Perfect Constant-Round Secure Computation via Perfect Randomizing Polynomials

Yuval Ishai[1]* and Eyal Kushilevitz[2]**

[1] Princeton University, USA. yishai@cs.princeton.edu.
[2] Technion, Israel. eyalk@cs.technion.ac.il.

Abstract. Various information-theoretic constant-round secure multiparty protocols are known for classes such as NC^1 and polynomial-size branching programs [1,13,18,3,19,10]. All these protocols have a small probability of failure, or alternatively use an *expected* constant number of rounds, suggesting that this might be an inherent phenomenon. In this paper we prove that this is not the case by presenting several constructions of *perfect* constant-round protocols.
Our protocols are obtained using *randomizing polynomials* – a recently introduced representation [19], which naturally relaxes the standard polynomial representation of boolean functions. Randomizing polynomials represent a function f by a low-degree mapping from its inputs and independent random inputs to a vector of outputs, whose distribution depends only on the value of f. We obtain several constructions of degree-optimal *perfect* randomizing polynomials, whose distinct output distributions are perfectly separated. These results on randomizing polynomials are of independent complexity-theoretic interest.

1 Introduction

Representation of functions by low-degree multivariate polynomials has proved to be a surprisingly powerful tool in complexity theory. Such a representation is also useful in the context of *secure multiparty computation*; in particular, most general-purpose protocols for secure multiparty computation can be used to evaluate constant-degree polynomials in a constant number of rounds.[1] A major difficulty, however, is that not many functions can be evaluated using low-degree polynomials, and even some very simple functions, like the logical OR of n bits, require polynomials of degree n.

A natural relaxation of the standard representation notion which gets around this obstacle was recently suggested in [19]. *Randomizing polynomials* extend the standard representation by incorporating randomness and by allowing *several* polynomials to simultaneously act on the same inputs and random inputs. Instead of directly outputting the value of the represented function, a randomizing polynomials vector is required to produce an *output distribution* which directly corresponds to this value. This is best illustrated by the next example, which shows a degree-2 representation of the OR function by randomizing polynomials.

---

* Work done while at AT&T Labs – Research and DIMACS.
** Work done in part while at IBM T.J. Watson Research Center. Supported in part by the Mitchell-Schoref program at the Technion and MANLAM Fund 120-044.
[1] More generally, the round complexity of these protocols is proportional to the *multiplicative depth* of an arithmetic circuit computing the function f of interest, where multiplicative depth is defined similarly to ordinary circuit depth except that addition gates are ignored.

Let $\mathbb{F}$ be some finite field, and $p = (p_1, \ldots, p_s)$ a vector of degree-2 polynomials over $\mathbb{F}$ in the n inputs $x = (x_1, \ldots, x_n)$ and the sn random inputs $r = (r_{ij})$, $1 \leq i \leq s, 1 \leq j \leq n$, defined by $p(x, r) = (\sum_{j=1}^{n} x_j r_{1j}\ , \ \ldots \ , \sum_{j=1}^{n} x_j r_{sj})$. For any input $x \in \{0,1\}^n$, let $P(x)$ denote the output distribution of p on x, i.e., the distribution over $\mathbb{F}^s$ induced by a uniform choice of r from $\mathbb{F}^{sn}$. It is not hard to verify that the above polynomial vector p satisfies the following two properties: (1) If $\mathrm{OR}(x) = \mathrm{OR}(y)$ then the output distributions $P(x)$ and $P(y)$ are *identical*; in other words, there exist probability distributions D_0, D_1 such that for any $x \in \{0,1\}^n$, $P(x)$ is equal to $D_{\mathrm{OR}(x)}$ (specifically, D_0 is concentrated on the zero vector, and D_1 is uniform over $\mathbb{F}^s$). (2) The statistical distance between D_0, D_1 is close to 1 (more precisely, it is $1 - |\mathbb{F}|^{-s}$).

Property (1) guarantees that from a sample of $P(x)$ it is impossible to learn *anything* about x except, perhaps, $\mathrm{OR}(x)$; Property (2) guarantees that from such a sample it is indeed possible to correctly compute $\mathrm{OR}(x)$ with high probability. Thus, learning a sample from $P(x)$ is, in a sense, information-theoretically equivalent to learning $\mathrm{OR}(x)$. Consequently, the task of securely computing the OR function may be reduced to the task of securely sampling from $P(x)$, which in turn can be reduced to that of securely evaluating a related vector of *deterministic* degree-2 polynomials over $\mathbb{F}$.

The application to secure computation will be more thoroughly discussed in Section 1.1. For the time being, however, we point out the fact that the above two output distributions D_0 and D_1 are not perfectly separated, and note that this does come at a cost: even if a perfect protocol is used for the evaluation of degree-2 polynomials, the resultant secure protocol for OR will have a nonzero error probability. This issue, which is further motivated below, stands in the center of the current work.

A general definition of randomizing polynomials $p(x, r)$ representing a boolean function $f(x)$ can be easily derived from the above example.[2] In [19] it was shown that *any* boolean function f can be represented by degree-3 randomizing polynomials, where the complexity of this representation (defined as the total number of inputs and outputs) is at most quadratic in the branching program size of f. It was also shown that almost all functions, with the exception of functions which are "similar" to the OR function, do not admit a degree-2 representation. However, the general degree-3 construction of [19] suffers from the same deficiency as the above example: the two output distributions D_0, D_1 are not completely disjoint. Thus, the general reduction it provides in the context of secure computation introduces a small probability of error. This raises the question whether *perfect* low-degree randomizing polynomials, for which the output distributions are perfectly separated, can be constructed, and if so at what cost.

Before describing our results we give some background on secure multiparty computation and motivate the above question in this context.

1.1 Secure Multiparty Computation and Its Round Complexity

A secure multiparty computation protocol allows k parties to evaluate a function of their inputs in a distributed way, so that both the *privacy* of their inputs and the *correctness* of the outputs are maintained. These properties should hold in the presence of an *adversary* which may corrupt at most t parties. The main focus of this work is on the *information-theoretic* setting for secure computation, in which security should hold

[2] For concreteness, we set a constant threshold of $1/2$ on the statistical distance between D_0, D_1; this distance can be amplified, without increasing the degree, by concatenating several copies of p having disjoint sets of random inputs.

against a computationally unbounded adversary. Nonetheless, our results are also useful in the alternative *computational* setting, as discussed in Section 1.2.

The *round complexity* of interactive protocols is one of their most important complexity measures. Indeed, substantial research efforts have been invested into characterizing the round complexity of various distributed tasks, such as zero-knowledge proofs and Byzantine Agreement. This is the case also for the general task of secure computation. Following the initial plausibility results [26,17,6,9], much of the research in this area has shifted to various *complexity* aspects of secure computation. In particular, the problem of obtaining constant-round secure protocols has attracted a considerable amount of attention [1,5,4,13,18,24,3,7,19,10,21]. Our work continues this line of research, and focuses on the following question: can *perfectly* secure computation be realized with a constant number of rounds in the *worst case*?[3]

In the computational setting for secure computation, any function that can be (efficiently) computed can also be securely computed in a constant number of rounds [26, 5,21]. The situation is not as well understood in the information-theoretic setting. Several (efficient) constant-round protocols are known in this setting for function classes such as NC^1, polynomial-size branching programs, and related linear algebra classes [1, 13,18,3,19,10]. All these protocols have a small probability of failure, or alternatively use an *expected* constant number of rounds, suggesting that this might be an inherent phenomenon. In the current work we show that this is not the case: we obtain *perfect* constant-round protocols which typically match or beat their previous non-perfect (information-theoretic) counterparts in every efficiency aspect.

1.2 Our Results

We present two main constructions of perfect randomizing polynomials, which in turn can be transformed via general-purpose protocols for perfectly secure computation (e.g., [6,11,14]) to perfect constant-round protocols. (This transformation is outlined in Section 2.3.) The communication complexity of the resultant protocols is proportional to the complexity of the underlying randomizing polynomials. Their exact number of rounds depends on the specific notion of security. For instance, t-security against a passive adversary can be achieved in 2 rounds if $t < k/3$ or in 3 rounds if $t < k/2$ (see [19]), and t-security against an active adversary can be achieved in 3 rounds with $t = \Omega(k)$ using a 2-round VSS protocol from [14] (assuming a broadcast channel is available). From now on, we describe the results in terms of randomizing polynomials and do not spell out the specific consequences for constant-round secure computation.

A combinatorial construction. Our first construction of perfect randomizing polynomials is combinatorial in nature, and is based on a boolean formula representation. We first derive a natural information-theoretic analogue of Yao's *garbled circuit* construction [26], which is originally cast in the computational setting, and observe that it gives rise to a representation by perfect degree-3 randomizing polynomials over $\mathrm{GF}(2)$. The complexity of this construction is at least quadratic (and always polynomial) in the formula size. We then present an optimization which allows a significant complexity improvement. We demonstrate this improvement for the function OR, for which the complexity of the optimized representation is $2^{O(\sqrt{\log n})} \cdot n$.

An algebraic construction. Our second construction is linear-algebraic in nature, and is based on a branching program representation. (The relevant branching program models

[3] The term "perfect security" binds together perfect privacy and correctness requirements; see, e.g., [8] for its formal definition.

are defined in Section 2.2.) We obtain, for any *counting* branching program of size ℓ, a perfect degree-3 representation of complexity ℓ^2. This construction applies to deterministic branching programs and logspace computation as special cases, but does not apply to the nondeterministic model.[4] In the full version of this paper we show that by settling for *statistical privacy* (a relaxation of property (1) of randomizing polynomials) it is possible to get an efficient representation also in the nondeterministic case. This yields perfectly-correct (yet statistically-private) constant-round protocols for NL.

We note that since branching programs can simulate formulas, the latter constructions can be efficiently applied to a presumably larger class of functions than the former. However, as the OR example demonstrates, the complexity of the (optimized) formula-based construction can be significantly better.

Efficiency advantages. In addition to providing perfect security in a strictly-constant number of rounds, our results also offer some independent efficiency advantages. In the information-theoretic setting, all previous constant-round protocols required either a significant parallel repetition of certain subprotocols or computation over large fields to make their failure probability small. (In the context of randomizing polynomials, this is needed to amplify the separation between the two output distributions.) Our constructions avoid this overhead. For instance, our perfect solution for branching programs over $\mathbb{F} = \mathrm{GF}(2)$ is slightly more efficient than a similar solution from [19] that can only achieve a small constant separation (which then needs to be amplified).

In the computational setting for secure computation, Yao's garbled circuit technique gives rise to constant-round protocols whose efficiency is *linear* in the *circuit* size of the function being computed and a security parameter [26,5,21]. Since none of the alternative information-theoretic techniques (and ours in particular) efficiently applies to circuits, they are generally considered less appealing. However, they do offer some efficiency advantages in the computational setting as well. Most notably, an advantage of most of these techniques over the garbled circuit approach is that they efficiently scale to *arithmetic* computation over large moduli. Our solution for *counting* branching programs, which applies to arithmetic formulas as a special case, is especially appealing in this context. This application and its generalization to arbitrary *rings* are further addressed in [12]. Finally, our constructions can be beneficial also in the boolean case; since their complexity does not involve a cryptographic security parameter (nor does its analysis hide large constants), they may be preferable for small branching programs or formulas which arise in specific applications (such as the "millionaire's problem" [25]).

2 Preliminaries

Notation. We let $\mathbb{F}$ denote a finite field; the underlying field $\mathbb{F}$ is often omitted when it can be understood from the context. The *statistical distance* between two probability distributions Y_0 and Y_1 is defined as $\mathrm{SD}(Y_0, Y_1) = \max_E |\Pr[Y_0 \in E] - \Pr[Y_1 \in E]|$.

2.1 Randomizing Polynomials

Syntax. A *randomizing polynomials vector* $p = (p_1, \ldots, p_s)$ is a vector of polynomials in the $n + m$ variables $x = (x_1, \ldots, x_n)$ and $r = (r_1, \ldots, r_m)$ over some finite field $\mathbb{F}$. The variables $x_1, \ldots, x_n$ will be referred to as the *inputs* of p and $r_1, \ldots, r_m$ as its *random inputs*. The *output complexity* of p is the number of polynomials s, its *randomness*

[4] Counting is usually more powerful than nondeterminism. However, in the context of perfect randomizing polynomials it is not clear how to eliminate the extra information provided by the exact count.

complexity is the number of random inputs m, and its *complexity* is the total number of all inputs and outputs $s+n+m$. Finally, the *degree* of p is defined as the (total) degree of its maximum-degree entry, where both ordinary inputs and random inputs count towards the degree.[5] For instance, if $p=(p_1,p_2)$ where $p_1(x,r)=x_1r_1^2$ and $p_2(x,r)=x_1x_2+r_1+r_2+r_3$, then the degree of p is 3.

Semantics. The following semantics of randomizing polynomials generalize the original ones from [19], and incorporate computational efficiency requirements which were not considered in the original definitions. However, the default notion of randomizing polynomials remains the same.

For any $x\in\mathbb{F}^n$ and $r\in\mathbb{F}^m$, the *output* $p(x,r)=(p_1(x,r),\ldots,p_s(x,r))$ is an s-tuple over $\mathbb{F}$. For any $x\in\mathbb{F}^n$, let $P(x)$ denote the *output distribution* of p on input x, induced by a uniform choice of $r\in\mathbb{F}^m$. Thus, randomizing polynomials may be thought of as computing a function from inputs to output distributions. We say that p represents a function f if the output distribution $P(x)$ "corresponds" to the function value $f(x)$. Motivated by the application to secure computation, we break this condition into two requirements, termed *privacy* and *correctness*.

Definition 1. *A polynomial vector $p(x,r)$ over $\mathbb{F}$ is an* ϵ-private, δ-correct randomizing polynomials representation *for a function $f:A^n\to B$, where $A\subseteq\mathbb{F}$ and B is an arbitrary set, if it has the following two properties:*

- ϵ-CORRECTNESS. *There exists a randomized* simulator *algorithm S such that for any input $x\in A^n$, $\mathrm{SD}(S(f(x)),P(x))\le\epsilon$.*
- δ-PRIVACY. *There exists a* reconstruction *algorithm C such that for any $x\in A^n$, $\Pr[C(P(x))\ne f(x)]\le\delta$.*

When referring to a uniform family *of randomizing polynomials, parameterized by n, we require that the simulator and the reconstruction algorithms be efficient in n.*

By default, we define randomizing polynomials to be 0-private, $1/4$-correct (similarly to [19]). The main new variant considered in this paper is that of *perfect* randomizing polynomials, defined as 0-private, 0-correct randomizing polynomials. Finally, we will also consider a third variant of *ϵ-private, perfectly-correct* randomizing polynomials.

2.2 Formulas and Branching Programs

Formulas. A formula is a single-output boolean circuit in n input variables in which each gate has a fan-out of 1. Specifically, a formula F is a directed binary tree. Each of its leaves is labeled by a *literal* which is either a variable from $x_1,\ldots,x_n$ or a negated variable from $\bar{x}_1,\ldots,\bar{x}_n$. Each internal node of the tree, referred to as a *gate*, has two incoming edges, referred to as *wires*, and is labeled by either AND or OR. This includes the node which is the root of the tree; we think of this node as also having an outgoing wire which is called the *output* wire of F. Any input $x\in\{0,1\}^n$ naturally assigns a unique *value* to each wire. The value of the formula F, denoted $F(x)$, is the value of its output wire.

Branching programs. Syntactically, a branching program (BP for short) is defined by a directed acyclic graph $G(V,E)$, two special vertices $s,t\in V$, and a labeling function ϕ assigning to each edge in E a literal (i.e., x_i or $\bar{x}_i$) or the constant 1. Its *size* is defined as $|V|-1$. Each input assignment $x=(x_1,\ldots,x_n)$ naturally induces an unlabeled subgraph G_x, whose edges include every $e\in E$ such that $\phi(e)$ is satisfied by x. An

[5] This convention is crucial for the application to secure multiparty computation.

accepting path on input x is a directed $s-t$ path in the graph G_x. We attach two main semantics to branching programs. A mod-q *counting* BP (CBP for short), where $q \geq 2$ is prime, computes the function $f : \{0,1\}^n \to \mathrm{GF}(q)$ such that $f(x)$ is the *number* of accepting paths on x modulo q. A mod-q *nondeterministic* BP (NBP for short) computes the boolean function $f : \{0,1\}^n \to \{0,1\}$, such that $f(x) = 1$ iff the number of accepting paths on x is *nonzero* modulo q. By setting q to be larger than the possible number of paths, we get the usual notion of nondeterminism over the integers. Finally, perhaps the most useful notion of BP is the special case of *deterministic BP*, where each input induces at most *one* accepting paths. A deterministic BP may be viewed as a mod-q CBP or NBP with an arbitrary $q \geq 2$.

2.3 Secure Multiparty Computation

The reader is referred to [8,16,22,15,2] for formal definitions of secure computation. We emphasize though that the issues addressed in this paper are quite insensitive to the exact notion of security. Our results provide *information-theoretic reductions* from the task of securely computing a general function f, represented by a formula or a branching program, to that of securely computing a vector of degree-3 polynomials. Such a reduction, originally described in [19] for the non-perfect case, proceeds as follows. Given a representation of $f(x)$ by $p(x,r)$, the secure computation of f (whose n inputs are arbitrarily partitioned among the k players) can be reduced to the secure computation of the randomized function $P(x)$. The latter, in turn, reduces to the secure computation of the *deterministic* function $p'(x, r^1, \ldots, r^{t+1}) \stackrel{\text{def}}{=} p(x, r^1 + \ldots + r^{t+1})$, where t is the security threshold, by assigning each input vector r^j to a distinct party and instructing it to pick it at random. Note that the degree of p' is the same as that of p. The above reduction preserves perfectness: if $p(x,r)$ is a perfect representation for f and if a perfectly secure protocol is used for evaluating p', then the resultant protocol for f is also perfectly secure.[6] In the remainder of this paper we will phrase our results in terms of randomizing polynomials and will not state the corollaries to secure computation.

3 Perfect Randomizing Polynomials from Formulas

In this section we construct perfect degree-3 randomizing polynomials from a boolean formula representation. The construction works over $\mathbb{F} = \mathrm{GF}(2)$, which we take to be the underlying field throughout this section. We start with a basic construction, which may be viewed as the natural information-theoretic analogue of Yao's garbled circuit construction. (Our notation for this section closely follows the presentation of Yao's construction from [23].) We later present an optimization of this basic construction.

Let F be a boolean formula of size s computing the function $f : \{0,1\}^n \to \{0,1\}$. It may be assumed, without loss of generality, that F has depth $d = O(\log s)$. We will efficiently transform F to a perfect degree-3 randomizing polynomials representation for f, whose complexity is polynomial in s. As usual, denote by x the input for F and by $x_1, \ldots, x_n$ its individual n variables. Let m be the number of wires in F, where the m-th wire is the output wire. For $i \in [m]$, denote by $b_i(x)$ (or simply b_i) the value of the i-th wire induced by the input x. In constructing the randomizing polynomials vector $p_F(x,r)$ we use random inputs of two types: m bits denoted $r_1, \ldots, r_m$ corresponding

[6] In the case of security against an *active* adversary, it is important that the input domain of p' be defined so that x is taken from A^n, the input domain of f, rather than from $\mathbb{F}^n$ (if they are different). In the default boolean case ($A = \{0,1\}$) standard protocols from the literature can be modified to handle such a restriction on the inputs of p' with little or no efficiency overhead.

to the m wires of F, and m pairs of strings W_i^0, W_i^1 again in correspondence with the m wires. The length of the strings W_i^b is defined inductively (from top to bottom) as follows: $|W_m^b| = 0$ (for $b \in \{0,1\}$) and if k is the output wire of some gate g and i, j are the input wires of this gate then $|W_j^b| = |W_i^b| = 2(|W_k^b|+1)$ (for $b \in \{0,1\}$); therefore the length of each of these strings is at most $O(2^d)$ =poly(s). We view each string W_i^b as if it is broken into two equal-size halves denoted $W_i^{b,0}, W_i^{b,1}$. We use c_i to denote the value of wire i masked by r_i; namely, $c_i = b_i \oplus r_i$.

To define the polynomial vector p_F, we specify several polynomials for each wire. In what follows $\oplus$ denotes bitwise-xor among strings; when we want to emphasize that the operation is applied to single bits we will usually denote it by either $+$ or $-$. We call *meta-polynomial* a polynomial that involves strings. Each meta-polynomial has a simple transformation into a vector of polynomials that operate bit-by-bit (e.g., for $a \in \mathrm{GF}(2)$ and strings A, B of length t the expression $a \cdot A \oplus B$ is a meta-polynomial that represents the length-t polynomial vector $(a \cdot A_1 + B_1, \ldots, a \cdot A_t + B_t)$. When we want to emphasize that a term T is actually a string we write $\langle T \rangle$. We use $\circ$ to denote concatenation. We now describe the polynomial vector associated with each wire.

Input wires: For an input wire i, labeled by a literal ℓ, we use the following meta-polynomial $\langle W_i^\ell \circ (\ell + r_i) \rangle$. Note that ℓ is either some variable x_u or its negation, which can be written as the degree-1 polynomial $1 - x_u$. Also, note that each term W_i^ℓ can be represented by $\ell \cdot W_i^1 \oplus (1-\ell) \cdot W_i^0$. All together, this is a degree-2 meta-polynomial which, as described above, is just a short writing for a vector of degree-2 polynomials over the boolean inputs x and boolean random inputs r, W.

Output wires of gates: Let g be a gate with input wires i, j and output wire k. We associate with this wire 4 meta-polynomials. Specifically, for each of the 4 choices of $c_i, c_j \in \{0,1\}$, we define a corresponding meta-polynomial on strings of length $|W_k^b|+1$. This degree-3 meta-polynomial can be thought of as the garbled table entry indexed by (c_i, c_j) and is defined as follows:

$$Q_k^{c_i,c_j}(x,r) \overset{\text{def}}{=} W_i^{c_i - r_i, c_j} \oplus W_j^{c_j - r_j, c_i} \oplus \langle W_k^{g(c_i - r_i, c_j - r_j)} \circ (g(c_i - r_i, c_j - r_j) + r_k) \rangle \quad (1)$$

Note that $Q_k^{c_i,c_j}$ actually depends only on the random inputs. Also note that g is either an AND gate, in which case $g(a,b) = a \cdot b$, or an OR gate, in which case $g(a,b) = 1 - (1-a) \cdot (1-b)$; hence all occurrences of g in the above expression can be replaced by degree-2 polynomials. Moreover, as above, each expression of the form W_i^h can be represented by $h \cdot W_i^1 \oplus (1-h) \cdot W_i^0$. The degree of the meta-polynomial $Q_k^{c_i,c_j}$ is 3.

Output wire of the formula: With this wire we associate a single degree-1 polynomial (in addition to the 4 meta-polynomials $Q_m^{c_i,c_j}$, as described above) which is simply the random input r_m.

We now show that the above construction is perfectly correct and private.

Correctness. Given $\alpha = p_F(x,r)$, it is possible to go over the formula from bottom to top and compute for each wire i the value $\langle W_i^{b_i} \circ c_i \rangle$. Applying this to the output wire m, and since for this wire we also have r_m in the output of p_F (by the construction), one can compute $f(x) = b_m = c_m + r_m$ as required.

Privacy. Consider any output vector, α, of p_F. This output consists of an output bit for each of the polynomials described above. Let α' be the vector α excluding its last bit (i.e., the value r_m); we claim that given any possible input x the output α' is obtained with the same probability. As a first step, by the correctness proof, for each wire i of F, the vector α' completely determines the string $W_i^{b_i}$ and the bit c_i (also note that the b_i's are determined by x). Therefore, if indeed α' is equally possible given any x, and since $r_m = c_m + f(x)$, then α reveals nothing but $f(x)$.

It remains to prove that for every input x the vector α' has the same probability to appear in the output. For this, consider the values $W_i^{b_i}, c_i$ for any i (which, as argued, are totally determined given α'). We show, by induction from top to bottom, that the number of choices for the strings $W_i^{1-b_i}$ that are consistent with α' is independent of x. This is clearly true for the output wire since $|W_m^{1-b_m}| = 0$. For the induction step consider an output wire k of a gate g whose input wires are i, j and assume, without loss of generality, that $c_i = c_j = 0$ (otherwise we just need to permute the 4 polynomials below accordingly). In this case α contains the output of the following 4 meta-polynomials:

$$\begin{aligned}
Q_k^{0,0}(x,r) &= W_i^{b_i,0} \oplus W_j^{b_j,0} \oplus \langle W_k^{g(b_i,b_j)} \circ (g(b_i,b_j) + r_k)\rangle \\
Q_k^{0,1}(x,r) &= W_i^{b_i,1} \oplus W_j^{1-b_j,0} \oplus \langle W_k^{g(b_i,1-b_j)} \circ (g(b_i,1-b_j) + r_k)\rangle \\
Q_k^{1,0}(x,r) &= W_i^{1-b_i,0} \oplus W_j^{b_j,1} \oplus \langle W_k^{g(1-b_i,b_j)} \circ (g(1-b_i,b_j) + r_k)\rangle \\
Q_k^{1,1}(x,r) &= W_i^{1-b_i,1} \oplus W_j^{1-b_j,1} \oplus \langle W_k^{g(1-b_i,1-b_j)} \circ (g(1-b_i,1-b_j) + r_k)\rangle
\end{aligned}$$

Note that at this stage we already assigned values to both strings corresponding to the output wire k of this gate, i.e. to $W_k^{b_k}, W_k^{1-b_k}$ (and clearly $g(b_i,b_j) = b_k$; together with c_k this determines r_k). Hence, the third summand in each of the above meta-polynomials is already fixed. On the other hand, for the input wires i, j we are only committed to the values $W_i^{b_i}, W_j^{b_j}$ and still have the freedom to choose $W_i^{1-b_i}, W_j^{1-b_j}$. Examining the 4 equations above we make the following observations: (a) in the first equation there are no unknowns (in fact, when we choose $W_k^{b_k}$ in the first part of the proof, we choose it so that this equation holds). (b) in the second and third equations there is a unique choice for $W_j^{1-b_j,0}, W_i^{1-b_i,0}$ that satisfies the equation. (c) in the fourth equation we have a constraint on what the string $W_i^{1-b_i,1} \oplus W_j^{1-b_j,1}$ should be. The number of choices that satisfy this constraint is clearly independent of x, as needed.

Efficiency. The complexity of the above construction is dominated by the total size of the strings W_i^b for all input wires i. The length of each string W_i^b depends only on the *depth* of wire i. For a wire of depth d, this length is $O(2^d)$. For example, if F is a balanced formula of size s and depth $\log_2 s$, then the complexity of p_F is $O(s^2)$.

3.1 An Efficiency Improvement

The above proof of privacy leaves some freedom in choosing the random strings (case (c) above). We can get rid of this freedom and thereby obtain some efficiency improvements over the basic construction. To do this, we break the symmetry between the 2 input wires i, j of each gate. We associate with one of these two wires, say j, a shorter string by letting $W_j^{b,1} = W_j^{b,0}$ for $b \in \{0,1\}$. The proof of correctness remains as before. In the proof of privacy, we no longer have freedom in (c), as $W_j^{1-b_j,1}$ was already fixed (because $W_j^{1-b_j,1} = W_j^{1-b_j,0}$); hence, there is a unique way to choose $W_i^{1-b_i,1}$ so as to satisfy the fourth equation (and this is, again, independent of x). The efficiency improvement is obtained since we do not have $|W_i^b| = |W_j^b| = 2(|W_k^b| + 1)$ as before, but rather we can do with $|W_j^b| = |W_k^b| + 1$ and only $|W_i^b| = 2(|W_k^b| + 1)$.

This simple observation already leads to some significant efficiency improvements. If F is a completely balanced formula of size s and depth $d = \log_2 s$, then the total length of the strings W_i^b (which dominates the complexity of p_F) is now only $O(3^d) = O(s^{\log_2 3})$.

For instance, since the OR_n function (OR of n bits) admits such a balanced formula, this basic optimization applies to OR_n yields complexity of $O(n^{\log_2 3})$ (compared with $O(n^2)$ given by the basic construction).

A further efficiency gain may be obtained by skewing the formula tree so that each of its leaves has roughly the same contribution to the overall complexity. We leave open the question of converting a general formula into an optimal equivalent form, and proceed with the interesting special case of the function OR_n (or, equivalently, AND_n). A useful feature of this function is that there is a complete freedom in choosing the shape of its formula: *any* binary tree T with n leaves naturally induces a formula F_T for OR_n, where the leaves are labeled by the n distinct inputs and the internal nodes by binary OR gates. Hence, our problem is equivalent to finding a binary tree T of size n which minimizes the total weight of its n vertices subject to the following constraints: (1) the root has weight 0; (2) if v is an internal node of weight w, then the weights of its two sons are $w+1$ and $2(w+1)$. It can be shown that such a tree T of total weight $n \cdot 2^{O(\sqrt{\log n})}$ can be efficiently constructed (details omitted for lack of space). Hence, there is a perfect degree-3 representation for OR_n of complexity $n \cdot 2^{O(\sqrt{\log n})}$.

4 Perfect Randomizing Polynomials from Branching Programs

In this section we construct perfect randomizing polynomials from a branching program representation. First, in Section 4.1, we give an overview of our solutions and provide some background and intuition. Then, in Section 4.2, we present a construction for CBP (which includes *deterministic* branching programs as a special case) and state our result for NBP whose proof is omitted from this version.

4.1 Overview of Constructions

Before describing our new solutions, it is instructive to review some previous ideas and techniques from [18,19] on which we rely. The previous construction of (nonperfect) randomizing polynomials from branching programs [19] uses an NBP representation, and is based on the following two facts.

Fact 1. [18] Given a mod-q NBP of size ℓ computing f, there exists a function $L(x)$, mapping an input x to an $\ell \times \ell$ matrix over $\mathbb{F} = \text{GF}(q)$, such that:

- Each entry of $L(x)$ is a degree-1 polynomial in a single variable x_i.
- $f(x)$ is in one-to-one correspondence with $\text{rank}(L(x))$. Specifically, if $f(x) = 1$ then $L(x)$ is of full rank, and if $f(x) = 0$ then its rank is one less than full.

Fact 2. [19] Let M be an arbitrary square matrix over $\mathbb{F}$, and R_1,R_2 be independent uniformly random matrices of the same dimension. Then, the distribution of the random variable $R_1 M R_2$ depends only on $\text{rank}(M)$. Moreover, if $\text{rank}(M) \neq \text{rank}(M')$ then $\text{SD}(R_1 M R_2, R_1 M' R_2) > \epsilon_q$, where ϵ_q is a constant depending only on q.

The final degree-3 representation, based on the above two facts, has the form $p(x, R_1, R_2) = R_1 L(x) R_2$, where both the random inputs and the output vector are parsed as matrices. Randomizing polynomials of this form cannot possibly achieve perfect correctness: setting all of the random inputs to 0 will always yield an all-0 output vector. A natural solution that comes to mind is to replace R_1 and R_2 by uniform *nonsingular* matrices. It is easy to see that perfect correctness will be achieved, and it can also be argued that the perfect privacy will not be violated. However, it is not clear how to incorporate random nonsingular matrices into the randomizing polynomials framework; in fact, it follows by a divisibility argument that it is *impossible* to generate a uniformly random nonsingular matrix over a finite field $\mathbb{F}$ from (finitely many) uniform random elements of $\mathbb{F}$, regardless of the complexity or the degree of such a generation.

To get around this problem, we take a closer look at the matrices $L(x)$ generated by the mapping L. It turns out that, for matrices *of this special form*, it is possible to pick R_1 and R_2 from carefully chosen *subgroups* of nonsingular matrices, so that: (1) random matrices from these subgroups can be generated by degree-1 polynomials; (2) $R_1L(x)R_2$ achieves perfect privacy and correctness with respect to the *number* of accepting paths (mod q) on input x. This approach gives our solution for CBP, presented next. However, it falls short of providing an efficient solution for their nondeterministic counterparts, except when the modulus q is small.

4.2 Counting Branching Programs

Throughout this section let $\mathbb{F} = \mathrm{GF}(q)$, where q is an arbitrary prime. Our goal is to convert a mod-q counting branching program computing $f : \{0,1\}^n \to \mathbb{F}$ into an efficient representation of f by perfect degree-3 randomizing polynomials. As outlined above, we start with a refined version of Fact 1. Its proof is similar to that of Fact 1 (from [18]) and is omitted.

Lemma 1. *Suppose there is a mod-q CBP of size ℓ computing f. Then, there exists a function $L(x)$, mapping an input x to an $\ell \times \ell$ matrix over $\mathbb{F} = \mathrm{GF}(q)$, such that:*

- *Each entry of $L(x)$ is a degree-1 polynomial in a single input variable x_i.*
- *$L(x)$ contains the constant -1 in each entry of its second diagonal (the one below the main diagonal) and the constant 0 below this diagonal.*
- *$f(x) = \det(L(x))$.*

Our variant of Fact 2 relies on the following simple randomization lemma.

Lemma 2. *Let $\mathcal{H}$ be a set of square matrices over $\mathbb{F}$, and $\mathcal{G}_1, \mathcal{G}_2$ be multiplicative groups of matrices of the same dimension as $\mathcal{H}$. Denote by '$\sim$' the equivalence relation on $\mathcal{H}$ defined by: $H \sim H'$ iff there exist $G_1 \in \mathcal{G}_1, G_2 \in \mathcal{G}_2$ such that $H = G_1H'G_2$. Let R_1, R_2 be uniformly and independently distributed matrices from $\mathcal{G}_1, \mathcal{G}_2$, respectively. Then, for any H, H' such that $H \sim H'$, the random variables R_1HR_2 and $R_1H'R_2$ are identically distributed.*

Lemma 2 will be instantiated with the following matrix sets.

Definition 2. *Let $\mathcal{H}$ be the set of $\ell \times \ell$ matrices over $\mathbb{F} = \mathrm{GF}(q)$ containing only -1's in their second diagonal (the diagonal below the main diagonal), and 0's below the second diagonal. Define two matrix groups $\mathcal{G}_1$ and $\mathcal{G}_2$ as follows:*

- *$\mathcal{G}_1$ consists of all matrices with 1's on the main diagonal and 0's below it.*
- *$\mathcal{G}_2$ consists of all matrices with 1's on the main diagonal and 0's in all of the remaining entries except, perhaps, those of the rightmost column.*

From now on, '$\sim$' denotes the equivalence relation on $\mathcal{H}$ induced by $\mathcal{G}_1, \mathcal{G}_2$, as defined in Lemma 2.

The following lemma shows that a matrix from $\mathcal{H}$ can be brought into a canonical form, uniquely defined by its determinant, by multiplying it from the left by some $G_1 \in \mathcal{G}_1$ and from the right by some $G_2 \in \mathcal{G}_2$.

Lemma 3. *For any $H \in \mathcal{H}$ there exist $G_1 \in \mathcal{G}_1$ and $G_2 \in \mathcal{G}_2$ such that G_1HG_2 contains -1's in its second diagonal, $\det(H)$ in its top-right entry, and 0's elsewhere.*

Proof. Consider two types of matrix operations: (a) Add to row i some multiple of row $i' > i$; (b) Add to the last column a multiple of some other column. As illustrated in Figure 1, a matrix $H \in \mathcal{H}$ can be transformed, using a sequence of (a) and (b) operations, to a matrix H_0 containing -1's in its second diagonal, an arbitrary value in its top-right entry, and 0's elsewhere. Note that none of these operations changes the determinant, and hence $\det(H_0) = \det(H)$. It follows that the top-right entry of H_0 must be equal to its determinant. We conclude the proof by observing that each operation of type (a) is a left-multiplication by a matrix from $\mathcal{G}_1$, and each operation of type (b) is a right-multiplication by a matrix from $\mathcal{G}_2$. □

$$\begin{pmatrix} * & * & * & * & * & * \\ -1 & * & * & * & * & * \\ 0 & -1 & * & * & * & * \\ 0 & 0 & -1 & * & * & * \\ 0 & 0 & 0 & -1 & * & * \\ 0 & 0 & 0 & 0 & -1 & * \end{pmatrix} \overset{\underline{(a)}}{\Longrightarrow} \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & * \\ -1 & 0 & 0 & 0 & 0 & * \\ 0 & -1 & 0 & 0 & 0 & * \\ 0 & 0 & -1 & 0 & 0 & * \\ 0 & 0 & 0 & -1 & 0 & * \\ 0 & 0 & 0 & 0 & -1 & * \end{pmatrix} \overset{\underline{(b)}}{\Longrightarrow} \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & * \\ -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \end{pmatrix}$$

Fig. 1. Bringing a matrix $H \in \mathcal{H}$ to a canonical form H_0

The following is an easy corollary.

Lemma 4. *Let $\mathcal{H}, \mathcal{G}_1, \mathcal{G}_2$ be as in Definition 2. Then, for any $H, H' \in \mathcal{H}$, $\det(H) = \det(H')$ implies $H \sim H'$.*

Our final randomizing polynomials construction is given in the next theorem.

Theorem 1. *Suppose that f can be computed by a mod-q counting branching program of size ℓ. Then, there exists (constructively) a representation of f by* perfect *degree-3 randomizing polynomials over $\mathbb{F} = \mathrm{GF}(q)$, with output complexity $\binom{\ell+1}{2}$ and randomness complexity $\binom{\ell}{2} + \ell - 1$.*

Proof. Consider the polynomial vector $p(x, r^1, r^2) = R_1(r^1)L(x)R_2(r^2)$, where L is as promised by Lemma 1, and R_1 (resp., R_2) is a degree-1 mapping of the random inputs r^1 (resp., r^2) to a uniformly random matrix from $\mathcal{G}_1$ (resp., $\mathcal{G}_2$). Note that the number of random field elements that r^1 and r^2 should contain is $\binom{\ell}{2}$ and $\ell - 1$, respectively. Hence the specified randomness complexity. For the output complexity, note that it is enough to include the $\binom{\ell+1}{2}$ entries of the output matrix on or above the main diagonal. The perfect privacy of p follows from Lemmas 1,2, and 4. Finally, its perfect correctness follows from the fact that the determinant of the output matrix is always equal to $\det(L(x))$, which by Lemma 1 is equal to $f(x)$. □

Due to lack of space, we omit the full treatment of the nondeterministic case. If the modulus q is small, the problem reduces to the previous counting case. In general, however, we are only able to obtain perfect correctness by relaxing the privacy requirement. The main relevant theorem is the following:

Theorem 2. *Given a mod-q NBP of size ℓ computing f and a security parameter k, it is possible to compute in time poly($\ell, \log q, k$) a perfectly-correct, 2^{-k}-private, degree-3 randomizing polynomials representation of f over $\mathrm{GF}(2)$.*

Remark. An alternative approach for obtaining perfect constant-round protocols for CBP is to reduce the problem to an *inversion* of a *triangular* matrix. Specifically, the value of the CBP may be obtained as the top-right entry of $(I - A_x)^{-1}$, where A_x is the adjacency matrix of the graph G_x (cf. [18]). Since $I - A_x \in \mathcal{G}_1$, a straightforward modification of the secure inversion protocol from [1] (essentially replacing random nonsingular matrices by random matrices from $\mathcal{G}_1$) can be used to compute the desired entry with perfect security and a strictly constant number of rounds. A disadvantage of this solution is that it requires more rounds than a protocol based on degree-3 randomizing polynomials.

Acknowledgments. We thank Amos Beimel, Ronald Cramer, Ivan Damgrard, and Serge Fehr for helpful discussions and comments.

References

1. J. Bar-Ilan and D. Beaver. Non-cryptographic fault-tolerant computing in a constant number of rounds. In *Proc. of 8th PODC*, pages 201–209, 1989.
2. D. Beaver. Secure Multi-party Protocols and Zero-Knowledge Proof Systems Tolerating a Faulty Minority. *J. Cryptology*, Springer-Verlag, (1991) 4: 75-122.
3. D. Beaver. Minimal-latency secure function evaluation. EUROCRYPT 2000.
4. D. Beaver, J. Feigenbaum, J. Kilian, and P. Rogaway. Security with low communication overhead. In *Proc. of CRYPTO '90*, pages 62–76.
5. D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols (extended abstract). In *Proc. of 22nd STOC*, pages 503–513, 1990.
6. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. STOC, 1988.
7. C. Cachin, J. Camenisch, J. Kilian, and J. Muller. One-round secure computation and secure autonomous mobile agents. In *ICALP 2000*.
8. R. Canetti. Security and composition of multiparty cryptographic protocols. *J. of Cryptology*, 13(1), 2000.
9. D. Chaum, C. Crépeau, and I. Damgrard. Multiparty unconditionally secure protocols (extended abstract). In *Proc. of 20th STOC*, pages 11–19, 1988.
10. R. Cramer and I. Damgrard. Secure distributed linear algebra in a constant number of rounds. In *Proc. Crypto 2001*.
11. R. Cramer, I. Damgrard, and U. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *Proc. of EUROCRYPT 2000*.
12. R. Cramer, S. Fehr, Y. Ishai, and E. Kushilevitz. Efficient Multi-Party Computation over Rings. Manuscript, 2002.
13. U. Feige, J. Kilian, and M. Naor. A minimal model for secure computation (extended abstract). In *Proc. of 26th STOC*, pages 554–563, 1994.
14. R. Gennaro, Y. Ishai, E. Kushilevitz and T. Rabin. The Round Complexity of Verifiable Secret Sharing and Secure Multicast. In *Proc. 33rd STOC*, 2001.
15. S. Goldwasser and L. Levin. Fair Computation of General Functions in Presence of Immoral Majority. In *CRYPTO '90, LNCS 537,* Springer-Verlag, 1990.
16. O. Goldreich. Secure multi-party computation. www.wisdom.weizmann.ac.il/~oded/pp.html, 2000.
17. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game (extended abstract). In *Proc. of 19th STOC*, pages 218–229, 1987.
18. Y. Ishai and E. Kushilevitz. Private simultaneous messages protocols with applications. In *Proc. of ISTCS '97*, pp. 174-183, 1997.
19. Y. Ishai and E. Kushilevitz. Randomizing Polynomials: A New Representation with Applications to Round-Efficient Secure Computation. In *Proc. of FOCS '00*.

20. J. Kilian. Basing cryptography on oblivious transfer. STOC ’98, pp. 20-31, 1988.
21. Y. Lindell. Parallel Coin-Tossing and Constant-Round Secure Two-Party Computation. In Prof. of Crypto ’01.
22. S. Micali and P. Rogaway. Secure computation. In *Proc. of CRYPTO ’91*.
23. M. Naor, B. Pinkas, and R. Sumner. Privacy Preserving Auctions and Mechanism Design. In *Proc. ACM Conference on Electronic Commerce 1999*, pages 129-139.
24. T. Sandler, A. Young, and M. Yung. Non-interactive cryptocomputing for NC^1. In *Proc. of 40th FOCS*, pages 554–566, 1999.
25. A. C. Yao. Protocols for secure computations (extended abstract). In Proc. of FOCS 1982.
26. A. C. Yao. How to generate and exchange secrets. In Proc. of FOCS 1986.