Efficient Multiparty Protocols Using Circuit Randomization

Donald Beaver *
AT&T Bell Laboratories

Abstract.

The difference between theory and practice often rests on one major factor: efficiency. In distributed systems, communication is usually expensive, and protocols designed for practical use must require as few rounds of communication and as small messages as possible.

A secure multiparty protocol to compute function F is a protocol that, when each player i of n players starts with private input x_i, provides each participant i with $F(x_1, \ldots, x_n)$ without revealing more information than what can be derived from learning the function value. Some number t of players may be corrupted by an adversary who may then change the messages they send. Recent solutions to this problem have suffered in practical terms: while theoretically using only polynomially-many rounds, in practice the constants and exponents of such polynomials are too great. Normally, such protocols express F as a circuit C_F, call on each player to secretly share x_i, and proceed to perform "secret addition and multiplication" on secretly shared values. The cost is proportional to the depth of C_F times the cost of secret multiplication; and multiplication requires several rounds of interaction.

We present a protocol that simplifies the body of such a protocol and significantly reduces the number of rounds of interaction. The steps of our protocol take advantage of a new and counterintuitive technique for evaluating a circuit: set every input to every gate in the circuit completely at random, and then make corrections. Our protocol replaces each secret multiplication — multiplication that requires further sharing, addition, zero-knowledge proofs, and secret reconstruction — that is used during the body of a standard protocol by a simple reconstruction of secretly shared values, thereby reducing rounds by an order of magnitude. Furthermore, these reconstructions require only broadcast messages (but do *not* require Byzantine Agreement). The simplicity of broadcast and reconstruction provides efficiency and ease of implementation. Our transformation is simple and compatible with other techniques for reducing rounds.

*Research supported by AT&T Bell Laboratories, Murray Hill, NJ. Contact: Donald Beaver 313 Whitmore, Penn State Univ., State College, PA 16802; (814) 863-0147; beaver@cs.psu.edu.

J. Feigenbaum (Ed.): Advances in Cryptology - CRYPTO '91, LNCS 576, pp. 420-432, 1992.

1 Introduction

À quoi bon l'enfant qui vient de naître?

Benjamin Franklin[1] (1783)

The biggest drawback to current theoretical research in cryptography is its general impracticality: while polynomially-bounded resources are mathematically satisfying, they are often effectively out of reach. In distributed computing, where communication speeds lag behind processor speeds, the number of rounds of communication and the message sizes are significant issues to consider. A factor of ten can mean the difference between utility and impracticality.

Many solutions to secure multiparty function evaluation have been proposed [10, 11, 9, 7, 8, 4, 14, 2, 12, 6, 5] but none seem easily implementable, despite a reasonable clarity in their description and a theoretically small requirement for resources. These methods normally rely on the share-compute-reveal paradigm, in which processors secretly share their inputs, run subprotocols to evaluate gates of a bounded fanin[2] arithmetic circuit C_F that expresses the function F to compute, and reveal the final secret representing the output. Each subprotocol is an addition or multiplication of secrets, and uses a constant number of rounds and a small polynomial number of messages. But an n^2 factor or even a constant factor of twenty in a network of a hundred processors is debilitating.

In Shamir's method for secret sharing, each processor i shares secret s by selecting a polynomial $f(u) = a_t u^t + \cdots + a_1 u + s$ with coefficients a_k chosen uniformly at random over a finite field E, setting $\text{PIECE}_j(s) = f(\alpha_j)$ (where $\alpha_j \neq 0$ is an identifier for player j), and sending $\text{PIECE}_j(s)$ to player j for each j. Ben-Or, Goldwasser, and Wigderson point out that when the α_j are selected carefully, the pieces of s turn out to be elements in a BCH code, and a unique secret is reconstructible despite up to t changes, when $t \leq \lfloor n/3 \rfloor$.

A natural advantage of using polynomials is that combining two secrets r and s to form a new secret q whose value is their sum is easy, and requires no interaction: each j sets $\text{PIECE}_i(q) \leftarrow \text{PIECE}_i(r) + \text{PIECE}_i(s)$. If $f(u)$ represents $f(0) = r$ and $g(u)$ represents $g(0) = s$, then $h(u) = f(u) + g(u)$ represents $h(0) = r + s$. Multiplication by a publicly known constant is also easy: $\text{PIECE}_i(cs) = c \cdot \text{PIECE}_i(s)$. Thus a new secret may be computed without having to reveal r and s.

When secrets are multiplied, however, problems arise: the degree of the polynomial $h(u) = f(u)g(u)$ determined by the products of the individual pieces is $2t$, which soon grows out of hand. To deal with this problem, an interactive subprotocol for "degree reduction" is employed [7, 8]. We shall not present the solution here, but we shall give a brief outline in order to consider its resource requirements. Essentially, each processor is required to reshare $\text{PIECE}_i(r)$ and $\text{PIECE}_i(s)$ along with a third secret c_i, and must then prove interactively that $c_i = \text{PIECE}_i(r)\text{PIECE}_i(s)$. The players also jointly create several uniformly random secrets. Each player's piece of the new secret $q = rs$ then becomes

[1] "What good is a newborn child?" —Upon being asked at the launching of one of the first hot-air balloons, what good it would serve.

[2] Multiplicative gates have fanin 2; additive gates have unbounded fanin.

a particular linear combination of the confirmed c_i values along with uniformly random secret values to preserve the uniformity of higher-order coefficients; there is a different linear combination to compute for each player. Thus, a multiplication involves sharing, creating random secrets, proving products are valid, and reconstruction.

A standard protocol to compute F evaluates C_F level by level, creating new secrets at each stage using linear combinations and multiplications. Let S be the number of rounds used to share a secret and let M be that to multiply. If the depth of C_F is D_F, then the standard solution requires some $S + D_F M + 1$ rounds. Some improvements can be made for functions $F \in$ NC; Bar-Ilan and Beaver [1] slice a $\log n$ factor from the depth of the circuit. But multiplications involving sharing and proofs and reconstructions are still involved at each step.

Our solution. We present a protocol that does evaluate the circuit C_F level by level, but *each level is simply a reconstruction of secrets*, rather than a full-blown multiplication. In other words, each level uses *only* one *round of interaction* requiring neither private channels nor broadcast (in the sense of Byzantine Agreement).

Whereas previous protocols employ a great deal of processing to protect against Byzantine (malicious, message-changing) adversaries throughout the execution of the protocols, our protocol requires no elaborate on-line protection during the body of the protocol. Malicious or random errors are easily detected by checking whether n points interpolate to a t^{th}-degree polynomial. Byzantine failures are corrected directly using BCH decoding as in normal secret-reconstruction — interaction, proofs, private communication, Byzantine Agreement, and other special procedures are not necessary. In fact, it suffices that all nonfaulty players broadcast a message (broadcasts from faulty players can be incomplete or inconsistent; the players need not agree on the set of messages apparently broadcast).

In addition to reducing network requirements — Byzantine Agreement and private channels can be costly to implement — the body of our protocol (send public messages, do BCH error correction) is easy to implement using established software. The reduction in programming complexity lends a greater degree of confidence in the security of an implementation.

The cost and the savings. We achieve these improvements by preprocessing: but the preprocessing stage costs only one phase of multiplication. The total number of rounds drops from $S + D_F M + 1$ rounds to $S + M + D_F$, a significant improvement of a factor of M over standard techniques. The total number of multiplications in our protocol remains *exactly* the same as in [7, 8], namely exactly the number of multiplicative gates in C_F, but our multiplications are performed all in parallel. The number of additions is increased slightly in order to generate (just) N uniformly random secrets, where N is the size of C_F. The increase in additions is a tiny fraction compared to the number of additions and multiplications required by a standard ([7, 8]) protocol.

The idea. Our solution arises from an idea that has no obvious value at first glance: *completely randomize every input and output to each gate in* C_F. That is, select random inputs to each gate and compute the results of each gate all at once. Surprisingly, such a mangled circuit turns out to be useful. Through a very simple error-correction proce-

dure, the accurate (and secret) values at each level can be computed without recourse to multiplication.

This error-correcting procedure is based on a technique introduced in [2] for proving that the product of two secrets is a third. Namely: if a dealer wishes to show that $c = ab$, where he has shared a, b, and c, then he chooses random numbers Δ_a and Δ_b and shares $d = (a + \Delta_a)(b + \Delta_b)$. A "verifier" (the system) checks the dealer randomly by flipping a coin: the dealer must either (1) reveal Δ_a and Δ_b, in which case the system checks that the linear combination $d - (a\Delta_b + b\Delta_a + \Delta_a\Delta_b) - c = 0$, or (2) reveal d and linear combinations $(a+\Delta_a)$ and $(b+\Delta_b)$, in which case the system checks $d = (a+\Delta_a)(b+\Delta_b)$. Appropriate repetitions prevent cheating; the dealer must cheat at many random points in order that his secrets be consistent. In the current work, Δ_a and Δ_b become corrections to a gate whose inputs $(a + \Delta_a)$ and $(b + \Delta_b)$ have been chosen uniformly at random, and whose output $d = (a + \Delta_a)(b + \Delta_b)$ has been precomputed.[3] Multiplication of two real inputs a and b reduces to computing a correction to d that, in turn, reduces to a linear combination of a, b, d, and the publicly revealed (but uniformly random) corrections Δ_a and Δ_b.

One-time tables. We coin the term "one-time table" to describe a set of precomputed values that support direct secure computation without broadcast or private channels. This is analogous to a *one-time pad*, which is a random sequence of bits permitting two parties to send arbitrary messages with perfect privacy over public channels. Our "circuit randomization" constructs a list of independent, secret products of independent, secret random numbers. This list depends on F only insofar as it must contain as many entries as there are multiplicative gates in a circuit for F — in the same way a one-time pad must contain enough bits to mask a given message. The cost of using an entry is one round of open communication and requires neither broadcast nor private channels. Thus, we propose the precomputation of a *one-time table*, using a few short and initial rounds of costly (broadcast or private) channels, as a general paradigm for secure protocol design.

Contents. We describe the protocol in §2, giving our main theorem in §2.2 and a proof sketch in §2.3. We discuss efficiency and practical issues in §3.

2 An Efficient Protocol for Circuit Evaluation

Fix n, the number of players; $t(n) < n/2$, a bound on the number of players who can be corrupted in a dynamic and malicious (Byzantine) way; E, a finite field such that $|E| > n + 1$; F, a function from E^n to E described by a polynomial-size circuit C_F (without loss of generality, this covers finite functions from polynomial bits to polynomial bits [with poly-size circuits] as well as the case where each player learns a private output); and $\alpha_1, \ldots, \alpha_n$, primitive n^{th} roots of unity over E. For the sake of exposition, we assume private and broadcast channels are available and that $t(n) < n/3$, but we note that our

[3] In [2], the *symbols* Δ_a and Δ_b appear as r and s, not to be confused with the use of r and s in this work.

techniques apply to any protocol based on the share-compute-reveal paradigm.

We employ the secret sharing (SHARE), reconstruction (REC), linear combination (LINEAR-COMBINE), and multiplication (MULTIPLY) protocols of [7] for $t < n/3$; for $n/3 \leq t < n/2$, methods of [14, 2] can be used. We refer the reader to [7, 8, 14, 2] for detailed descriptions. Before describing the protocol, we note that Figure 1 gives a simple and well-known subprotocol UNIFSECRET to generate uniformly random secrets by adding uniformly random secrets shared by each party.

UNIFSECRET (r)		
1	$(1 \leq i \leq n)$	Player i sets $r_i \leftarrow \mathit{uniform}(E)$
2	$(1 \leq i \leq n)$	Player i shares r_i.
3		Run LINEAR-COMBINE : $r \leftarrow \sum_{i=1}^{n} r_i$.

Figure 1: Protocol to produce a uniformly random secret field element r.

Now, assume that circuit C_F has N wires $x_1, \ldots, x_N$, where $x_1, \ldots, x_n$ carry the inputs. Assume also that there are gates $g_k \in \{+, \times\}$ of fan-in 2, and write $x_k = g_k(x_{i_k}, x_{j_k})$ to mean that gate g_k has inputs x_{i_k} and x_{j_k} and output x_{k_k} with $i_k, j_k < l$. With each gate a depth **level**(g_k) is associated in a standard way. To evaluate C_F, one assigns the input wires the values of $x_1, \ldots, x_n$, then for each level L from 1 to D_F, one evaluates all the gates g_k at depth L, finally evaluating g_N to produce the circuit output x_N.

The share-compute-reveal paradigm follows exactly this pattern, using subprotocols to evaluate gates at each level and thereby produce new secrets x_k, until the final secret x_N is calculated. The result x_N can then be reconstructed using REC . We shall take an analogous but somewhat different route.

Let us ignore secret sharing for the moment and focus on the values used in the calculation of $x_N = F(x_1, \ldots, x_n)$.

In particular, let us create N *uniformly random* values $r_1, \ldots, r_N$, one for every x_i. For every gate g_k, compute $s_k = g_k(r_{i_k}, r_{j_k})$. These values have no apparent connection to the correct x_k values, nor to the final output.

But consider now the *corrections* $\Delta_i = r_i - x_i$ to each wire. We begin the evaluation of C_F by computing the corrections to the inputs: $\Delta_1 = r_1 - x_1, \ldots, \Delta_n = r_n - x_n$. Given an additive gate $g_k = (+)$ with random inputs r_{i_k}, r_{j_k}, random output r_k, and input wire corrections $\Delta_{i_k}, \Delta_{j_k}$, we can compute the correction Δ_k for the output wire as $\Delta_k = r_k - s_k - \Delta_{i_k} - \Delta_{j_k}$, because:

$$\begin{aligned} \Delta_k &= r_k - (x_{i_k} + x_{j_k}) \\ &= r_k - (r_{i_k} + \Delta_{i_k}) - (r_{j_k} + \Delta_{j_k}) \\ &= r_k - (r_{i_k} + r_{j_k}) - \Delta_{i_k} - \Delta_{j_k} \\ &= r_k - s_k - \Delta_{i_k} - \Delta_{j_k}. \end{aligned}$$

We may thus regard Δ_k as a *linear combination* of previously calculated, "known" constants Δ_{i_k} and Δ_{j_k} with the values r_k and s_k.

But more interesting, and crucial to our solution, is the observation that for multiplicative gates $g_k = (\times)$, $\Delta_k = r_k - s_k - r_{i_k}\Delta_{j_k} - \Delta_{i_k}r_{j_k} - \Delta_{i_k}\Delta_{j_k}$, because:

$$\begin{aligned}
\Delta_k &= r_k - (x_{i_k}x_{j_k}) \\
&= r_k - (r_{i_k} + \Delta_{i_k})(r_{j_k} + \Delta_{j_k}) \\
&= r_k - r_{i_k}r_{j_k} - r_{i_k}\Delta_{j_k} - \Delta_{i_k}r_{j_k} - \Delta_{i_k}\Delta_{j_k} \\
&= r_k - s_k - r_{i_k}\Delta_{j_k} - \Delta_{i_k}r_{j_k} - \Delta_{i_k}\Delta_{j_k}.
\end{aligned}$$

In other words, if Δ_{i_k} and Δ_{j_k} are already calculated and "known," then Δ_k is a *linear combination* of these "known" constants and the values r_k, s_k, r_{i_k}, r_{j_k}.

Now, let's return to our worries about security. Say that all values r_k are uniformly random and secretly shared, that all values $q_k = g_k(r_{i_k}, r_{j_k})$ have been secretly computed and are secrets, and that the lowest level of corrections $\Delta_1 = r_1 - x_1, \ldots, \Delta_n = r_n - x_n$ have been computed *and reconstructed.* Then the computation of a new correction at each level is a *linear combination* of publicly known values $(\Delta_{i_k}, \Delta_{j_k})$ and secret values $(r_k, s_k, r_{i_k}, r_{j_k})$, requiring no interaction. In fact, each player computes his piece of new Δ_k as a linear combination of pieces; with $g_k = \times$, for example, player i computes:

$$\text{PIECE}_i(\Delta_k) \leftarrow \text{PIECE}_i(r_k) - \text{PIECE}_i(s_k) - \text{PIECE}_i(r_{i_k})\Delta_{j_k} - \Delta_{i_k}\text{PIECE}_i(r_{j_k}) - \Delta_{i_k}\Delta_{j_k}.$$

The interaction for each level arises from *reconstructing* Δ_k from these pieces of Δ_k, which involves a single broadcast of each piece by each player. (Byzantine faults — inconsistencies in pieces supplied by faulty players — are covered by BCH error-correction without the need for Byzantine Agreement or other interaction.)

Now, note that because every r_k is uniformly random, every Δ_k value is also uniformly random, regardless of the inputs. Thus, no information whatsoever is revealed by reconstructing the Δ_k values. The only reconstruction that contains nonrandom information is the reconstruction of the final output secret $x_N = F(x_1, \ldots, x_n)$.

With this explanation in hand, the reader should be prepared to read Figure 2, which describes the protocol. Note that the final level is computed differently because the secret x_N does not need to be fed into a new, randomized gate.

2.1 An Optimization

With a simple optimization, additive layers in the circuit can be ignored, in the sense that they need not lead to interaction. We "compress" additive gates by computing the corrections to their outputs at the same time as the corrections to their input wires. In fact, no "randomization" of additive gates (secret generation of R_1, R_2, and computation of $S = R_1 + R_2$ during preprocessing) is needed.

Let us illustrate the technique with an example. Let gate $g_5(x_1, x_2) = x_1x_2$, $g_6(x_3, x_4) = x_3x_4$, and $g_7(x_5, x_6) = x_5 + x_6$. That is, the output of g_7 is $(x_1x_2 + x_3x_4)$. Rather than go through two stages (multiply, add), we need only one (multiply). Define $\delta_k = s_k - x_k$ for $1 \le k \le N$. Although the δ_k values are not uniformly distributed (and

RANDCIRCUIT (C_F)

1.1	$(1 \le i \le n)$	Player i: run SHARE(x_i), dealing secret x_i
1.2	$(1 \le k \le N)$	Run UNIFSECRET(r_k)
2.1	$(1 \le k \le n)$	Run LINEAR-COMBINE: $\Delta_k \leftarrow r_k - x_k$
2.2	$(1 \le k \le n)$	Run REC(Δ_k) to reveal Δ_k
2.2	$(N+1 \le k \le N)$	if $g_k = (+)$ **then** run LINEAR-COMBINE: $s_k \leftarrow r_{i_k} + r_{j_k}$
		else run MULTIPLY: $s_k \leftarrow r_{i_k} r_{j_k}$
3		**For** $L = 1..(D_F - 1)$ **do**
		$(\forall k$ such that **level**$(g_k) = L)$:
3.1		If $g_k = (+)$ **then** run LINEAR-COMBINE:
		$\Delta_k \leftarrow r_k - s_k - \Delta_{i_k} - \Delta_{j_k}$
		else run MULTIPLY:
		$\Delta_k \leftarrow r_k - s_k - r_{i_k}\Delta_{j_k} - \Delta_{i_k} r_{j_k} - \Delta_{i_k}\Delta_{j_k}$
3.2		Run REC(Δ_k)
4.1		If $g_N = (+)$ **then** run LINEAR-COMBINE:
		$x_N \leftarrow s_N + \Delta_{i_N} + \Delta_{j_N}$
		else run MULTIPLY:
		$x_N \leftarrow s_N + r_{i_N}\Delta_{j_N} + \Delta_{i_N} r_{j_N} + \Delta_{i_N}\Delta_{j_N}$
4.1	Run REC(x_N) to reveal $F(x_1, \ldots, x_n) = x_N$.	

Figure 2: Protocol to evaluate function F represented by arithmetic circuit C_F of size N and depth D_F over field E. SHARE, REC, LINEAR-COMBINE, MULTIPLY, and UNIFSECRET are described in the text. "$(1 \le i \le n)$" means run in parallel; **for** loops are executed in sequence.

hence should not be revealed), they satisfy a set of equations similar to those for the Δ_k values; thus,

$$\begin{aligned} \delta_5 &= -r_1\Delta_2 - \Delta_1 r_2 - \Delta_1\Delta_2, \\ \delta_6 &= -r_3\Delta_4 - \Delta_3 r_4 - \Delta_3\Delta_4. \end{aligned}$$

This gives

$$\begin{aligned} \Delta_7 &= r_7 - (x_5 + x_6) \\ &= r_7 - (s_5 + \delta_5) - (s_6 + \delta_6) \\ &= r_7 - s_5 - s_6 + r_1\Delta_2 + \Delta_1 r_2 + \Delta_1\Delta_2 + r_3\Delta_4 + \Delta_3 r_4 + \Delta_3\Delta_4, \end{aligned}$$

which expresses Δ_7 as a linear combination of secret inputs to g_5 and g_6, not of inputs to g_7. Random secrets r_5 and r_6 are avoided altogether. Thus, not only need gate g_7 not be "randomized," the computation of the correction Δ_7 to its output need not wait for the computation and reconstruction of values Δ_5 and Δ_6.

A simple way to think of this is to set $r_{i_k} = s_{i_k}$ and $r_{j_k} = s_{j_k}$ for every additive gate $s_k = g_k(r_{i_k}, r_{j_k})$; then $s_k = r_{i_k} + r_{j_k} = s_{i_k} + s_{j_k}$, and the formula for Δ_k depends only on the Δ values for the inputs to g_{i_k} and g_{j_k}. The example given above generalizes in a natural way to give a general technique that "ignores" additive gates.

2.2 Main Results

The definitions of *resilience* — a combination of security and fault-tolerance — and of *perfect* and *exponential* are given in §2.3.

Theorem 1 *Let $t(n) < n/3$. Let S be the number of rounds to share a secret, and let M be the number of rounds to multiply. Let F be any function from E^n to E described by a poly-size circuit C_F. Then* RANDCIRCUIT *is a perfectly t-resilient protocol for F against Byzantine adversaries and requires only $S + M + D_F$ rounds of interaction, a savings of $\Omega(D_F)$ rounds over previous protocols.*

Proof. Referring to Figure 2, step 1 requires S rounds, step 2 requires M rounds, step 3 uses $(D_F - 1)$ secret reconstructions, and step 4 uses one secret reconstruction. Earlier protocols used $S + MD_F + 1$; the improvement is $(D_F - 1)(M - 1) = \Omega(D_F)$ rounds. See §2.3 for a proof of resilience (security). □

Let RANDCIRCUIT′ denote the protocol obtained from RANDCIRCUIT by using secret-sharing, addition, and multiplication protocols designed to withstand $t < n/2$ rather than $t < n/3$ faults, and let S', M' be the number of rounds required by the corresponding subprotocols.

Theorem 2 *For $t(n) < n/2$,* RANDCIRCUIT′ *is an exponentially t-resilient protocol for F against Byzantine adversaries and requires only $S' + M' + D_F$ rounds of interaction, a savings of $\Omega(nD_F)$ rounds over previous protocols.*

Proof. As in Theorem 1, the number of rounds used is $S' + M' + D_F$; existing methods for multiplication require $\Omega(t)$ rounds when t faults are possible, so the difference of $(D_F - 1)(M' - 1)$ gives the savings reported. Because of the exponentially small chance of error in the secret addition and multiplication protocols, the resilience of our protocol is only exponential; but better resilience is shown impossible in [7]. □

Using the optimization described in §2.1, we observe that the round complexity depends only on the multiplicative depth of a circuit C_F for F. Let RANDCIRCUIT* denote the protocol obtained from RANDCIRCUIT by computing corrections to additive gates at the same time as the corrections to the multiplicative gates that feed them, as described in §2.1; "randomization" is not performed for additive gates. Let $\mathbf{depth}_\times(C)$ give the multiplicative depth of circuit C, and let $\mathrm{depth}_\times(F)$ be the minimum over all polynomial-size bounded-fanin arithmetic circuits C_F computing function F.

Theorem 3 *For $t(n) < n/3$, protocol* RANDCIRCUIT* *is a perfectly t-resilient protocol for F against Byzantine adversaries and requires only $S + M + \mathbf{depth}_{\times}(F)$ rounds of interaction. An analogous result with exponential resilience holds for a protocol* RANDCIRCUIT*′ *designed for $t(n) < n/2$.*

We remark that avoiding the randomization for additive gates reduces the message complexity as well; the UNIFSECRET protocol is called only for multiplicative gates, not for all N gates in C_F.

Proof. The proof follows the lines of the proof of Theorem 1. The improvement in round complexity arises from "compressing" additive gates as described in §2.1. □

2.3 Proof Sketch

We adopt the security definitions of [2, 3]. A brief sketch is included here for completeness. An ensemble P is a collection of distributions $P(z,k)$ for $z \in \Sigma^*$ and $k \in \mathbf{N}$. The difference between distributions $\mathcal{P}$ and $\mathcal{Q}$ on finite domain X is $\sum_X |\Pr_{\mathcal{P}}[x] - \Pr_{\mathcal{Q}}[x]|$. Ensembles P and Q are $O(\delta(k))$-**indistinguishable** ($P \approx^{\delta(k)} Q$) if $(\exists k_0)(\forall k \geq k_0)(\forall z)|P(z,k) - Q(z,k)| < \delta(k)$. **Perfect** ($O(0)$), **exponential** ($O(c^{-k})$), **statistical** ($(\forall c)O(k^{-c})$), and **computational** indistinguishability are straightforwardly defined.

A protocol α with n players having m-bit inputs $\vec{x} = (x_1, \ldots, x_n)$ and auxiliary inputs $\vec{a} = (a_1, \ldots, a_n)$ induces a distribution $[\alpha](\vec{x}\cdot\vec{a}, k)$ on outputs $\vec{y} = (y_1, \ldots, y_n)$; k is a security parameter given to each player. When an adversary $\mathcal{A}$ with auxiliary input a_A is allowed to attack α, the resulting distribution on *all* outputs $\vec{y}\cdot y_A$ is denoted $[\alpha, \mathcal{A}](\vec{x}\cdot\vec{a}\cdot a_A)$. An **interface** is an interactive machine with two tapes; it communicates with an adversary $\mathcal{A}$ on one, and acts as an adversary on the other. The distribution induced by running protocol β with adversary $\mathcal{I}(\mathcal{A})$ (*i.e.* $\mathcal{A}$ with interface $\mathcal{I}$ translating its corruptions) is denoted $[\beta, \mathcal{I}(\mathcal{A})](\vec{x}\cdot\vec{a}\cdot a_A)$. Ranging over all $z = \vec{x}\cdot\vec{a}\cdot a_A$ and k, we consider ensembles $[\alpha, \mathcal{A}]$ and $[\beta, \mathcal{I}(\mathcal{A})]$.

Protocol α is as resilient (*i.e.* secure and reliable) as β if an adversary $\mathcal{A}$, when it attacks α, gains the same information and has the same effect on nonfaulty outputs as when $\mathcal{A}$ attacks β through $\mathcal{I}$:

Definition 1 (Relative Resilience) *Let* $\mathtt{ADV}_\alpha$ *and* $\mathtt{ADV}_\beta$ *denote the set of allowed adversaries for α and β. Protocol α is* **as resilient as** *β, written $\alpha \succeq \beta$, if there exists an interface $\mathcal{I}$ from α to β such that for all $\mathcal{A} \in \mathtt{ADV}_\alpha$, we have $\mathcal{I}(\mathcal{A}) \in \mathtt{ADV}_\beta$ and*

$$[\alpha, \mathcal{A}] \approx [\beta, \mathcal{I}(\mathcal{A})].$$

The ideal protocol for F, $\mathtt{ID}(F)$, contains an incorruptible host that collects inputs x_i in the first round and returns $F(x_1, \ldots, x_n)$ in the second.

Definition 2 (Resilience) *α is a t-resilient protocol for F if $\alpha \succeq \mathtt{ID}(F)$, against t-adversaries.*

Proof of Theorem 1. We must find an interface $\mathcal{I}$ that translates attacks on RANDCIRCUIT to attacks on the ideal, trusted-host protocol $\mathtt{ID}(F)$ for F. Because initial messages (preprocessing steps 1 and 2) between nonfaulty players are not seen by an adversary, and because most messages created by nonfaulty players (whether sent to corrupted players or seen when later corrupted) are generated uniformly at random, subject to simple linear algebraic conditions and the input values and output value $F(x_1, \ldots, x_n)$, an interface is neither hard to design nor to prove correct. The following lemmas describe the responses that $\mathcal{I}$ must make in order to supply $\mathcal{A}$ with a proper set of responses to corruption requests while at the same time inducing the same outputs in $\mathtt{ID}(F)$ that $\mathcal{A}$ induces in RANDCIRCUIT.

Lemma 4 *(Messages from nonfaulty to faulty before final round.) Let $t(n) < n/3$. For any corrupted subset $T \subset \{1, .., n\}$ and any round r except the last, the set of messages from players not in T to players in T consists of uniformly random field elements subject to interpolating to a uniformly random polynomial of degree t.*

Proof. Simple linear algebra. In the case of messages regarding Δ reconstructions, the "uniformly random polynomials" have uniformly random Δ values as their free terms. In the case of messages regarding secretly-shared inputs or random secrets, a set of t messages to players in T is a uniformly random t-vector, whether it be generated from a uniformly random polynomial of degree t having free term x_i or from a uniformly random polynomial of degree t. □

Lemma 5 *(Newly corrupt player i before final round; how to generate past view.) Let $t(n) < n/3$. For any corrupted subset $T \subset \{1, .., n\}$, any round r except the last, and any $i \notin T$, the distribution on secret values generated by i through round r is uniform, the distribution on messages through round r from players not in T to player i is uniform subject to interpolating to already revealed Δ_k values, and the distribution on messages from player i to players not in T is uniform from among the set of solutions consistent with the secret values player i randomly generated, the Δ_k values revealed through round r, and its input x_i.*

Proof. Simple linear algebra. □

Lemma 6 *(Messages from nonfaulty at final round.) Let $t(n) < n/3$. For any corrupted subset $T \subset \{1, .., n\}$, at the final round the distribution on messages from players not in T to players in T is uniform subject to interpolating to $F(x_1, \ldots, x_n)$ [with x_i replaced by default value 0 for any player who failed to share an input].*

Proof. Simple linear algebra. □

Lemma 7 *(Newly corrupt player i at final round; how to generate past view.) Let $t(n) < n/3$. For any corrupted subset $T \subset \{1, .., n\}$, any round r except the last, and any $i \notin T$,*

the distribution on secret values generated by i through round r is uniform, the distribution on messages through round r from players not in T to player i is uniform subject to interpolating to already revealed Δ_k values, and the distribution on messages from player i to players not in T is uniform from among the set of solutions consistent with the secret values player i randomly generated, the Δ_k values revealed through round r, its input x_i, and $F(x'_1, \ldots, x'_n)$ [where x'_j is x_j for nonfaulty j but is default value 0 for any player j who failed to share an input].

Proof. Simple linear algebra. □

The interface collects pieces of x_i distributed by players i corrupted at the start, and sets x'_i accordingly. For every player i corrupted by $\mathcal{A}$, $\mathcal{I}$ requests the corruption of i in $\mathrm{ID}(F)$, obtaining x_i, which it then returns to $\mathcal{A}$ along with a view of RANDCIRCUIT constructed according to the lemmas (this involves solutions to simple linear algebraic equations). For messages sent from nonfaulty players to faulty players, $\mathcal{I}$ chooses random elements according to the lemmas and sends them to $\mathcal{A}$. Interface $\mathcal{I}$ records all outgoing messages from $\mathcal{A}$ intended to replace messages from corrupted players, and uses them to construct deterministic responses from nonfaulty, virtual players (in preprocessing steps **1** and **2**, nonfaulty players may detect cheating and use default value 0 for cheating players; in later steps, fewer than t modifications to the n-vector used in reconstruction does not affect the Δ value computed by nonfaulty players). For space reasons, we shall not describe the interface in greater detail.

The resilience of RANDCIRCUIT follows directly from the fact that the messages sent to $\mathcal{A}$ from $\mathcal{I}$ are *identical* to responses to corruption requests when $\mathcal{A}$ attacks RANDCIRCUIT, and the inputs x'_i that $\mathcal{I}$ passes on to $\mathrm{ID}(F)$ are exactly what $\mathcal{A}$ chooses and shares (whether or not by default). □

3 Theory and Practice

We have reduced the number of rounds of interaction by an order of magnitude, removing yet another obstacle to the use of theoretical results in practice. Our methods apply to any protocol based on circuit evaluation, including cryptographic multiparty protocols, two-party oblivious circuit evaluation, *etc.*. The advantage of circuit randomization over direct evaluation is proportional to the cost of multiplication *vs.* the cost of addition.

Theoretically, the number of rounds can be reduced further by an $O(\log n)$ factor using the results of [1]. In practice, the choice whether to follow this route as well depends on just how large the $O(\log n)$ reduction is. The methods of [1] require multiplying three 3×3 matrices, which normally costs two multiplications, but which now costs 2 rounds (each matrix multiplication costs a round to correct the $\times$-gates). Since the submission of this abstract, we have discovered an alternative to [1] that permits unbounded fan-in multiplication but suffers neither from *expected* round complexity (a complication that makes implementation difficult) nor from having to convert circuit layers to matrix products.

The RANDCIRCUIT protocol is, in many senses, simpler than direct circuit evaluation.

The initial processing is perhaps less than immediately intuitive, but it contains the same set of computations that must be done during direct circuit evaluation, parallelized and applied to random secrets. The only added complexity arises from creating random secrets at the start, but this subprotocol is easy relative to the one for multiplication.

In fact, implementation is far easier than for direct circuit evaluation, since the body of the protocol consists of reconstructions, which require simply a weak broadcast (all nonfaulty players broadcast their value, without Byzantine Agreement) from each player. A weak broadcast is far simpler to implement than a complicated multiplication subprotocol that itself contains private communications and Byzantine Agreement protocols. In asynchronous settings, using weak broadcast rather than interaction becomes a tremendous advantage, not just for efficiency but for correctness in implementation.

Interestingly, because the steps of the protocol are reconstructions, error correction in the sense of zero-knowledge proofs of behavior is not needed. Error detection and correction is performed locally without interaction. Detection is simple: check whether received pieces interpolate to a degree-t polynomial. Correction uses a local BCH-error-code correction, without interaction. Thus, the overhead of repeated proofs, cut-and-choose methods, or other interactive error detection techniques is not necessary after the first multiplication.

It is interesting to note that the randomization techniques used here apply to circuit evaluation in both the cryptographic and the noncryptographic settings. The techniques we present were inspired by a proof system for multilinear polynomials [2] that has a similarity to recent interactive proof systems for functions expressible as low-degree polynomials, perhaps suggesting deeper connections and a wider applicability than simply to cryptographic protocol optimization. In any case, the practical advantages are evident.

Acknowledgements

The author would like to thank Donald W. Beaver for his perserverance in keeping the author awake and contemplating new ideas during late-night-early-morning discussions over warm milk.

References

[1] J. Bar-Ilan, D. Beaver. "Non-Cryptographic Fault-Tolerant Computing in a Constant Expected Number of Rounds of Interaction." *Proceedings of PODC,* ACM, 1989, 201–209.

[2] D. Beaver. "Secure Multiparty Protocols and Zero Knowledge Proof Systems Tolerating a Faulty Minority." To appear, *J. Cryptology.* An earlier version appeared as "Secure Multiparty Protocols Tolerating Half Faulty Processors." *Proceedings of Crypto 1989,* ACM, 1989.

[3] D. Beaver. *Security, Fault Tolerance, and Communication Complexity in Distributed Systems.* PhD Thesis, Harvard University, Cambridge, 1990.

[4] D. Beaver, S. Goldwasser. "Multiparty Computation with Faulty Majority." *Proceedings of the* 30^{th} *FOCS,* IEEE, 1989, 468–473.

[5] D. Beaver, S. Haber. "Cryptographic Protocols Provably Secure Against Dynamic Adversaries." Submitted to FOCS 91.

[6] D. Beaver, S. Micali, P. Rogaway. "The Round Complexity of Secure Protocols." *Proceedings of the* 22^{st} *STOC,* ACM, 1990, 503–513.

[7] M. Ben-Or, S. Goldwasser, A. Wigderson. "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation." *Proceedings of the* 20^{th} *STOC,* ACM, 1988, 1–10.

[8] D. Chaum, C. Crépeau, I. Damgård. "Multiparty Unconditionally Secure Protocols." *Proceedings of the* 20^{th} *STOC,* ACM, 1988, 11–19.

[9] Z. Galil, S. Haber, M. Yung. "Cryptographic Computation: Secure Fault-Tolerant Protocols and the Public-Key Model." *Proceedings of Crypto 1987,* Springer–Verlag, 1988, 135–155.

[10] O. Goldreich, S. Micali, A. Wigderson. "Proofs that Yield Nothing but Their Validity and a Methodology of Cryptographic Protocol Design." *Proceedings of the* 27^{th} *FOCS,* IEEE, 1986, 174–187.

[11] O. Goldreich, S. Micali, A. Wigderson. "How to Play Any Mental Game, or A Completeness Theorem for Protocols with Honest Majority." *Proceedings of the* 19^{th} *STOC,* ACM, 1987, 218–229.

[12] S. Goldwasser, L. Levin. "Fair Computation of General Functions in Presence of Immoral Majority." *Proceedings of Crypto 1990.*

[13] S. Haber. *Multi-Party Cryptographic Computation: Techniques and Applications,* PhD Thesis, Columbia University, 1988.

[14] T. Rabin, M. Ben-Or. "Verifiable Secret Sharing and Multiparty Protocols with Honest Majority." *Proceedings of the* 21^{st} *STOC,* ACM, 1989, 73–85.

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