potentially, a security threat even in the presence of *static* adversaries.

General compilers of robust protocols. In the sequel we present general constructions for transforming any protocol into an "equivalent" one, that in addition is robust against honest-looking parties. We formalize this notion as follows. Let C be a transformation that takes (descriptions of) n-party protocols and outputs (descriptions of) n-party protocols. We say that C is a t-emulating compiler if for any n-party protocol π and for any adversary $\mathcal{A}'$ there exists an adversary $\mathcal{A}$ such that $\text{EXEC}^{C(\pi),\mathcal{A}'} \stackrel{c}{\approx} \text{EXEC}^{\pi,\mathcal{A}}$. A compiler is t-robust against GHL parties (resp., t-robust against LHL parties) if it is t-emulating, and for any given protocol π the protocol $C(\pi)$ is t-robust to GHL (resp., LHL) parties.

Secure function evaluation with honest-looking parties. The standard notion of secure function evaluation, as defined in [26, 19, 3, 6] and elsewhere, concentrates on the case where all uncorrupted parties follow the prescribed protocol. In a nutshell, a protocol π securely evaluates a function f if it emulates an ideal process for distributively evaluating f, in the following sense: For *any* adversary $\mathcal{A}$ that interacts with parties running π, there should *exist* an adversary $\mathcal{S}$ that interacts with the ideal process, and such that the global output of the ideal process interacting with $\mathcal{S}$ is indistinguishable from the global output of running π with $\mathcal{A}$.

A natural extension of this definition may say that π securely evaluates f *in the presence of honest-looking parties,* if for any protocol π' that is honest-looking for π, and for any subset $I \subset [n]$, the protocol $\pi_{(I,\pi')}$ emulates the ideal process of evaluating f (when the outputs of the parties in I are disregarded).

Adopting this formalization, we now get as an easy corollary that if a protocol π is t-robust against GHL (resp., LHL) parties, and in addition π t-securely evaluates a function f, then π t-securely evaluates f in the presence of GHL (resp., LHL) parties. We leave the formalization of this definition and claim out of the scope of this paper.

Protocol families. Definitions 2, 4 and 5 treat n, the number of parties, as a constant. In the sequel we discuss protocol families, where the number of parties is polynomially related to the security parameter, and in particular tends to infinity. Robustness of protocol families is defined as follows. Let $\Pi = \{\pi^n\}_{n\in\mathbf{N}}$ be a family of multi-party protocols (π^n is an n-party protocol). Then, Π is $t(n)$-robust to GHL/LHL parties if for all large enough n, π^n is $t(n)$-robust to GHL/LHL parties. For simplicity, we identify n with the security parameter, k.

3 GHL-security versus local entropy

We show that security against honest-looking parties is intimately related to the amount of internal randomness used by the (honest) protocols, in the following way: First we show an example that suggests that proving robustness to GHL parties of protocols where parties 'use a lot of locally generated randomness' is implausible. Next we show that if the protocol instructs each party to use only a small (i.e., logarithmic in the security parameter) amount of randomness then robustness to GHL parties is guaranteed. This fact plays a central role in the construction of the next section.

A protocol that is not robust to GHL parties. It was observed in [7] that even the seemingly innocuous protocol where each party is instructed to send only a single random string,

independent from its input, cannot be proven robust to even GHL parties if one-pass black-box simulation is used. Using a technique from [9], we generalize this example and in particular remove the restriction to black-box simulation.

Recall that a family of permutation pairs $\mathcal{G} = \{g_{m,0}, g_{m,1}\}_{m\in\{0,1\}^*}$ (i.e., for each value of m, both $g_{m,0}$ and $g_{m,1}$ are permutations on a common domain D_m) is claw free if no PPT adversary can, given $m \stackrel{R}{\leftarrow} \{0,1\}^n$, find with non-negligible (in n) probability values $v_0, v_1 \in D_m$ such that $g_{m,0}(v_0) = g_{m,1}(v_1)$. A function family $\mathcal{H} = \{h_m\}_{m\in\{0,1\}^*}$ is collision-free if no PPT adversary can, given $m \stackrel{R}{\leftarrow} \{0,1\}^n$, find with non-negligible probability values $v_0, v_1 \in D_m$ such that $h_m(v_0) = h_m(v_1)$. Existence of claw free permutation pairs implies existence of collision-free functions [12].

Proposition 1: Assume claw-free permutation pairs exist, and let $t = \Omega(n^\epsilon)$ for some $0 < \epsilon < 1$. Let $\Pi = \{\pi^n\}_{n\in\mathbf{N}}$ be the following protocol family, where P_1 has an input $m \in \{0,1\}^n$ and all other parties have binary input. First, party P_1 broadcasts m; next, each party chooses and broadcasts a random value in the domain D_m of a claw-free permutation pair $\mathcal{G}\{g_{m,0}, g_{m,1}\}$. The protocol has empty output. Then, Π is not t-robust to GHL parties.

Proof sketch: Consider the following protocol family Π'. Let x_i denote P_i's input. Instead of sending a random value in D_m, each party P_i, $i \neq 1$, chooses $r_i \stackrel{R}{\leftarrow} D_m$ and sends $g_{m,x_i}(r_i)$ to all other parties. Clearly Π' is GHL for Π, since the g's are permutations. We show that if Π' satisfies Definition 5 (as a protocol family) then claws can be found in $\mathcal{G}$.

Let $\mathcal{H} = \{h_m\}$ be a function family that is collision-free if $\mathcal{G}$ is claw-free, and where $h_m : (D_m)^{n-1} \to [n]^{t-1}$, where $n = |m|$. Let $\mathcal{A}'$ be the following adversary, operating against Π'. $\mathcal{A}'$ gets n and auxiliary input (which is ignored). First, $\mathcal{A}'$ corrupts P_1, and interprets P_1's input as an index m of a domain D_m for family $\mathcal{G}$. $\mathcal{A}'$ lets P_1 announce m and records the values $s_2...s_n \in D_{x_1}$ sent by the parties. Then, $\mathcal{A}'$ feeds $s = s_2...s_n$ to function h_m and interprets the result as a set, C, of $t-1$ more parties to corrupt. It then corrupts these parties and outputs s, C, and the internal random choices r_C, and inputs x_C, of the parties in C.

Now, let $n \in \mathbf{N}$ and let $I = [n]$ (i.e., all parties are running π'^n). Assume that there exists an adversary $\mathcal{A}$ such that $\text{ADV}_{\pi^n_{(I,\pi'^n)},\mathcal{A}'} \stackrel{c}{\approx} \text{ADV}_{\pi^n,\mathcal{A}}$. That is, $\mathcal{A}$ has the following functionality: first it gets auxiliary and random input. Next it corrupts P_1, learns m, records the messages sent by the other parties, corrupt a set of parties and learns their inputs. (It can be seen that $\mathcal{A}$ must comply with this order of events.) Next $\mathcal{A}$ generates an output $\hat{O} = (\hat{s}, \hat{C}, \hat{r}_C, \hat{v}_C)$ that is indistinguishable from a real output of $\mathcal{A}'$ interacting with π' on inputs $\vec{x} = m, x_2...x_n$ for the parties (and some auxiliary input).

We construct an algorithm $\mathcal{D}$ that, on input $m \stackrel{R}{\leftarrow} \{0,1\}^n$, uses $\mathcal{A}$ to either find claws in $\mathcal{G}$ or to find collisions in $\mathcal{H}$. $\mathcal{D}$ starts by running $\mathcal{A}$ on some random input ρ. When $\mathcal{A}$ corrupts P_1 it is given input value m for P_1. Whenever $\mathcal{A}$ corrupts other parties, it is told that the corresponding inputs are 0. Finally $\mathcal{A}$ generates an output $\hat{O} = (\hat{s}, \hat{C}, \hat{r}_C, \hat{x}_C)$. Clearly this output has the property that $h_m(\hat{s}) = (\hat{C})$ (that is, h_m(first round communication) = {identities of corrupted parties}.) Next, $\mathcal{D}$ runs $\mathcal{A}$ again, on the same random input ρ, but now when $\mathcal{A}$ corrupts the last party, $\mathcal{D}$ tells $\mathcal{A}$ that the input of this party is 1. Let P_* denote this party. Next $\mathcal{A}$ generates an output $\hat{O}'$.

Note that $\hat{C}$, the set of corrupted parties, is the same in $\hat{O}$ and in $\hat{O}'$. This is so since until the point where all corruptions are done, the view of $\mathcal{A}$ is the same in both runs. This fact is used by $\mathcal{D}$ as follows.

Let $\hat{s}_*$ and $\hat{s}'_*$ denote the values sent by P_* in $\hat{O}$ and in $\hat{O}'$, respectively. Let r_* and r'_* be the corresponding internal random choices of P_*. Now, if $\hat{s}_* = \hat{s}'_*$ then by the validity of $\mathcal{A}$ we have that $g_{m,0}(\hat{r}_*) = g_{m,1}(\hat{r}'_*) = \hat{s}_*$, and $\mathcal{D}$ has found a claw in $\mathcal{G}$. If, on the other hand, $\hat{s}_* \neq \hat{s}'_*$ then of-course $\hat{s} \neq \hat{s}'$. However, by the validity of $\mathcal{A}$ we have that $h_m(\hat{s}) = h_m(\hat{s}') = \hat{C}$, and $\mathcal{D}$ has found a collision in $\mathcal{H}$. ■

Remarks:

(I). The example protocol analyzed in Proposition 1 is quite generic. In fact, most known protocols for secure function evaluation can be shown in a similar manner to not withstand GHL parties. Furthermore, it seems that in the computational setting, where no ideally secure channels exist, it is *necessary* to have each party use a 'large amount of randomness' in a way that does not allow security against GHL parties.

(II). The proof of Proposition 1 requires n, the number of parties, to grow to infinity. We remark that if we restrict the simulator to one-pass black-box simulation then similar examples can be shown where the number of parties is a small as $n = 2$.

Small local entropy implies robustness to GHL parties. We show that any protocol that instructs each uncorrupted party to use only a small (i.e., logarithmic in the security parameter) number of random bits is t-robust to GHL parties, for any t. Interestingly, the result holds even if the GHL protocol uses as many random bits as it wishes.

Proposition 2: Let $t = t(n) < n$, and let $\Pi = \{\pi^n\}_{n\in\mathbf{N}}$ be a protocol family where the random input of each party is of size at most $O(\log n)$. Then Π is t-robust to GHL parties.

Proof sketch: Let $n \in \mathbf{N}$ Let Π' be a GHL protocol for Π, and let $\mathcal{A}'$ be an adversary. We construct an adversary $\mathcal{A}$ such that for all families of subsets $\mathcal{I} = \{I \subset [n]\}_{n\in\mathbf{N}}$ it holds that $\text{EXEC}_{\Pi_{(\mathcal{I},\Pi')},\mathcal{A}',\bar{\mathcal{I}}} \stackrel{c}{\approx} \text{EXEC}_{\Pi,\mathcal{A},\bar{\mathcal{I}}}$.

Let $n \in \mathbf{N}$ and let $I \subset [n]$. Adversary $\mathcal{A}$ runs a copy of $\mathcal{A}'$, follows the instructions of $\mathcal{A}'$, and forwards all the gathered information to $\mathcal{A}'$, with the following exception. When $\mathcal{A}'$ wishes to corrupt a party P_i that runs the GHL protocol π'^n (that is, when $i \in I$), $\mathcal{A}$ proceeds as follows. First, $\mathcal{A}$ corrupts P_i, obtains P_i's input, x_i, and hands it to $\mathcal{A}'$. Next, $\mathcal{A}$ has to provide $\mathcal{A}'$ with random input for P_i, *according to protocol* π^n. This is done as follows: $\mathcal{A}'$ chooses a value r_i uniformly at random from all the possible random inputs of P_i that are consistent with x_i and P_i's past messages that are already known to $\mathcal{A}'$. Next, $\mathcal{A}$ hands r_i to $\mathcal{A}'$ and the interaction continues. (If no consistent r_i is found then $\mathcal{A}$ halts.) Finally, $\mathcal{A}$ outputs whatever $\mathcal{A}'$ outputs.

We first note that $\mathcal{A}$ runs in polynomial time, since there are only polynomially many possible random inputs for each P_i. Next assume that there is a distinguisher D between $\text{EXEC}_{\pi^n_{(I,\pi'^n)},\mathcal{A}',\bar{I}}(n,\vec{x},z)$ and $\text{EXEC}_{\pi^n,\mathcal{A},\bar{I}}(n,\vec{x},z)$ for some $n,\vec{x},z$. Then D can be used to distinguish between $\text{GH}^{(i)}_{\pi^n_{(I,\pi'^n)},\mathcal{A}}(n,\vec{x},z)$ and $\text{GH}^{(i)}_{\pi,\mathcal{A}}(n,\vec{x},z)$ for some $P_i, n, \vec{x}, z$, thus contradicting the premise that Π is GHL for Π (see Definition 2). We omit further details from this abstract. ■

4 Robust protocols for GHL parties

We begin by stating the main theorem of this section. Recall that private channels between parties are provided, and that the adversary is probabilistic-polynomial time. In addition, we assume existence of pseudorandom number generators (PRGs), which is equivalent to assuming existence of one-way functions.

Theorem 3: Assume that one way-way functions exist, and let $t(n) \leq n^{-20}$. Then[4] there exists a compiler that is t-robust to GHL parties.

By Proposition 2, it suffices to describe a t-emulating compiler C such that, given any protocol π, the protocol $C(\pi)$ instructs each party to use at most $O(\log n)$ random bits. Our compiler follows this path. In fact, each party will use at most *one* random bit.

Protocol outline. The protocol consists of two main phases. The first phase is independent of the inputs, and its sole goal is to generate randomness for the second phase. The second phase consists of running the original protocol, using the randomness generated in the first phase. We stress that the second phase is entirely deterministic; its only source of randomness is the first phase.

We motivate our (rather complex) construction of the first phase. A naive attempt at constructing this phase may proceed as follows: each party chooses one random bit and sends it to a special party, called the generator. The generator collects all random bits, feeds them to a PRG, and hands each party a sufficiently long piece of the PRG's output. Each party now uses this string as its random input for phase two.

This solution, however, clearly does not work. A first problem is that the generator may be corrupted. This problem can be easily solved, at the expense of reducing t: have $t+1$ distinct generators, and $\frac{n}{t+1}$ distinct randomness suppliers for each generator. Each party will bitwise-xor the strings received from all generators, and use the outcome for phase two. However, this is also not sufficient. A more subtle problem remains: we must show that the pseudo-randomness supplied to phase two does not compromise the security of the underlying protocol. Technically, in order to make the proof work, the phase one protocol should be accompanied with a "simulator" that has the following property: Whenever a party is corrupted the simulator gets as input some arbitrary string, ρ, of the appropriate length, and comes up with a view of the adversary that is of the right distribution *conditioned on the event that the party's output of phase one is* ρ. Guaranteeing this property is the driving force behind our construction, outlined next.

PROTOCOL SETUP: Let k be a security parameter. We will need $n \geq k^{17}$ parties, and will allow $t \leq k^2$. We start by splitting the parties to two sets A and B of equal size. The parties in A produce randomness for the parties in B, and vice versa. That is, each party doubles up both as a producer and as a consumer of randomness. We describe how the parties in A play as producers and how the parties in B play as consumers. The activity with reversed roles is identical. The parties in A are partitioned into $O(t^4 \log n)$ groups. Each group consists of $O(t^4)$ generators, one of which is called the leader, and of $O(t^4 k)$ suppliers. With each generator we associate k suppliers. We remark that, in our protocol, each group will play the role played by single generator in the above naive protocol.

[4] We remark that the exponent 20 here is not optimized and can be somewhat reduced.

Each party in B (taking the role of a consumer) is assigned to $O(t^3 \log n)$ groups in A, as follows. We use a family H of $O(t \log n)$ perfect hash functions $h : \{1, \ldots n\} \to \{1, \ldots, O(t^2)\}$. With each function h and each point in $\{1, \ldots, O(t^2)\}$ we associate $t+1$ distinct groups. Consumer P_i is assigned to the groups associated with $h(P_i)$ for all $h \in H$. This assignment guarantees that any newly corrupted consumer will have $t+1$ groups that are not assigned to any other consumer that is already corrupted. Consequently, when a consumer is corrupted there is at least one group that is assigned to this consumer, and whose outputs are completely unknown to the adversary.

PROTOCOL MOVES: The protocol starts by having each of the suppliers send a random bit to the generator associated with it. Next, each generator puts aside $\log t$ random bits, feeds the remaining $k - \log t$ random bits to a PRG, and obtains a pseudorandom string, T, of the appropriate length.

Next, the generators in each group form a "random path" of length t^2 among themselves, with the leader being the first in the path. This is done as follows: first the leader chooses (using the $\log t$ remaining random bits) another generator g in the group, and notifies g that it is the second in the path. Next, g chooses the next generator in the path, and so on for t^2 steps. Once the path is complete, the last generator in the path sends its generated string T to its neighbor in the path. The neighbor bitwise-xors the received string with its own generated string, and passes the result to its next neighbor. Once the group leader receives the string, it bitwise-xors it with its own string, and sends a sufficiently long segment of the resulting string to each consumer that is assigned to the group.

The purpose of this multi-round, gradual buildup of the pseudorandom string generated by the group is to allow the simulator of phase one to always generate "credible" inputs for corrupted generators: Since the path of generators contains a long enough simple sub-path to prevent the adversary from checking consistency along it, the simulator can always set either the input string or the output string to convenient values. We now provide the details.

A detailed description of phase one: First we divide the parties into several categories, where different categories of parties perform different tasks. We describe the different categories and their sizes.

1 Divide all n parties into two disjoint sets A, B of equal size.
2. In each set form $O(t^4 \log n)$ disjoint *groups* $G_1, \ldots, G_{t^4 \log n}$:
3. Where each groups consists of the following disjoint parties:
5. – $O(t^4)$ parties called *generators* $g_1, \ldots, g_{t^4}$, (g_1 is also called a *leader*)
6. – $O(t^4 \cdot (k + 4 \log t))$ parties called *suppliers* where with each generator g_i a disjoint set of $(k + 4 \log t)$ suppliers $s_i^1 \ldots$

We use a family H of $O(t \log n)$ perfect hash functions where every $h \in H$ maps $\{1, \ldots n\}$ to $\{1, \ldots, O(t^2)\}$. With each hash function h, we associate $O(t^3)$ distinct groups, $t+1$ groups for each value in the range of h.

1. Each s_i^j (via private channel) sends a random bit to the g_i associated with it.
2. For each group G do:
3. each $g_i \in G$ uses k supplier bits to generate
pseudo-random sequence T_i of length $n \cdot T(n)$.
g_i uses the remaining $4 \log t$ supplier bits to select
a random $g \in G$, denoted NEXT(g_i).
4. $x \longleftarrow t^2$, CURRENT $\longleftarrow g_1$
5. Until $x > 0$ do:
6. party CURRENT sends "(CURRENT,x)" to
NEXT(CURRENT)
7. party NEXT(CURRENT) marks CURRENT

as its PREDECESSOR,
party NEXT(CURRENT) becomes CURRENT,
decrements x and repeats.

8. Party $g_i \in G$ which hold $x = 0$ sends
T_i to PREDECESSOR(g_i)
9. For x from 1 to t^2 do:
10. party g_i which holds x upon receiving
msg T from NEXT(g_i) sends
bitwise-xor $T_i \oplus T$ to PREDECESSOR(g_i)
11. g_1 computes $T_1 \oplus T$, and partitions it into n equal
distinct segments $r_1, \ldots, r_n$.
12. For each party P_i (in A/B)
13 For each hash function $h \in H$
14. the $t+1$ leaders of the groups associated
with the output $h(P_i)$
send their r_i (computed in step 11) to P_i.
15. Each party bitwise-xors the pseudo-random strings
received from the $O(t^3 \log n)$ different groups
and uses it as its random string.

Proof of security. Let π be an n-party protocol, let $\pi' = C(\pi)$ be the transformed protocol, and let $\mathcal{A}'$ be an adversary that interacts with π'. We construct an adversary $\mathcal{A}$ that interacts with π and such that $\text{EXEC}_{\pi',\mathcal{A}'}(n, \vec{x}, z) \stackrel{c}{\approx} \text{EXEC}_{n,\pi,\mathcal{A}}(n, \vec{x}, z)$ for all inputs $\vec{x}$ and auxiliary input z.

Adversary $\mathcal{A}$ consists of two phases, corresponding to the two phases of π'. Phase one of the simulator proceeds as follows. First, $\mathcal{A}$ chooses a random bit for each supplier, and carries out a complete simulation of the ensuing execution up til the end of phase one. That is, $\mathcal{A}$ follows $\mathcal{A}'$'s instructions and hands $\mathcal{A}'$ all the gathered information. Whenever $\mathcal{A}'$ corrupts a party during phase one, $\mathcal{A}$ hands $\mathcal{A}'$ the true input and random input of that party, as well as all the messages that this party received so far. Note that throughout phase one $\mathcal{A}$ has not yet started interacting with its network.

Phase two of $\mathcal{A}$ consists of following the instructions of $\mathcal{A}'$, and handing $\mathcal{A}$ all the gathered information, with the following exception. Whenever a party P_i is corrupted, $\mathcal{A}$ obtains the random input γ of P_i, and simulates P_i's internal data from phase one, as follows.

We first describe how to simulate P_i's role as a consumer of randomness. As noted above, the construction guarantees that there exist at least one group assigned to P_i that is not assigned to any other corrupted party, and such that no party in g is corrupted. We regard the output values of all other groups assigned to P_i as already fixed to arbitrary values (otherwise, we fix them to random values). Next, we fix the output of group g (sent by g's leader to P_i) to the bitwise-xor of γ with the already fixed outputs of all other groups. Call this value δ. Later, when parties in group g will be corrupted, we will have to demonstrate how the value δ was generated.

Next we describe how to simulate P_i's role as a producer. We distinguish several cases:

1. If P_i is the first party in its group to be corrupted, and none of the parties assigned to this group was corrupted, then we fix the random choices of the producers in this group to some random values, and present P_i's internal data accordingly. We call such a group fixed.

2. If P_i belongs to a group that is already fixed then its internal data are set according to the already fixed values of the group.

3. If P_i belongs to a group who is not fixed but whose output value δ was fixed by some consumer assigned to it, then $\mathcal{A}$ proceeds as follows. First, it chooses a random path of length t^2 among the generators in the group, as described in the protocol. Let us assume for simplicity that the path is simple. Otherwise, pick a long enough simple sub-path (such a sub-path exists with high probability), and repeat the following

arguments on this sub-path.

A generator is called set if either its output (input) strings are known to the adversary. Whenever a generator is set, the simulator proceeds as follows: it chooses random bits for all the generator's suppliers, applies the PRG to these values, and fixes the appropriate input (output) value. The group leader is set with its output fixed to δ. The last generator in the path is set since it has no input string. Say that a simple sub-path is free if none of the generators in this sub-path are set. When the output value δ of this group is fixed, the group's free path is of length $O(t^2/\log t)$. (Note that a free path hides an inconsistency: The input/output values of the set generators at the two ends of the path are unlikely to agree, if all the generators along the path were set. The goal of the simulation is to keep this inconsistency hidden from the adversary.) Next:

3a. If P_i is a generator on the free path then $\mathcal{A}$ makes the following decision: If P_i is closer to the head of the free path (i.e., closer to the group leader's side) then set P_i and all the generators in the segment between P_i and the head of the path, fixing their output values to be consistent with the input values of their predecessors. If P_i is closer to the tail of the free path then set P_i and all the generators in the segment between P_i and the tail of the path, fixing their input values to be consistent with the output values of their successors. Note that we are now left with a path of at least half the length from before.

3b. If P_i is a generator not on the free path, or a supplier, then its internal view is chosen (by $\mathcal{A}$) in an obvious way.

Analyzing $\mathcal{A}$, we first state the property of perfect hash functions, that guaranteed that each newly corrupted P_i has a group associated with it that is not yet fixed:

Lemma 4: [14]Let n and t be any two integers. Then there exists a family $|H| = O(t \log n)$ of hash functions such that every $h \in H$ maps $\{1, \ldots n\}$ to $\{1, \ldots, O(t^2)\}$ and for every subset of $\{1, \ldots n\}$ of size at most t, at least one $h \in H$ is 1-1 over this subset.

The above lemma implies that no matter which t parties the adversary will corrupt in the underlying simulation, there will be at least one hash function h for which every time the adversary corrupts a new party a new $(t+1)$ groups which supplies randomness for this party are introduced. Therefore, at least one of the $(t+1)$ groups have not been seen by $\mathcal{A}'$.

Next we note that the simulation is successful as long as the simulator does not "run out" of the free path. We have seen that the length of the free path as worst halves at each "hit" of the adversary. It remains to note that the adversary "hits" the free path at most $\Omega(\log t)$ times. This is guaranteed by the fact that, since the path is chosen randomly and locally, the adversary's probability of hitting the free path at a given corruption is the same as in a 'blind' random choice. Specifically, we show:

Lemma 5: With probability at least $1 - e^{-(\log t)^2/3}$ the length of the maximum simple sub-path is at least $t^2/\log t$.

Proof: The probability that at any step we visit generator already visited is at most $1/t^2$. The bound follows from the multiplicative form of the Chernoff bound [1]: if we have m independent experiments with success. probability at most p, then the total number of successes S is bounded by $\Pr[S > (1+\gamma)pm] \leq e^{-mp\gamma^2/3}$ ■

Now, we fix any such simple sub-path (of length $t^2/logt$) and pick t random elements from G. We now consider the following experiment. We pick t random generators. Every time we hit the simple sub-sequence, we eliminate its shorter part and keep the longer part. We wish to measure the probability that the final resulting interval is greater then t:

Lemma 6: Fix $t^2/\log t$ generators. Now pick t generators at random from G. The probability that we pick more then $\log t - \log\log t$ of chosen generators is at most $e^{-t(\ logt - \log\log t)/3}$

Proof: Chernoff bound. ■

5 A LHL-secure protocol for computing any function

Theorem 7: Assume that one way-way functions exist, and let $t(n) \leq n^{-22}$. Then[5] there exists a compiler that is t-robust to LHL parties.

The idea of the construction is to reduce the locally honest-looking (LHL) parties to globally honest-looking (GHL) behavior, by making sure that any deviation from the protocol that is detectable by the entire community will be detected by each single party. Due to space limitations we give only very high-level description here. First, we make sure that every message sent in the network is "registered" within sufficiently many parties. Next, pick a random party, reveal its communication, and check consistency of the party's behavior. We now give an outline how this is done.

Two Main tools: We start with a discussion of the implication of the definition, which is the most delicate issue of this section: suppose sender S must send a bit b to $4t+1$ intermediaries, who then must re-transmit this bit b to some receiver R. Next, we execute deterministic Byzantine agreement among the intermediaries and R to determine what this bit b is. Note that (1) if sender wishes to be locally honest-looking he must send the same bit to all the intermediaries and (2) if intermediaries wish to be locally honest-looking they must send the same bit to the receiver. To see (1) notice that after the Byzantine agreement, if the sender does not send the same bit to all the intermediaries, and assuming that all the intermediaries are totally honest, there will be some intermediary who's value will disagree with the outcome of the Byzantine agreement. That intermediary will decide that if all intermediaries are honest then the sender is not honest-looking, hence the sender can not do this by the specification of the honest-looking behavior. (Notice that this is a valid test, as in Definition 3.) To see (2), notice that after intermediaries and S run Byzantine Agreement, if there is some intermediary who did not send the required value, then the receiver will decide that this intermediary is not honest-looking, again assuming that all but this player is honest. Hence, intermediaries can not afford to change the values either. We will use this basic construct, together with deterministic BA as a basic building block in our solution.

Second, notice that the community can generate a common unbiased coin. This is done as follows: $t+1$ parties flip random coins and send it to distinct $4t$ parties each. Then, for each coin a deterministic BA protocol is executed. The XOR of this t+1 bits defines the coin. Notice that even with adaptive adversary, this coin can not be biased, since all the intermediaries are distinct.

The Reduction. We first augment phase two of the GHL protocol to start with Verifiable Secret Sharing of the private inputs of the players. (In cases that phase two already starts with a VSS of the players' inputs, such as [5], this step is not necessary.) As the rest of phase two, the VSS invocations use the randomness provided by phase one of the GHL protocol.

We then further modify the entire GHL protocol from the previous section as follows. Here we let $n \geq k^{19}$. We partition the parties as follows. $O(t^{17})$ parties play the same roles as in the GHL protocol. To each one of these parties assign $4t$ new parties, called intermediaries. Each message that was sent directly to party P_i in the GHL protocol is now sent to each one of the $4t$ intermediaries associated with it. Next, each one of these intermediaries forwards the message to P_i. (This provision applies both to messages of phase one and to messages of phase two of the modified GHL protocol.)

Once the GHL protocol is done, the parties jointly choose at random a single party, P_i, to be audited as follows: the parties use the intermediaries (of P_i and of all the parties that P_i communicated with) to jointly verify that P_i followed its protocol properly given its inputs and outputs, without revealing P_i's private input. (Notice that the random input of P_i was received from other parties of phase one of the GHL protocol, that P_i already executed a verifiable secret sharing of its private input in phase two of the GHL, and that P_i is in fact "almost" deterministic, except for a single random bit it contributed to the GHL computation of phase one.) The verification is done by invoking another modified GHL computation for securely evaluating the validity of the statement *"P_i has correctly executed the Verifiable Secret Sharing algorithm, based on its input, the pseudorandom values it received and the messages it sent during the sharing stage".* This secure

[5] Again 22 here is not optimized and can be somewhat reduced.

GHL computation is different from the previous one, since players for this computation do not have private inputs.

Finally, the parties verify that the last GHL computation from the previous audit step of player P_i was performed correctly. For this end, an additional party P_j is chosen randomly and audited, as follows. This time, P_j has no private inputs; thus its entire communication is uncovered and agreed upon, where the intermediaries of P_j run deterministic Byzantine Agreement to agree on the messages received and sent by P_j.

Acknowledgments

We thank Le Gamin bistro where most of this work (with plenty of snacks and espresso) was done.

References

[1] N. Alon and J. Spencer. **The Probabilistic Method**. Wiley, 1992.

[2] D. Beaver, "Foundations of Secure Interactive Computing", *CRYPTO,* 1991.

[3] D. Beaver, "Secure Multi-party Protocols and Zero-Knowledge Proof Systems Tolerating a Faulty Minority", J. Cryptology (1991) 4: 75-122.

[4] D. Beaver and S. Goldwasser, "Multi-party computation with faulty majority", *30th FOCS*, 1989, pp. 468-473.

[5] M. Ben-Or, S. Goldwasser and A. Wigderson, "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation", *20th STOC,* 1988, pp. 1-10.

[6] R. Canetti, "Security and composition of multi-party protocols", Available at the Theory of Cryptography Library, http://philby.ucsd.edu, 1998.

[7] R. Canetti, U. Feige, O. Goldreich and M. Naor, "Adaptively Secure Computation", *28th STOC,* 1996. Fuller version in MIT-LCS-TR #682, 1996.

[8] R. Canetti, E. Kushilevitz, R. Ostrovsky and A. Rosen, "Randomness vs. Fault-Tolerance", Available at the Theory of Cryptography Library,
http://philby.ucsd.edu, 1998.
Preliminary version at *16th PODC,* 35-45, 1997.

[9] R. Canetti, T. Malkin, Y. Yishay, in preparation.

[10] D. Chaum, C. Crepeau, and I. Damgard. Multi-party Unconditionally Secure Protocols. In *Proc. 20th Annual Symp. on the Theory of Computing*, pages 11–19. ACM, 1988.

[11] G. Di Crescenzo, R. Ostrovsky and S. Rajagopalan, "Conditional Oblivious Transfer and Sender-Anonymous Timed-Release Encryption" In Proceedings of *EUROCRYPT 99*, to appear.

[12] I.B. Damgård, "Collision free hash functions and public key signature schemes", *EUROCRYPT 87 (LNCS 304)*, pp. 203–216, 1988.

[13] A. Fiat and M. Naor, "Broadcast Encryption", Advances in Cryptology - Crypto '92, Springer-Verlag LNCS 839, pp. 257-270, 1994.

[14] M. Fredman, J. Komlos, and E. Szemeredi "Storing A Sparse Table with O(1) Access Time" *Journal of the ACM* **31**, 1984, pp. 538-544.

[15] J. Garay and Y. Moses. Fully Polynomial Byzantine Agreement in $t+1$ Rounds. *SIAM Journal on Computing*, 27(1), 1998.

[16] O. Goldreich. "Secure Multi-Party Computation" 1998. First draft available at `http://theory.lcs.mit.edu/~oded`

[17] O. Goldreich, S. Micali and A. Wigderson, "How to Play any Mental Game", *19th STOC,* 1987, pp. 218-229.

[18] 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 *28th FOCS,* 1987.

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

[20] S. Goldwasser and S. Micali, Probabilistic encryption, *JCSS,* Vol. 28, No 2, April 1984, pp. 270-299.

[21] J. Hastad, R. Impagliazzo, L. Levin, and M. Luby, "Construction of a Pseudo-Random Generator from One-Way Function", to appear in SIAM J. on Computing. previous versions: FOCS 89 and STOC 90.

[22] E. Kushilevitz, and A. Rosén, "A Randomness-Rounds Tradeoff in Private Computation", CRYPTO-94, LNCS 839, pp. 397-410, 1994.

[23] E. Kushilevitz, S. Micali, and R. Ostrovsky, "Reducibility and Completeness in Multi-Party Private Computations", Proc. of 35th FOCS, 1994, pp. 478-489. (full version joint, with J. Kilian to appear in SICOMP).

[24] E. Kushilevitz, R. Ostrovsky, and A. Rosén, "Characterizing Linear Size Circuits in Terms of Privacy", Invited paper to the Journal of Computer and System Sciences special issue for STOC 96. Appeared in Vol 58, December 1998. Preliminary version in the Proc. of 28th STOC, pp. 541–550, 1996.

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

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

[27] T. Rabin and M. Ben-Or, "Verifiable Secret Sharing and Multi-party Protocols with Honest Majority", *21st STOC,* 1989, pp. 73-85.

[28] A. Yao, "theory and applications of trapdoor functions", In *Proc. 23rd Annual Symp. on Foundations of Computer Science*, pages 80–91. IEEE, 1982.

[29] A. Yao, "Protocols for Secure Computation", In *Proc. 23rd Annual Symp. on Foundations of Computer Science*, pages 160–164. IEEE, 1982.

[30] A. Yao, "How to generate and exchange secrets", In *Proc. 27th Annual Symp. on Foundations of Computer Science*, pages 162–167. IEEE, 1986.