The Round Complexity of Secure Protocols

(Extended Abstract)

Donald Beaver*
Harvard University

Silvio Micali†
MIT

Phillip Rogaway†
MIT

Abstract

In a network of n players, each player i having private input x_i, we show how the players can collaboratively evaluate a function $f(x_1, \ldots, x_n)$ in a way that does not compromise the privacy of the players' inputs, and yet requires only a constant number of rounds of interaction.

The underlying model of computation is a complete network of private channels, with broadcast, and a majority of the players must behave honestly. Our solution assumes the existence of a one-way function.

1 Introduction

Secure function evaluation. Assume we have n parties, $1, \ldots, n$; each party i has a private input x_i known only to him. The parties want to *correctly* evaluate a given function f on their inputs, that is to compute $y = f(x_1, \ldots, x_n)$, while maintaining the *privacy* of their own inputs. That is, they do not want to reveal more than the value y implicitly reveals. *Secure function evaluation* consists of distributively evaluating a function so as to satisfy both the correctness and privacy constraints. This task is made particularly difficult by the fact that some of the players may be maliciously faulty and try to cooperate in order to disrupt the correctness and the privacy of the computation.

Secure function evaluation arises in two main settings. First, in fault-tolerant computation. In this setting correctness is the main issue: we insist that the values a distributed system returns are correct, no matter how some components in the system fail. However, even if one is solely interested in correctness, privacy helps to achieve it most strongly: if one wants to maliciously influence the outcome of an election, say, it is helpful to know who plans to vote for whom. Second, secure function computation is central to protocol design, as the correctness and privacy of *any* protocol can be reduced to it. Here, as people may be behind their computers, correctness and privacy are equally important.

The first general solution for secure function evaluation was found by Yao [Ya86] for the two-party case, and by Goldreich, Micali and Wigderson [GMW87] for the multiparty case. Many other protocols for the multiparty case have been found since. In particular, the protocols of Ben-Or, Goldwasser and Wigderson [BGW88], Chaum, Crépeau and Damgård[CCD88], and Rabin and Ben-Or [RB89], succeed in defeating the influence of bad players without making use of cryptography, assuming that the privacy of communication among players is guaranteed. Other general protocols with different and interesting properties include [GV87, CR87, CDG87, GHY87, Be88, BG89, Ch89].

The GMW paradigm. In the above multiparty protocols, the underlying notions of security are often quite different, and so are the assumed communication models. Nonetheless, all of them follow the same paradigm of [GMW87] that we now describe.

There are three stages. In the first stage, each player *shares* the bits of his private input. Sharing a bit b entails breaking b into n "shares," $b_1, \ldots, b_n$, and giving share b_i to player i. For some parameter t, $t < n/2$, we require that no t players get information about b from their pieces; and yet, b is recoverable, and is *known* to be recoverable, given the cooperation of the $n - t$ good players—even if the t bad players try to obstruct b's recovery, or try to alter the recovered value. The value b which a player has effectively "committed to" is independent of the values that honest players may concurrently be committing to.

After the sharing stage, a *computation* stage follows, in which each player, given his own shares of $x_1, \ldots, x_n$, computes his own share of $f(x_1, \ldots, x_n)$. To accomplish this, the function f to be evaluated is represented by a

*Aiken Computation Laboratory, Harvard University, Cambridge, MA 02138. Supported in part by NSF grant CCR-870-4513.

†MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139. Supported by ARO grant DAAL 03-86-K-0171 and NSF grant CCR-8719689.

Boolean circuit, C. Thus, in Stage 1, each player gets his shares of the values along the input wires of C. In Stage 2, for each gate of the circuit, the parties compute shares of the value of the output wire from shares of the values of the input wires in a privacy-preserving manner. (Revealing the "incoming" shares for a gate will certainly allow the parties to compute its "outgoing" shares, but this will not preserve privacy. Even the output value of a single, internal gate constitutes "additional" information that must not be revealed.) This privacy-preserving computation, for a general gate, employs interaction. In this way, the parties interact, working their way up the circuit, from leaves to root, and eventually hold shares for the value corresponding to the output wire of C.

In the third and final stage, the output value of C is recovered by the players.

The problem. In view of this brief description, it can be seen that all of these protocols for secure multiparty function evaluation run in unbounded "distributed time," that is, using an unbounded number of rounds of communications. Even though the interaction for each gate can be implemented in a way that requires only a constant number of rounds, the total number of rounds will still be linear in the depth of the underlying circuit.

For many concrete computations, the resulting number of rounds would be prohibitive; in distributed computation, the number of rounds is generally the most valuable resource.

Bar-Ilan and Beaver [BB89] were the first to investigate reducing the round complexity for secure function evaluation. They exhibited a non-cryptographic method that always saves a logarithmic factor of rounds (logarithmic in the total length of the players' inputs), while the total amount of communication grows only by a polynomial factor. Alternatively, they show that the number of rounds can be reduced to a constant, but at the expense of an exponential blowup in the message sizes. We insist that the total amount of communication be polynomially bounded. While their result shows that the depth of a circuit is not a lower bound for the number of rounds necessary for securely evaluating it, the savings is far from being substantial in a general setting. Thus, the key question is:

> *How many rounds are necessary to securely evaluate a circuit while keeping the amount of communication polynomial in the size of the circuit?*

Our Result. Many of us believed that more complicated functions (i.e., those "with deeper circuits") required more rounds for secure evaluation. In this paper we show that, using cryptography, this is not the case.

Main Theorem (Informal version.) *There exists a cryptographic protocol that allows n players, the majority of whom are honest, to evaluate any circuit securely. The protocol uses a constant number of rounds and a polynomial amount of communication. The protocol works on a complete network (with private channels and supporting broadcast) and with any one-way function. The protocol tolerates any polynomial-time dynamic adversary.*

In other words, we prove that, with respect to secure function evaluation, interaction is like an atom. Without interaction secure function evaluation is impossible; but with a tiny bit of interaction, it is fully possible. A more formal version will be given by Theorem 3, but even now we would like to emphasize that our result is largely independent of the underlying communication model.

The model of computation. We have stated our main theorem assuming a rich model—a complete network of private channels, supporting broadcast—but analogous results hold under more restricted models. For example, if one increases the cryptographic assumption from a one-way function to public-key cryptography, then constant-round secure computation is possible on a network supporting only broadcast. Similarly, for $t < n/3$, the broadcast channels can be dispensed with by using a constant expected time Byzantine agreement protocol.

Security. To describe the security that our protocol achieves, we must describe the type of adversary that we are capable of defeating, and in what sense the adversary is defeated.

These are non-trivial matters. Many valuable notions of security have been proposed and used in the literature. In this abstract, we adopt notions of security due to Kilian, Micali, and Rogaway [KMR90].

We believe their formalization to be the "right one," and present it in Section 2. However, we make no attempt to compare this notion with previous ones, or to support the above claim.

A bird's-eye view of our solution

Our method can be described as finding the right way to generalize the older two-party protocol of Yao [Ya86]. His result has been used within the GMW paradigm for computing the desired "outgoing shares" of each gate from its "incoming shares" by engaging in many, *suitably chosen,* two-party computations. This use, however, leads to an unbounded number of rounds. We, instead, modify the construction "from the inside," generalizing it to many parties, but preserving the constant round complexity.

At a very high level, the idea is to use interaction to construct a *common* "garbled circuit," along with a set of "garbled inputs" for this circuit. Then, each individual player evaluates the garbled circuit, without interacting with other players. The intermediate information that this garbled circuit produces is meaningless to the players, but the output of the circuit is intelligible and is guaranteed to be correct.

The circuit is scrambled in a very special way, so to allow the players to perform the brunt of the scrambling locally, rather than use intensive interaction to simulate this computation step-by-step. Still, the requisite local computation is not excessive: each player will invest roughly as much time as is needed to evaluate the function f without any privacy constraints.

The common garbled circuit is constructed by joining various pieces, each piece being contributed by an individual player. Of course, nobody is trusted to contribute a correct piece, so each player uses interaction to prove to the "community" that he has done his work correctly. As usual, verification is simpler than computation, and correctness of very deep circuits (evaluated locally by individual players) can be verified by small, shallow circuits. These can be evaluated securely in a constant number of rounds using the gate-by-gate approach of previous protocols. In the end, the community can be certain that it is issuing a correct garbled circuit, which has been found with very little interaction.

We will see what this local and joint computation looks like after we describe our goal more formally.

2 Preliminaries

For this abstract, we restrict our definitions to finite functions. The definitions easily extend to handle, say, function families or probabilistic functions.

The symbol n will always denote the number of players, and the symbol ℓ, the length of each individual input and output. Thus the players are trying to securely evaluate a function $f : (\Sigma^\ell)^n \to (\Sigma^\ell)^n$, each player i learning $f_i(\vec{x})$. Under these conditions, f can be represented by a Boolean circuit C. The goal of this section is to define the notion of securely computing f.

Notation

Let $\Sigma = \{0,1\}$, let Λ be the empty string, and let 1^k be k written in unary. For $x, y \in \Sigma^*$, $|x| = |y|$, $x \oplus y$ is the exclusive-or (XOR) of the these strings. If $x = a_1 \cdots a_n$ is a string, $x[i:j]$ denotes the substring $a_i \cdots a_j$. For x and y strings, $x \circ y$ denotes their concatenation. When b is a bit, $\overline{b}$ is its complement.

For $T \subseteq \{1, \ldots, n\}$, we write $\overline{T}$ for $\{1, \ldots, n\} - T$. If $\vec{x} = (x_1, \ldots, x_n)$ and $T \subseteq \{1, \ldots, n\}$, $\vec{x}_T = \{(i, x_i) : i \in T\}$ (i.e., $\vec{x}_T$ keeps track off the indices as well as the values). If $\vec{x} = \{x_1, \ldots, x_n\}$, $\vec{x}' = \{x'_1, \ldots, x'_n\}$, and $T \subseteq \{1, \ldots, n\}$, then $\vec{x}_{\overline{T}} \cup \vec{x}'_T$ can be regarded as an n-vector, $(y_1, \ldots, y_n)$, where $y_i = x_i$ if $i \in T$ and $y_i = x'_i$ otherwise.

For $f : (\Sigma^\ell)^n \to (\Sigma^\ell)^n$ and $T \subseteq \{1, \ldots, n\}$, $f_T(\vec{x})$ is the function of $\vec{x}$ defined by $f_T(\vec{x}) = (f(\vec{x}))_T$, (i.e., $f_T(\vec{x})$ is the tagged T-coordinates of the image of $\vec{x}$ under f).

The notation $[x \leftarrow X; y \leftarrow Y; \ldots : S(x, y, \ldots)]$ denotes the distribution induced by the probabilistic algorithm S when its inputs are drawn by performing the indicated experiment (in the order specified).

A *simulator* is a probabilistic polynomial-time algorithm.

Basic definitions

A function $\epsilon : \mathbf{N} \longrightarrow \mathbf{R}$ is *negligible* if $\epsilon(k) = k^{-\omega(1)}$, (i.e., if $\epsilon(k)$ vanishes faster than the inverse of any polynomial). An *ensemble* $\{\mathcal{A}_k\}$ is a family of probability measures on Σ^* for which there is a polynomial q such that only strings of length $\leq q(k)$ have nonzero probability in $\mathcal{A}_k$. For $\mathcal{A}$ taken from some ensemble and C a Boolean circuit (with sufficiently many inputs), $p^C_{\mathcal{A}}$ is the probability that C outputs 1 on input drawn according to $\mathcal{A}$. Ensembles $\{\mathcal{A}_k\}$ and $\{\mathcal{B}_k\}$ are *computationally indistinguishable* if for any polynomial-size circuit family $\mathcal{C} = \{C_k\}$, the function $\epsilon(k) = |p^{C_k}_{\mathcal{A}_k} - p^{C_k}_{\mathcal{B}_k}|$ is negligible; these ensembles are *statistically indistinguishable* if $\epsilon(k) = \max_{S_k \subseteq \Sigma^*} |\Pr_{\mathcal{A}_k}[S_k] - \Pr_{\mathcal{B}_k}[S_k]|$ is negligible.

Model of computation

There are a variety of models of computation under which secure distributed computation has been considered. To describe and develop our results, we adopt a synchronous model of computation with both private channels and broadcast; we call this the *standard model*. This is the usual model for maximally fault-tolerant non-cryptographic computation, and is roughly described below.

We envisage a network of processors whose computation is controlled by a common clock ticking at time $0, 1, 2, \ldots$. Local computation is "instantaneous" compared to the ticking of the clock. *Round* i is the interval of time between clock tick i and $i+1$.

There is a read-only *common input* tape that initially contains a string 1^k and is readable by all processors. The value k is called the *security parameter.*

Each processor i has a private read/write *work* tape (initially containing the string Λ); a private read-only *input* tape (initially containing x_i); a private write-only *output* tape (initially containing the string Λ); and some additional *communication* tapes described below.

Between each pair of processors i and j there is a *private channel* $i \to j$ for processor i to securely send messages to processor j. That is, this tape is exclusive-write for i and exclusive-read for j.

Each processor i also has a *broadcast channel.* This is a tape that is exclusive-write for i and readable by all processors j.

Proper conventions are assumed so that, at round r and for each player i there is a well-defined message (possibly the empty message) that i is broadcasting (i.e., writing on its broadcasting channel) in this round; and, for each pair of players i and j, there is a unique message that i is securely sending to j (i.e., is writing on channel $i \to j$) in this round.

It is understood that channels are properly "labeled" so that the recipient "knows" who is the sender of a message.

Each processor has access to a fair coin; that is, it can enter a distinguished state from which it enters either of two successor states with equal probability.

A model of computation defined as above but supporting only broadcast for communication is called a *broadcast model.*

Processor i of the network runs program P_i; the n-tuple of programs $\mathcal{P} = (P_i, \ldots, P_n)$ is called a *protocol.* For each player i, i's *history* in the execution of a protocol consists of everything that player i has had access to: its private and common input, all broadcast messages, all messages received along private channels to i, and the coins that processor i has flipped.

The Adversary

For simplicity, we assume a uniform adversary. In this case, an *adversary* A is a probabilistic polynomial-time algorithm. An adversary *acts* on a network, *corrupting* processors. A processor is called *bad* after it has been corrupted by A, *good* if it has not yet been corrupted by A.

When the adversary corrupts a processor, she learns its entire internal configuration, will read all future messages sent to it, and will choose what messages it will send in the future—essentially, the adversary totally subsumes the corrupted processor.

The adversary has the ability to corrupt processors in a *dynamic* fashion. At the beginning of each round, the adversary may choose to corrupt some new processors. By doing so she learns (in particular) all the messages sent to it in the current round. Having done this, the adversary may decide whether to corrupt another new processor, and so on, until she decides not to corrupt any more processors during this round. At this point she composes all the messages from the bad processors to the good ones for the current round. These messages (and the ones between good processors) are guaranteed to be delivered by the end of the round.

A *t-adversary* is an adversary that corrupts at most t processors.

The protocol starts at round 1 and terminates at the first round by which all good processors have terminated. The adversary terminates by the end of this round, as well.

When the adversary terminates, the nonblank portion of her output tape contains some string. Fixing $\mathcal{P}$, k, and $\vec{x} = (x_1, \ldots, x_n)$, adversary A defines a probability space $\mathbf{VIEW}_A^k(\vec{x})$, the space of "adversary outputs." Without loss of generality, the adversary's output, or *view,* is an encoding of her history (that is, an encoding of her coin flips, what messages she received when, etc.)

When the protocol terminates, each good player has output a certain value. We denote by $\mathbf{OUTPUT}_A^k(\vec{x})$ the values output by the good players, tagged by the good players' identity; that is, a point from this probability space is a tagged vector $\vec{y}_G$.

Security

As with zero-knowledge interactive proofs, the notion of privacy involves the approximability of the adversary's view by a simulator. The question is: the view should be approximable given *what*?

We now state definitions from [KMR90].

A *t-bounded $(\vec{x}, f)$-oracle* behaves as follows. It accepts two types of queries, called *component* queries and *output* queries.

A *component query* is an integer i, $1 \leq i \leq n$. It is answered by x_i if t or fewer component queries have been made so far, and no output query has been made so far; it is answered by $(x_i, f_i(\vec{x}_{\overline{T}} \cup \vec{x}'_T))$ if t or fewer component queries have been made so far, and the proper output query previously made was $\vec{x}'_T$. Additional or improper component queries are not answered.[1]

An *output query* is a tagged vector $\vec{x}'_T$. It is answered by $f_T(\vec{x}_{\overline{T}} \cup \vec{x}'_T)$ if T consists precisely of the component queries made so far, and if this is the first output query. Additional or improper output queries are not answered.

Note that we permit component queries to follow the output query, as long as the total number of component queries is bounded by t.

We consider simulators S having access to a t-bounded $(\vec{x}, f)$-oracle. The output of such a simulator is denoted OUTPUT $S^{O_t(\vec{x},f)}(1^k)$; this is a probability space. We let QUERIES $S^{O_t(\vec{x},f)}(1^k)$ denote the indices i for which there was *never* a component query, together with the (single) output query; a point from this probability space is written $(G, \vec{x}'_T)$. Because of the possibility of component queries following the output query, G may be a proper subset of $\overline{T}$.

We are now ready to define the notion of security.

[1] For the purpose of computing string-valued functions, this can be simplified: a component query i returns x_i if there have been t or fewer component queries so far.

Definition 1 *Fix $f : (\Sigma^\ell)^n \to (\Sigma^\ell)^n$. Protocol $\mathcal{P}$ t-securely computes f if for all t-adversaries A there exists a simulator S (capable of querying a t-bounded oracle), such that*

- `(Privacy)` *for all $\vec{x} \in (\Sigma^\ell)^n$, the k-parameterized ensembles*

  $$\mathbf{VIEW}_A^k(\vec{x})$$

  and

  $$\text{Output } S^{O_t(\vec{x},f)}(1^k)$$

  are computationally indistinguishable; and,

- `(Correctness)` *for all $\vec{x} \in (\Sigma^\ell)^n$, the k-parameterized ensembles*

  $$\mathbf{OUTPUT}_A^k(\vec{x})$$

  and

  $$[(G, \vec{x}'_T) \leftarrow \text{Queries } S^{O_t(\vec{x},f)}(1^k) : f_G(\vec{x}_{\overline{T}} \cup \vec{x}'_T)]$$

  are statistically indistinguishable.

Remarks. This notion of security implicitly enforces *independence*—that is, the adversary's inability to influence the outcome of the protocol on the basis of values privately held by uncorrupted players. It also implicitly enforces *fairness*—the adversary's inability to obtain more information than the good players do. Fairness is generally not an issue in multiparty protocols with honest majority, but implicitly enforcing independence through the correctness constraint is novel.

Strong Security

The above definition of security essentially says that any adversary has *its own* simulator. We will actually achieve something stronger. Namely, that there will exist a *single* simulator that works for *any* adversary A with which it interacts in a special fashion. The simulator creates a "virtual world" and has A act in it. In this virtual world, A sees an indistinguishable view from that which it would see when interacting in the real network. Moreover, the simulator does not monitor the internal computation of A, nor does it choose the coin flips for A to use. The simulator outputs what adversary A in the virtual execution outputs, and the simulator asks component queries only for those processors "corrupted" during the simulation. (Details on the interaction between the simulator and the adversary will appear in the final paper.)

We call this notion of security *strong security.* Not only is strong security what is actually achieved by our protocol, it is a desirable end goal in itself.

3 The Protocol

Building blocks

This subsection describes some "well known" results which we require to describe our protocol.

Verifiable Secret Sharing. Let us informally state the notion of verifiable secret sharing (VSS), originally introduced by [CGMA85]. This is a way to secretly commit to a value. A distinguished player D (the *dealer*) has a private input bit b. At the end of an execution of a VSS protocol tolerating t faults, each player i holds his own private *share* b_i of bit b. We require three properties to hold:

1. With probability $> 1 - 1/2^k$, there exists a unique value b' such that if the good players broadcast their private shares, each good player will *locally* compute b' from the broadcasted values—regardless of the values broadcast by the bad players.
2. If the dealer is good, $b' = b$.
3. If the dealer is good, the view of any t-adversary who does not corrupt D is independent of b.

We say that player D *commits* to bit b if VSS is executed with dealer D and private input b. At the end of such an execution, b is said to be a *committed bit.* More generally (and more informally), any bit represented by shares as above, is called a *shared bit*, even if its shares do not originate from a given dealer executing VSS. We will also speak of shared strings—meaning that each bit of the string is a shared bit.

We say that a committed value is *privately revealed to player i* if all good players send to i their private shares of this value along the appropriate private channels; in this way, i alone recovers the value. A committed value is *publicly revealed* if all good players broadcast their private shares of the value; in this way, all the players recover the value.

Computing on shared bits. [RB89] have shown that in our model of computation, VSS is implementable in a constant number of rounds and tolerating any computationally unbounded t-adversary, for any $t < n/2$. Furthermore, they (as well as [Be88]) have shown that any Boolean circuit C with bounded fan-in and depth d can be evaluated on shared values securely and secretly—that is, the inputs to the computation are shared bits, and the result also is a shared bit (which will be revealed only if so wanted). This computation requires $O(d)$ rounds and communication complexity local computation polynomial in k and $|C|$ (where $|C|$ is the size of the circuit C).

While $O(d)$ rounds are sufficient for securely and secretly evaluating any depth-d circuit, this does not mean that one cannot get by with fewer rounds. In particular, a closer look at the underlying method shows that (when the underlying field of the VSS protocol is taken to have characteristic 2) the XOR function (on any number of bits) can be securely and secretly evaluated in a constant number of rounds and polynomial communication.

Collaborative coin flipping. This, in turn, allows the players to collaboratively obtain shared, random bits (known to no one) in a constant number of rounds: each player commits a random bit, and the XOR of these committals is securely and secretly evaluated. This shared value is the shared random bit.

Proving assertions to the community. It is useful to conceptualize as a primitive "prove a given assertion to the community." In particular, players in our protocol will be required to prove that they have done some local computation correctly. Even though this local computation may require deep circuits, the proof that it has been done correctly will be fast:

Proposition 2 *Let $\mathcal{G} : \Sigma^a \to \Sigma^b$ be a function represented by a circuit C, and let let $\sigma_1, \cdots, \sigma_a$ be shared bits which have been previously opened to some player i. Then player i can commit the bits of $\mathcal{G}(\sigma_1 \cdots \sigma_a)$, and prove to the community that the committals represent the correctly computed value of $\mathcal{G}$ on $\sigma_1 \cdots \sigma_a$. The proof reveals nothing in the information-theoretic sense and requires only a constant number of rounds and a polynomial amount of communication (in the size of C and in the security parameter).*

To prove that $\mathcal{G}$ was evaluated correctly, player i will commit (in addition to the function value) a "certificate" $(c_1, \ldots, c_b)$ to demonstrate his claim. The certificate consists of the Boolean values at all (internal) wires of the circuit C that evaluates $\mathcal{G}$. The community, acting as the "verifier," checks the condition "the values specified on the circuit's wires are correct for each gate." This is just the conjunct of constant depth predicates on committed values, and so the predicates can be securely evaluated and publicly revealed in a constant number of rounds. The players compute the AND on their own—that is, they accept the proof only if they believe the circuit is locally correct everywhere.

Pseudorandom generators. Our protocol makes use of a "perfect" pseudorandom generator. This is a deterministic polynomial-time algorithm stretching a short, truly random input string (the "seed") to a longer, pseudorandom string. It is required that the ensemble of output strings be computationally indistinguishable from truly random strings of the same length. This notion is due to Blum and Micali [BM82] and Yao [Ya82b], who showed that such generators exist under suitable complexity assumptions.

Complexity assumption

The only complexity assumption used for our main theorem is the existence of a pseudorandom generator.

Recently, Impagliazzo, Levin and Luby [ILL89] and Håstad [Ha90] have shown this assumption to be equivalent to the existence of a one-way function.

The structure of our protocol

The protocol we construct has two phases. The first phase is a non-cryptographic protocol. The only information revealed to players in the first phase of the protocol are some random strings.

The second phase of the protocol consists of only a single round. During this round, the players publicly reveal information which was computed during the first phase of the protocol. The information that is revealed is precisely the "garbled circuit" and its "garbled input" mentioned in the introduction. Though what is revealed contains information which betrays the players' private inputs, the information is nonetheless unusable (given the cryptographic assumption) with respect to polynomial-time computation. After recovering the garbled circuit and garbled input, the players evaluate it individually, without interaction, obtaining the desired result.

Description of our protocol

Obvious cheating. As we have said, a protocol is a set of instructions to be followed by the good players. As we also said, though, the adversary may have some players deviate from their prescribed program. The subtle adversary will do this without exposing the corrupted players. That is, though the corrupted players follow different instructions, the messages they send are not obviously dictated by the adversary. Of course, an adversary may not be subtle at all, and instruct a player to, say, send nothing when a message is expected, or to provide to the community a clearly fallacious "proof" of an assertion. This sort of behavior can be taken care of by proper conventions (deciding on default values, etc.). These conventions would complicate the description of our protocol, but without adding any particular insight. Thus, for the sake of simplicity and for focusing on the important issues, we first describe the protocol when no obvious cheating is detected, and later discuss the necessary additions.

Notation. We fix notation for the remainder of the paper. We let $\mathcal{G}$ denote a pseudorandom generator that stretches a k-bit string s to a $k+2nk$ bit string, and we define F, G and H to be the first k, next nk, and last nk bits produced by the generator, $F(s) = \mathcal{G}(s)[1:k]$, $G(s) = \mathcal{G}(s)[k+1:k+nk]$, and $H(s) = \mathcal{G}(s)[k+nk+1:k+2nk]$.

Without loss of generality, the circuit being evaluated consists solely of two-input gates. The circuit has W wires, which are labeled $0, 1, \ldots, W-1$. To simplify the exposition of the protocol, we will assume there is only a single output wire, which is wire $W-1$, and all players are to learn this bit. The protocol can readily be extended to handle the computation of functions $f : (\Sigma^\ell)^n \to (\Sigma^\ell)^n$, and to more general scenarios.

In the description that follows, the superscript i ranges over the players, $1 \le i \le n$. The subscript ω ranges over the wires, $0 \le \omega \le W-1$. Each wire ω has two indices associated with it, 2ω and $2\omega+1$. The subscript j ranges over such indices, $0 \le j \le 2W-1$. Phrases in quotes are comments.

Phase I

In this phase, the players compute the garbled circuit and the garbled input, leaving this information as shared values.

1. Each player shares his input bits. Let b_ω be the shared bit associated with each *input* wire ω of the circuit. "Each b_ω is a bit from some particular player's x_i value."

2. The players collaboratively flip $(2kn+1)W$ coins, which define

   (a) $2nW$ length-k strings, s_j^i, for $1 \le i \le n$, $0 \le j \le 2W-1$, and

   (b) bits $(\lambda_0, \ldots, \lambda_{W-1})$.

   "The strings s_j^i are called *seeds*, while the strings $s_j^1 \circ \cdots \circ s_j^n$ are called *super-seeds*. Each super-seed is associated with a bit, as follows: $s_{2\omega}^1 \circ \cdots \circ s_{2\omega}^n$ is associated with λ_ω, while $s_{2\omega+1}^1 \circ \cdots \circ s_{2\omega+1}^n$ is associated with $\overline{\lambda_\omega}$. The value λ_{W-1} will be publicly revealed in Step 6."

3. Each player i gets seeds s_j^i, $0 \le j \le 2W-1$, privately revealed to him.

4. The strings s_j^i revealed to player i are used by player i as the seeds of the pseudorandom generator $\mathcal{G}$. Thus player i locally computes $f_j^i = F(s_j^i)$, $g_j^i = G(s_j^i)$, and $h_j^i = H(s_j^i)$, for $0 \le j \le 2W-1$. Player i then commits each f_j^i, g_j^i, and h_j^i, and proves to the community that these committals were computed correctly. "The strings $f_j^1 \circ \cdots \circ f_j^n$ are called *wire-labels*. They will be publicly revealed in Step 6. The other values committed here will never be revealed."

5. "The players securely and secretly compute some simple functions on the shared values. These values will be opened in Step 6. Namely:"

   (a) For each input wire ω of the circuit, the players securely and secretly compute the *garbled input* for this wire, defined by

   $$\sigma_\omega^1 \circ \cdots \circ \sigma_\omega^n = s^1_{2\omega+(b_\omega \oplus \lambda_\omega)} \circ \cdots \circ s^n_{2\omega+(b_\omega \oplus \lambda_\omega)},$$

   where $|\sigma_j^1| = \cdots = |\sigma_j^n| = k$. "Thus the garbled input consists of one super-seed for each input wire of the circuit. Which of the two super-seeds associated with input wire ω—the super-seed indexed by 2ω or the super-seed indexed by $2\omega+1$—is determined by λ_ω and b_ω."

   (b) For each gate g of the circuit, the players securely and secretly compute the *gate-labels* for gate g. These consist of four strings, A_g, B_g, C_g, and D_g. These strings are defined as follows: If gate g computes the function $\otimes$, the left wire is wire α, the right wire is wire β, and and the output wire is wire γ, $0 \le \alpha, \beta, \gamma \le W-1$), then, writing $a = 2\alpha$, $b = 2\beta$, and $c = 2\gamma$,

   $$\begin{aligned}
   A_g &= g_a^1 \oplus \cdots \oplus g_a^n \oplus g_b^1 \oplus \cdots \oplus g_b^n \\
   &\quad \oplus \begin{cases} s_c^1 \circ \cdots \circ s_c^n & \text{if } \lambda_\alpha \otimes \lambda_\beta = \lambda_\gamma \\ s_{c+1}^1 \circ \cdots \circ s_{c+1}^n & \text{otherwise} \end{cases} \\
   B_g &= h_a^1 \oplus \cdots \oplus h_a^n \oplus g_{b+1}^1 \oplus \cdots \oplus g_{b+1}^n \\
   &\quad \oplus \begin{cases} s_c^1 \circ \cdots \circ s_c^n & \text{if } \lambda_\alpha \otimes \overline{\lambda_\beta} = \lambda_\gamma \\ s_{c+1}^1 \circ \cdots \circ s_{c+1}^n & \text{otherwise} \end{cases} \\
   C_g &= g_{a+1}^1 \oplus \cdots \oplus g_{a+1}^n \oplus h_b^1 \oplus \cdots \oplus h_b^n \\
   &\quad \oplus \begin{cases} s_c^1 \circ \cdots \circ s_c^n & \text{if } \overline{\lambda_\alpha} \otimes \lambda_\beta = \lambda_\gamma \\ s_{c+1}^1 \circ \cdots \circ s_{c+1}^n & \text{otherwise} \end{cases} \\
   D_g &= h_{a+1}^1 \oplus \cdots \oplus h_{a+1}^n \oplus h_{b+1}^1 \oplus \cdots \oplus h_{b+1}^n \\
   &\quad \oplus \begin{cases} s_c^1 \circ \cdots \circ s_c^n & \text{if } \overline{\lambda_\alpha} \otimes \overline{\lambda_\beta} = \lambda_\gamma \\ s_{c+1}^1 \circ \cdots \circ s_{c+1}^n & \text{otherwise} \end{cases}
   \end{aligned}$$

Phase II

In this phase, the players publicly reveal the garbled circuit and the garbled input, and they evaluate it on their own.

6. The players publicly reveal

   (a) "the bit associated with the output wire," λ_{W-1}, computed in Step 2;

   (b) the gate-labels f_j^i, $1 \le i \le n$, $0 \le j \le 2W-1$, committed in Step 4;

   (c) the garbled input $\sigma_\omega^1 \circ \cdots \circ \sigma_\omega^n$ associated with each input wire ω of the circuit, computed in Step (5a); and

   (d) the gate-labels A_g, B_g, C_g, and D_g associated with each gate g, computed in Step (5b).

   "Together, the strings revealed in Steps 6(a), 6(b), and 6(d) constitute the *garbled circuit,* while those revealed in Step 6(c) are the *garbled input.*"

7. "Each player evaluates the garbled circuit by himself, learning a super-seed for each wire. The super-seed of the root determines the output of the circuit according to the value of λ_{W-1}. To evaluate the circuit:"

   Initially, for each input wire ω, you *hold* $s^1_{2\omega} \circ \cdots \circ s^n_{2\omega} = \sigma^1_\omega \circ \cdots \circ \sigma^n_\omega$ if $F(\sigma^1_\omega) = f^1_{2\omega}$, and you *hold* $s^1_{2\omega+1} \circ \cdots \circ s^n_{2\omega+1} = \sigma^1_\omega \circ \cdots \circ \sigma^n_\omega$ otherwise.

   Suppose inductively that you hold $s^1_{a+p} \circ \cdots \circ s^n_{a+p}$ for the left input wire of a gate g, and you hold $s^1_{b+q} \circ \cdots \circ s^n_{b+q}$ for the right input wire, where a and b are even and $p, q \in \{0,1\}$. Suppose the output wire of the gate is wire γ, $0 \le \gamma \le W-1$. Then, for $1 \le i \le n$, set $g^i_{a+p} = G(s^i_{a+p})$, $h^i_{a+p} = H(s^i_{a+p})$, $g^i_{b+q} = G(s^i_{b+q})$, and $h^i_{b+q} = H(s^i_{b+q})$, and compute $\sigma_\gamma = \sigma^1_\gamma \circ \cdots \circ \sigma^n_\gamma$ ($|\sigma^1_\gamma| = \cdots = |\sigma^n_\gamma| = k$) according to

$$\sigma_\gamma = \begin{cases} g^1_{a+p} \oplus \cdots \oplus g^n_{a+p} \oplus g^1_{b+q} \oplus \cdots \oplus g^n_{b+q} \oplus A_g & \text{if } p = 0 \text{ and } q = 0 \\ h^1_{a+p} \oplus \cdots \oplus h^n_{a+p} \oplus g^1_{b+q} \oplus \cdots \oplus g^n_{b+q} \oplus B_g & \text{if } p = 0 \text{ and } q = 1 \\ g^1_{a+p} \oplus \cdots \oplus g^n_{a+p} \oplus h^1_{b+q} \oplus \cdots \oplus h^n_{b+q} \oplus C_g & \text{if } p = 1 \text{ and } q = 0 \\ h^1_{a+p} \oplus \cdots \oplus h^n_{a+p} \oplus h^1_{b+q} \oplus \cdots \oplus h^n_{b+q} \oplus D_g & \text{if } p = 1 \text{ and } q = 1 \end{cases}$$

   You are now *holding* $s^1_{2\gamma} \circ \cdots \circ s^n_{2\gamma} = \sigma_\gamma$ if $F(\sigma^1_\gamma) = f^1_{2\gamma}$, and you are holding $s^1_{2\gamma+1} \circ \cdots \circ s^n_{2\gamma+1} = \sigma_\gamma$ otherwise.

   When you come to hold $s^1_{2W+p} \circ \cdots \circ s^n_{2W+p}$, output $p \oplus \lambda_{W-1}$ on your private output tape.

This completes the description of the protocol.

A very simple example

The following example may be useful in understand the preceding protocol. Suppose that there are $n = 3$ players, P_1, P_2 and P_3, and each player holds a single bit, $x_1 = 0$, $x_2 = 1$ and $x_3 = 1$, respectively. The players wish to compute function $f(x_1, x_2, x_3) = x_1 x_2 \vee x_3$. They know the fixed circuit C of Figure 1 which evaluates this function (including the labeling of wires and gates).

Figure 2 depicts the garbled circuit and the garbled input collaboratively computed by the community, if $(\lambda_0, \lambda_1, \lambda_2, \lambda_3, \lambda_4) = (0, 1, 0, 0, 1)$. The values of the gate-labels are shown in Figure 3.

What to do when cheating is detected

The protocol we have just given, with simple modifications to deal with ostensibly cheating players, provides us with secure constant round computation in the standard model. We now sketch the main modifications.

Figure 1: The circuit C to be securely evaluated.

If a fault is detected before the completion of Step 1, we replace the faulting player's inputs with "default" values. This certainly incorporates scenarios such as the default not being the value on any "proper" input, and the output including a list of players faulting before they have committed their input bits.

For faults occurring after Step 1, the computation proceeds using the input values already committed by the offending player. There is no need to reveal a bad players' private input in order to accomplish this. Only the bad player's seeds need to be known by the good players—and even these need not be known by the good players if the fault occurs after Step 4. However, for faults occurring before the completion of Step 4, the good players effectively "fill in" for the bad player. Random seeds which would otherwise be issued to the bad player are revealed to all of the players instead. In this way, the parties select a garbled circuit and garbled input from the same distribution as if all players had played honestly after the committal in Step 1.

Faults detected in Steps 5 and 6 need result in no "punitive" action whatsoever (that is, nothing of the faulting player needs be divulged).

Figure 2: The garbled circuit, garbled input, and λ_ω-values for the circuit of Figure 1, where $(\lambda_0, \lambda_1, \lambda_2, \lambda_3, \lambda_4) = (0, 1, 0, 0, 1)$, and $(b_0, b_1, b_2) = (0, 1, 1)$.

4 Proving Security

A full proof of the security for our protocol is very involved and not well suited to an abstract. In this section, we formally state our main theorem and briefly discuss the nature of a proof.

Theorem 3 *Assume the standard model, $t < n/2$, $f : (\Sigma^\ell)^n \to (\Sigma^\ell)^n$ a function realized by a circuit C. Assume that a pseudorandom generator exists. Then there is a protocol $\mathcal{P}$ that strongly t-securely computes f in* `const` *rounds, where* `const` *is an absolute constant. Furthermore, a natural encoding of protocol $\mathcal{P}$ can be found by a fixed algorithm in time polynomial in $|C|$. Local computation in $\mathcal{P}$ is* $\texttt{poly}(|C|, k)$ *time bounded, where* `poly` *is a fixed polynomial.*

As mentioned in Section 1, corresponding results for other models of computation follow. For example, the result of Feldman [Fe88] implies that public-key encryption can be used to establish the analogous result for the broadcast model.

$$
\begin{aligned}
A_1 &= g_0^1 \oplus \cdots \oplus g_0^n \oplus g_2^1 \oplus \cdots \oplus g_2^n \oplus s_6^1 \circ \cdots \circ s_6^n \\
B_1 &= h_0^1 \oplus \cdots \oplus h_0^n \oplus g_3^1 \oplus \cdots \oplus g_3^n \oplus s_6^1 \circ \cdots \circ s_6^n \\
C_1 &= g_1^1 \oplus \cdots \oplus g_1^n \oplus h_2^1 \oplus \cdots \oplus h_2^n \oplus s_7^1 \circ \cdots \circ s_7^n \\
D_1 &= h_1^1 \oplus \cdots \oplus h_1^n \oplus h_3^1 \oplus \cdots \oplus h_3^n \oplus s_6^1 \circ \cdots \circ s_6^n \\
\\
A_2 &= g_6^1 \oplus \cdots \oplus g_6^n \oplus g_4^1 \oplus \cdots \oplus g_4^n \oplus s_9^1 \circ \cdots \circ s_9^n \\
B_2 &= h_6^1 \oplus \cdots \oplus h_6^n \oplus g_5^1 \oplus \cdots \oplus g_5^n \oplus s_8^1 \circ \cdots \circ s_8^n \\
C_2 &= g_7^1 \oplus \cdots \oplus g_7^n \oplus h_4^1 \oplus \cdots \oplus h_4^n \oplus s_8^1 \circ \cdots \circ s_8^n \\
D_2 &= h_7^1 \oplus \cdots \oplus h_7^n \oplus h_5^1 \oplus \cdots \oplus h_5^n \oplus s_8^1 \circ \cdots \circ s_8^n
\end{aligned}
$$

Figure 3: The gate-labels for the two gates of Figure 2.

Corollary 4 *Same as Theorem 3, but in the broadcast model and assuming a secure public-key cryptosystem.*

To give a second example, the result of Ben-Or and El-Yaniv [BE88] can be applied to give secure constant-round computation in the model with only private channels.

Corollary 5 *Same as Theorem 3, but for $t < n/3$ and in the model with only private channels (no broadcast).*

To prove Theorem 3 we must exhibit a simulator S which satisfies the privacy and correctness constraints. This is a very big job.

Privacy is the main issue. To achieve privacy, the simulator S must provide the adversary A with a computationally indistinguishable view from that which it would see when interacting in the real network. This is argued round-by-round, repeatedly showing how to extend the view that the simulator has created so far for the adversary to a view that works for one more round. The simulator must imitate broadcast messages and messages sent along private channels to players corrupted in the simulation, and the simulator must provide to the adversary a fake "state" for processors when they are corrupted by the adversary during the the simulation.

The proof mirrors the two phase structure of the protocol. The first phase of the protocol is argued to be strongly t-private in the "information-theoretic sense," and revealing nothing. Then we show how to extend this simulation for the one round of the second phase.

For that, we must show both how to simulate the players' final round of broadcasts, and also, how to simulate the players' private state for those players who are corrupted during the adversary's final round.

The former task is the more interesting. Proceeding intuitively, to simulate the player's final round of messages, we must insure that the garbled circuit and the garbled input released in Step 6 is drawn from a distribution that the simulator can approximate (up to computational indistinguishability) given the simulator's previous history. To do this, the simulator constructed for the first phase of the protocol, observing outgoing messages to at least $(n+1)/2$ players, "knows" the values $\vec{x}'_T$ that the adversary has effectively committed to. Likewise, the simulator will "know" the seeds issued to all of the players.

The simulator uses this information in constructing a "fake" garbled circuit and garbled input which it will (effectively) hand over to the adversary.

To construct a convincing garbled circuit and garbled input, an output oracle query of $\vec{x}'_T$ is made, and the simulator will learn (for a Boolean function) a bit b. Wire-labels are selected as the image under F of the seeds which the simulator already knows from the first phase of the simulation. Then, a "random path" through the garbled circuit is selected. This path (together with the seeds) determines which one of the four gate labels is to be "used" in evaluating the circuit, and what the garbled input is. The bit b is used to select λ_{W-1} so that the circuit (given the selected path) computes the bit b. All that remains unspecified are the three "unused" gate-labels associated with each gate; the simulator simply fills in random strings here.

Of course, it is far from clear that this "fake" garbled circuit and garbled input will in any sense fool the adversary. A key lemma to establishing this is that, for any $\vec{w} \in (\Sigma^\ell)^n$, the space of "real" garbled circuits and garbled inputs for $\vec{w}$ is computationally indistinguishable from the space of "fake" garbled circuits and garbled inputs which we have just specified—and this remains so, even if conditioned on some particular partial choice of seeds $\{s^i_j\}_{i\in T, 0\le j\le 2W-1}$, for $|T| \le n-1$.

5 Open Problems

Our cryptographic protocol for constant-round, secure function evaluation is *optimal* in the sense that it is based on the mildest possible cryptographic assumption, the existence of a one-way function, and some positive number of rounds is certainly required.

Let us conclude by stating what we believe to be the key open question in this area—namely,

> *Is there a constant-round, non-cryptographic protocol for secure function evaluation?*

That is, is there a constant-round protocol, in the standard model, that allows n computationally unbounded players to defeat a computationally unbounded adversary, while using only a polynomial amount of communication?

This problem is open even for the case of $t = 1$.

A partial answer to this question is now known, due to Beaver, Feigenbaum, Kilian, and Rogaway [BFKR89]: they show that *any* functions on $O((n \log n)/t)$ shared bits can be t-securely evaluated in a constant number of rounds by a network of n processors.

Acknowledgments

The various authors happily acknowledge discussions with Mihir Bellare, Shafi Goldwasser, Stuart Haber, Joe Kilian, Rafail Ostrovsky, Charlie Rackoff, and Moti Yung.

References

[BB89] J. Bar-Ilan and D. Beaver, "Non-Cryptographic Fault Tolerant Computing in a Constant Number of Rounds of Interaction," *Proc. of the 8th PODC* (1989), 201-09.

[Be88] D. Beaver, "Secure Multiparty Protocols Tolerating Half Faulty Processors," Harvard Technical Report TR-19-88, and in CRYPTO-89 Proceedings.

[Be90] D. Beaver, "Security, Fault-Tolerance, and Communication Complexity for Distributed Systems," Harvard University Ph.D. Thesis, 1990.

[BFKR89] D. Beaver, J. Feigenbaum, J. Kilian, and P. Rogaway, "Cryptographic Applications of Locally Random Reductions," Bell Laboratories Technical Memorandum, November 1989. Also a 1990 PODC submission, "Security with Low Communication Overhead."

[BG89] D. Beaver and S. Goldwasser, "Multiparty Computations with Faulty Majority," CRYPTO-89 Proceedings.

[BE88] M. Ben-Or and R. El-Yaniv, "Interactive Consistency in Constant Expected Time," manuscript, December 1988.

[BGW88] M. Ben-Or, S. Goldwasser and A. Wigderson, "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation," *Proc. of the 20th STOC* (1988), 1–10.

[BM82] M. Blum and S. Micali, "How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits," *SIAM J. of Computing*, 13:850–864 (1984), and FOCS 82.

[Ch89] D. Chaum, "The Spymasters Double-Agent Problem: Multiparty Computations Secure Unconditionally from Minorities and Cryptographically from Majorities," CRYPTO-89 Proceeding.

[CCD88] D. Chaum, C. Crépeau and I. Damgård, "Multiparty Unconditionally Secure Protocols," *Proc. of the 20th STOC* (1988), 11–19.

[CDG87] D. Chaum, I. Damgård and J. van de Graff, "Multiparty Computations Ensuring the Privacy of Each Party's Input and Correctness of the Result," CRYPTO-87 Proceedings, 87–119.

[CGMA85] B. Chor, O. Goldreich, S. Micali and B. Awerbuch, "Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults," *Proc. of the 26th FOCS* (1985), 383-95.

[CR87] B. Chor and M. Rabin, "Achieving Independence in Logarithmic Number of Rounds," *Proc. of the 6th PODC* (1987).

[Fe88] P. Feldman, "One Can Always Assume Private Channels," unpublished manuscript (1988).

[FM88] P. Feldman and S. Micali, "Optimal Algorithms for Byzantine Agreement," *Proc. of the 20th STOC* (1988), 148–161.

[GHY87] Z. Galil, S. Haber and M. Yung, "Cryptographic Computation: Secure Fault-Tolerant Protocols and the Public-Key Model," CRYPTO-87 Proceedings, 135-155.

[GMW87] O. Goldreich, S. Micali and A. Wigderson, "How to Play Any Mental Game," *Proc. of the 19th STOC* (1987), 218–229.

[GV87] O. Goldreich and R. Vainish, "How to Solve any Protocol Problem—An Efficiency Improvement," CRYPTO-87 Proceedings, 76–86.

[Ha88] S. Haber, "Multi-Party Cryptographic Computation: Techniques and Applications," Columbia University Ph.D. Thesis (1988).

[Ha90] J. Håstad, "Pseudo-Random Generators with Uniform Assumptions," these proceedings.

[KMR90] J. Kilian, S. Micali and P. Rogaway, "The Notion of Secure Computation," manuscript, March 1990.

[ILL89] R. Impagliazzo, L. Levin and M. Luby, "Pseudo-Random Generation from One-Way Functions," *Proc. of the 21st STOC* (1989), 12–23.

[RB89] T. Rabin and M. Ben-Or, "Verifiable Secret Sharing and Multiparty Protocols with Honest Majority," *Proc. of the 21st STOC* (1989), 73–85.

[Ro90] P. Rogaway, "The Round Complexity of Secure Protocols," MIT Ph.D. Thesis (1990).

[Ya82a] A. Yao, Protocols for Secure Computation, *Proc. of the 23rd FOCS* (1982) 160–164.

[Ya82b] A. Yao, "Theory and Applications of Trapdoor Functions," *Proc. of the 23rd FOCS* (1982) 80–91.

[Ya86] A. Yao, "How to Generate and Exchange Secrets," *Proc. of the 18th STOC* (1986) 162–167.