How to Garble Arithmetic Circuits

Benny Applebaum
School of Electrical Engineering
Tel-Aviv University
Tel-Aviv, Israel
Email: benny.applebaum@gmail.com

Yuval Ishai, Eyal Kushilevitz
Department of Computer Science
Technion
Haifa, Israel
Email: {yuvali,eyalk}@cs.technion.ac.il

Abstract— Yao's garbled circuit construction transforms a boolean circuit $C : \{0,1\}^n \to \{0,1\}^m$ into a "garbled circuit" $\widehat{C}$ along with n pairs of k-bit keys, one for each input bit, such that $\widehat{C}$ together with the n keys corresponding to an input x reveal $C(x)$ and no additional information about x. The garbled circuit construction is a central tool for constant-round secure computation and has several other applications.

Motivated by these applications, we suggest an efficient arithmetic variant of Yao's original construction. Our construction transforms an arithmetic circuit $C : \mathbb{Z}^n \to \mathbb{Z}^m$ over integers from a bounded (but possibly exponential) range into a garbled circuit $\widehat{C}$ along with n affine functions $L_i : \mathbb{Z} \to \mathbb{Z}^k$ such that $\widehat{C}$ together with the n integer vectors $L_i(x_i)$ reveal $C(x)$ and no additional information about x. The security of our construction relies on the intractability of the learning with errors (LWE) problem.

1. Introduction

Yao's *garbled circuit* (GC) construction [41] is an efficient transformation which maps any boolean circuit $C : \{0,1\}^n \to \{0,1\}^m$ together with secret randomness into a "garbled circuit" $\widehat{C}$ along with n pairs of short k-bit keys (K_i^0, K_i^1), such that for any (unknown) input x, the garbled circuit $\widehat{C}$ together with the n keys $K_x = (K_1^{x_1}, \ldots, K_n^{x_n})$ reveal $C(x)$ but give no additional information about x. The latter requirement is formalized by requiring the existence of an efficient *decoder* which recovers $C(x)$ from $(\widehat{C}, K_x)$ and an efficient *simulator* which given $C(x)$ samples from a distribution which is computationally indistinguishable from $(\widehat{C}, K_x)$. The keys are short in the sense that their length k does not depend on the size of C. Yao's celebrated result shows that such a transformation can be based on the existence of any pseudorandom generator [9], [40], or equivalently a one-way function [20].

The GC construction was originally motivated by the problem of secure multiparty computation [40], [18]. Until recently, Yao's construction provided the only general technique for *constant-round* secure computation [41], [8], [27], [25]. The recent breakthrough results on fully homomorphic encryption [16], [17], [10] give an alternative technique for constant-round secure computation that has better (in fact, nearly optimal) asymptotic communication complexity, but is still quite inefficient in practice. Along the years, the GC construction has found a diverse range of other applications to problems such as computing on encrypted data [38], [11], parallel cryptography [4], [3], verifiable computation [15], [6], software protection [19], functional encryption [37], and key-dependent message security [7], [1].

In some natural application scenarios, the GC construction still provides the *only* known technique for obtaining general feasibility results. As a simple example, imagine a scenario of sending a computationally weak device U to the field in order to perform some expensive computation C on sensitive data x. The computation is too complex for U to quickly perform it on its own, but since the input x is sensitive it cannot just send the entire input out. The GC technique provides the only known *non-interactive* solution: In an offline phase, before going to the field, U publishes $\widehat{C}$ and privately keeps the list of keys (K_i^0, K_i^1). Once it observes the input x, it suffices for U to select the corresponding keys K_x and sends them out – an inexpensive operation whose cost is independent of the complexity of C. The rest of the world can, at this point, recover $C(x)$ and nothing else.

It is instructive to take a slightly more abstract view at the features of the GC construction that are useful in the above application. The device U uses its secret randomness r to generate a garbled circuit $\widehat{C}$ and n pairs of keys in an offline phase. When the input x arrives, it selects the n keys K_x corresponding to x. Overall, x is mapped via an efficient randomized transformation into the string $\hat{y} = (\widehat{C}, K_x)$ which reveals the output $y = C(x)$ but hides all additional information about x. Thus $\hat{y}$ can be viewed as a *randomized encoding* of $C(x)$ [22], [4]. An important feature of this encoding is that each input x_i influences only on a *small* part of the encoding $\hat{y}$ (a single short key) via a *simple* function. The simple and short dependency of the encoded output on the inputs x_i makes the above solution efficient. This is the key feature of the GC construction that is used by essentially all of its applications.

The arithmetic setting. Despite its fundamental nature and the wide array of applications, Yao's original result has not

The first author is supported by Alon Fellowship and ISF grant 1155/11.
The second author is supported by ERC Starting Grant 259426, ISF grant 1361/10, and BSF grant 2008411.
The third author is supported by ISF grant 1361/10 and BSF grant 2008411.

been significantly improved or generalized since the 1980s. One longstanding question is that of finding an efficient variant of the GC construction which applies to *arithmetic* circuits. An arithmetic circuit over a ring R is defined similarly to a standard boolean circuit, except that each wire carries an element of R and each gate can perform an addition or multiplication operation over R. Since arithmetic computations arise in many real-life scenarios, this question has a natural motivation in the context of most of the applications discussed above.

Before we formulate our precise notion of garbling arithmetic circuits, let us briefly explain the disadvantages of some natural solution approaches. Yao's original construction requires $\widehat{C}$ to include an "encrypted truth-table" for each binary gate of C. This can be naturally extended to the case of non-binary gates, but at the cost of a quadratic blowup in the size of the input domain (cf. [32]). In particular, this approach is not feasible at all for arithmetic computations over rings of super-polynomial size.

An asymptotically better approach is to implement the arithmetic circuit by an equivalent boolean circuit (replacing each R-operation by a corresponding boolean sub-circuit). Given the bit-representation of the inputs, this approach can be used to simulate the arithmetic computation with an overhead which depends on the boolean complexity of ring operations. While providing reasonable asymptotic efficiency in theory (e.g., via fast integer multiplication techniques), the concrete overhead of this approach is quite high. Moreover, there are scenarios where one does not have access to *bits* of the inputs and must treat them as atomic ring elements. For example, in the context of computing on encrypted data,[1] one may wish to encrypt large integers using an *additively homomorphic* encryption scheme such as Pallier's encryption [33]. In this scenario, there is no efficient non-interactive procedure for obtaining (encrypted) individual bits of the input; the only feasible operations on encrypted inputs include addition and scalar multiplication.

Garbling arithmetic circuits. The above discussion motivates the following arithmetic analogue of Yao's result:

> Can we efficiently transform an arithmetic circuit $C : R^n \to R^m$ into a garbled circuit $\widehat{C}$, along with n *affine* key functions $L_i : R \to R^k$, such that $\widehat{C}$ together with the n outputs $L_i(x_i)$ reveal $C(x)$ and no additional information about x?

We refer to a general transformation as above as an *affine randomized encoding* (ARE) compiler over R. An affine function L_i is of the form $L_i(x_i) = a_i x_i + b_i$ where a_i, b_i are vectors in R^k that may depend on the secret randomness but not on the input x. We view the output of L_i as a "key" and require its length k to be independent of the size of C, as in the boolean case. The above requirement captures the abstract feature discussed above of a "simple" and "short" dependency of the encoded output on the inputs, where the notion of simplicity is adapted to the arithmetic setting.

It is instructive to observe that the original formulation of the GC construction coincides with the above formulation of an ARE compiler over the *binary* field. Indeed, when $R = \mathbb{F}_2$, the i-th key function $a_i x_i + b_i$ outputs a selection between b_i and $a_i \oplus b_i$ depending on the value of x_i, which is equivalent to having x_i select between a pair of short keys. An ARE compiler can be used to efficiently extend the applications of the GC construction to the arithmetic setting while avoiding the need to access individual bits of the input. This can lead to stronger theoretical feasibility results (where parties are restricted to arithmetic computations on ring elements) as well as to efficiency improvements.

1.1. Our Contribution

We present a general framework for constructing ARE compilers along with an efficient instantiation over the ring of integers. Our main compiler transforms an arithmetic circuit $C : \mathbb{Z}^n \to \mathbb{Z}^m$ over integers from a bounded (but possibly exponential) range[2] into a garbled circuit $\widehat{C}$ along with n affine functions $L_i : \mathbb{Z} \to \mathbb{Z}^k$ such that $\widehat{C}$ together with the n integer vectors $L_i(x_i)$ reveal $C(x)$ and no additional information about x. The security of this construction relies on the hardness of the learning with error (LWE) problem of Regev [35].

We also present a simpler direct construction of an ARE compiler over the integers based on a general assumption (the existence of one-way function). This is done by combining Yao's construction with previous information-theoretic randomization techniques via the Chinese Remainder Theorem (see Section 8). While this approach provides a stronger feasibility result, it does not enjoy the efficiency and generality advantages of our main approach. For example, for some natural classes of arithmetic circuits (including ones computing constant-degree polynomials over integers) we obtain a "constant rate" encoding whose length is only a constant multiple of the description length of the function, independently of the integer size or the security parameter (see Section 7.2). This type of efficiency seems inherently impossible to get using the approach of Section 8 or any alternative technique from the literature. The technical core of our LWE-based construction relies on a special "functional encryption" scheme which yields a succinct encoding

[1] In the problem of computing on encrypted data, a Receiver publishes an encryption c of her input x, so that any Sender holding a secret function f can reveal $f(x)$ to the Receiver (and nothing else about f) by sending her a single message that depends on c.

[2] More precisely, the ARE compiler uses a bound U on the values of circuit wires as an additional input, and provides no correctness or privacy guarantees when this bound is not met. Efficient arithmetic computations over integers with polynomial bit-length are captured by Valiant's algebraic complexity class $\mathbf{VP}$ [39], and are believed to be strictly stronger than log-space variants that are captured by algebraic formulas and branching programs [24], [31].

for *affine* circuits (see Section 6). This scheme may be of independent interest.

A new framework for garbled circuits. On the conceptual level, we suggest a new general view of garbled circuits. This view provides a clean reduction from the task of constructing ARE compilers over a general ring R to the construction of a simple primitive over R. (See Section 2.) As a result, any new instantiation of this primitive over a ring R would immediately imply an ARE compiler over R with related efficiency. This has the potential to improve both the efficiency of our LWE-based instantiation (e.g., achieving a constant rate for larger classes of circuits) and its generality (e.g., applying to general circuits over large *finite* fields $\mathbb{F}_p$).[3] Our new framework also leads to a conceptually simpler derivation of Yao's original result for boolean circuits (see Section 2 below). We believe that this more general view leads to a better understanding of Yao's classical result which has remained essentially untouched since the 1980s.

2. Our Techniques

Our starting point is an information-theoretic ARE for *simple* low-depth circuits. Previous randomized encoding techniques from the literature [26], [12], [23], [13] imply ARE compilers over finite rings that are weaker from our main notion of ARE compiler in two ways. First, these compilers can only efficiently apply to restricted classes of circuits such as arithmetic formulas and branching programs. Second, even for these classes, known constructions require the length of each key to be bigger than the representation size of the function (i.e., formula or branching program size), contradicting our requirement of having short keys. A key idea which underlies our approach is that a solution for the second problem can lead to a solution for the first one. Details follow.

Imagine that we have a *key-shrinking gadget* which allows us to transform an affine encoding with long keys c, d into an affine encoding with short keys a, b. Now we can encode an arithmetic circuit C as follows. Instead of individually considering each wire and gate of C as in the original garbled circuit construction, we view C as a composition of "simple" circuits, called *layers*, and will build the encoder by processing one layer at a time, starting from the outputs (top layer) and proceeding towards the inputs (bottom layer).

Assume that C consists of $i+1$ layers of depth 1 each, and that we already encoded the top i layers of C. Specifically, let C' denote the sub-circuit which consists the top i layers, B denote the depth-1 circuit in the bottom layer, and $y' = B(x)$. Namely, $C(x) = C'(B(x))$. Suppose that Enc' is an affine encoding with k-bit long keys for the function $C'(y')$. Then, an encoding Enc of $C(x)$ is obtained using the following three steps:

- (*Substitution*) First we take the encoding $\mathsf{Enc}'(y')$ and substitute y' with $B(x)$. Let $f(x) = \mathsf{Enc}'(B(x))$ denote the resulting randomized function. Since Enc' encodes C', we have that f encodes C. However, even if Enc' is affine (in y'), the encoding f may not be affine (in x) as the layer B may contain multiplication gates. For instance, if the first entry of y' is the product of the pair x_1, x_2, then f will contain outputs of the form $Q = a_1 * (x_1 * x_2) + b_1$.
- (*Affinization*) We turn f into an affine encoder g by applying to each of its outputs the information-theoretic encoder of arithmetic formulas. Here we rely on the fact that the above expression Q is finite and thus has a constant size formula. (See Section 5 for more details.) The problem is that now the keys of the encoding g are longer (by a constant factor) than those of Enc', and so if we repeat this process, the size of the keys will become exponential in the depth.
- (*Key shrinking*) We fix the latter problem by first rearranging terms into blocks (grouping together outputs that depend on the same variable) and then applying the key-shrinking gadget to each output block. As a result, the size of the keys is reduced at the cost of generating additional outputs that do not depend on the inputs and thus can be accumulated in the garbled circuit part.

See Section 6.1 for more details.

Shrinking the keys in the binary case. The above process reduces the construction of ARE compilers to an implementation of the key-shrinking gadget. In the binary case, this can be done quite easily with the help of a standard symmetric encryption scheme. For simplicity, let us treat affine functions as "selector" functions in which a bit s selects one of two vectors K_0 and K_1. We denote this operation by $s \upharpoonright (K_0, K_1) = K_s$. (Selectors and affine functions are essentially equivalent in the binary domain.) Our goal can be described as follows: We are given the function $f(y, c, d) = y \upharpoonright (c, d)$, where y is a bit and c, d are long keys, and we would like to encode it by a function of the form $g(y, c, d; r) = (W, y \upharpoonright (a, b))$. The new keys a, b may depend on c, d and on the randomness r (but not on y) and should be shorter than c and d (e.g., of length equal to the security parameter). The string W can also depend on c, d and r, and may be of arbitrary polynomial length.[4] A simple way to implement g is to employ symmetric encryption, where c is encrypted under the random key a and d under the random key b. Then g can output the two ciphertexts $E_a(c), E_b(d)$ together with the value $y \upharpoonright (a, b)$. The ciphertexts should be permuted in a random order, as otherwise the value of y is leaked and the security of the encoding is compromised.

[3] One concrete approach is to try and adapt our construction to work under the Ring-LWE assumption [30].

[4] Recall that the notion of encoding requires that the value $g(y, c, d; r)$ reveals $f(y, c, d)$, i.e., the value of the key chosen by y, but no other information.

It is not hard to see that g satisfies our syntactic requirements and, in addition, its output allows one to recover the value $y \upharpoonright (c,d)$ by decrypting the corresponding ciphertext using the key $y \upharpoonright (a,b)$. This variant of the gadget assumes that the encryption is verifiable (i.e., decryption with incorrect key can be identified), and it typically leads to a statistically small decoding error (as in the GC variant of [28]). A perfectly correct version can be achieved by appending to the encoding the value $\pi \oplus y$, where π is the random bit which determines the order of the ciphertexts (i.e., the pair $E_a(c), E_b(d)$ is permuted if and only if $\pi = 1$). During the decoding, this additional information points to the ciphertext which should be decrypted, without revealing any additional information. (This version corresponds to the GC variant in [8], [32], [3].)

Shrinking the keys in the arithmetic case. In the arithmetic case, the function f is affine, i.e., $f(y,c,d) = cy + d$, and we would like to encode it by the function $g(y,c,d;r) = (W, ay+b)$ where y is an element from the ring R, a,b,c,d are vectors over R, and the syntactic properties are similar to the previous case (e.g., a and b should be short). For simplicity, assume in the following that R is a finite field. We reduce the key shrinking problem to the following primitive: find a randomized function which maps a pair of keys $c,d \in R^n$ into a pair of (possibly longer) keys $u,v \in R^N$ such that (1) $yu+v$ encodes $yc+d$, and (2) for every choice of y,c,d and z in the support of $yu+v$, the distribution of u conditioned on $yu+v=z$ can be "hidden" (planted) in a linear space of dimension k, proportional to the security parameter. That is, a random linear space containing u is computationally indistinguishable from a totally random linear space. Given this primitive, the key-shrinking gadget g can be constructed as follows: Pick $u,v \in R^N$ using the primitive, pick random $a,b \xleftarrow{R} R^k$ and a matrix $W \in R^{N \times k}$ such that $Wa = u$ and $Wb = v$, and finally output $(W, ya+b)$. Decoding is done by computing the product $W \cdot (ya+b)$, which, by linear algebra, equals to $yu+v$ and thus reveals $yc+d$. The "hiding" requirement implies that no other information is revealed. While we do not know how to implement the primitive over finite fields (nor can we rule out the existence of such an implementation), we implement a similar primitive over the integers under the LWE assumption. The main technical content of our work is devoted to this construction, which is described in detail in Section 6.

3. Preliminaries

We recall the learning with error (LWE) problem, explicitly put forward by Regev [35]. Let κ be a security parameter and let $k = k(\kappa)$ (dimension) and $q = q(\kappa)$ (modulus) be positive integers. Let $\chi = \chi(\kappa)$ (noise distribution) and $\mathcal{S} = \mathcal{S}(k)$ (information distribution) be a pair of probability distributions over $\mathbb{Z}_{q(\kappa)}$. Let $\mathcal{U}_q$ denote the uniform distribution over $\mathbb{Z}_q$.

Definition 3.1 (Learning with Errors). *The* search learning with errors *assumption* $\mathsf{SLWE}(k,q,\chi,\mathcal{S})$ *asserts that for every polynomial* $m = m(k)$ *the following holds. Given the* $\mathsf{LWE}^m(k,q,\chi,\mathcal{S})$ *distribution*

$$(M, r = Ms + e), \text{ where } M \xleftarrow{R} \mathcal{U}_q^{m \times k}, s \xleftarrow{R} \mathcal{S}^k, e \xleftarrow{R} \chi^m,$$

it is infeasible to recover s *with more than negligible probability in* κ. *The* decisional learning with errors *assumption* $\mathsf{DLWE}(k,q,\chi,\mathcal{S})$ *asserts that for every polynomial* $m = m(k)$ *the distribution* $\mathsf{LWE}^m(k,q,\chi,\mathcal{S})$ *is computationally indistinguishable from* $(\mathcal{U}_q^{m \times k}, \mathcal{U}_q^m)$.

In this work both $\mathcal{S}$ and χ will be instantiated with a "rectangular" probability distribution Φ_ρ in which an element of $\mathbb{Z}_q$ is chosen uniformly at random from the interval $[-\rho, +\rho]$, and where $\rho = q^{\Omega(1)}$. While this is somewhat non-standard, it can be shown that, for our choice of parameters (i.e., q is superpolynomial in κ), this variant of DLWE is implied by SLWE in the more standard setting in which $\mathcal{S}$ is uniform over $\mathbb{Z}_q$ and χ is a Gaussian noise of standard deviation $\rho' = \rho^{\Omega(1)} = q^{\Omega(1)}$. (For this we employ the results of [2], [34]. See full version for details.)

In our construction we will let $q = (U + 2^\kappa)^{O(1)}$ and $k = \log^\gamma q$, where κ is a security parameter, U is an upper bound on the range of wire values, and $\gamma > 1$ is a constant. The choice of γ affects the efficiency of the construction. Smaller γ means better efficiency under a stronger assumption. Known approximation algorithms for the lattice shortest vector problem (SVP) imply that the assumption is false for $\gamma < 1$. For $\gamma > 1$, the best known attacks require superpolynomial time and our assumption can be reduced to the conjectured hardness of approximating SVP [34] to within subexponential factors (i.e., 2^{k^ϵ} where ϵ vanishes with γ). (See [35], [34], [29], [36] for a survey of DLWE and related lattice problems.)

4. Affine Randomized Encodings

In this section, we define our main goal of garbling arithmetic circuits. This goal can be viewed as an instance of the general notion of randomized encoding put forward in [22], [4]. Here we would like to transform a "complex" arithmetic computation into a "simple" randomized arithmetic computation, where the output of the latter encodes the output of the former and (computationally) hides all additional information about the input. Our notion of simplicity directly generalizes the features of Yao's construction in the boolean case: On input $x = (x_1, \ldots, x_n) \in \mathbb{Z}^n$, the output $(W, L_1, \ldots, L_n)$ includes a "garbled circuit part" W, which is independent of the input and depends only on bits of randomness, and n short "keys" where the i-th key is computed by applying an *affine* (degree-1) function L_i, determined only by the randomness, to the i-th input x_i. The complexity of the construction should be polynomial in the size of the circuit being encoded, a cryptographic security

parameter, and an upper bound on the bit-length of the values of circuit wires. Furthermore, the size of the keys should be independent of the circuit size.

We now formalize the above. We use a minimal model of arithmetic circuits which allows only addition, subtraction, and multiplication operations. The circuit does not have access to the bit representation of the inputs. In the randomized case, we allow the circuit to pick random inputs from $\{0,1\}$.

Definition 4.1 (**Arithmetic circuits**). *An* arithmetic circuit *C is syntactically similar to a standard boolean circuit, except that the gates are labeled by '+' (addition), '-' (subtraction) or '*' (multiplication). Each input wire can be labeled by an input variable x_i or a constant $c \in \{0,1\}$. In a* randomized *arithmetic circuit, input wires can also be labeled by random inputs r_j. Given a ring R, an arithmetic circuit C with n inputs and m outputs naturally defines a function $C^R : R^n \to R^m$. In the randomized case, $C^R(x)$ outputs a distribution over R^m induced by a uniform and independent choice of r_j from $\{0,1\}$. We denote by $C^R(x;r)$ the output of C^R on inputs $x_i \in R$ and independent random inputs $r_j \in \{0,1\}$. When R is omitted, it is understood to be the ring $\mathbb{Z}$ of integers.*

Definition 4.2 (**Affine circuits**). *We say that a randomized arithmetic circuit C is* affine *if its outputs are partitioned into $n+1$ vectors $(y_0, y_1, \ldots, y_n)$, where y_0 depends only on the random inputs r_j, and each y_i is of the form $a_i * x_i + b_i$ where a_i and b_i are vectors of ring elements that depend only on the random inputs r_j. In the context of our construction, we will refer to y_0 as the* garbled circuit part *and to the other y_i, $1 \le i \le n$, as* keys. *For $1 \le i \le n$, we denote the i-th key length $|y_i|$ by ℓ_i.*

We now define the notion of an affine randomized encoding compiler, which captures our main goal of garbling arithmetic circuits. Here we restrict the attention to the ring of integers $R = \mathbb{Z}$, though later we will also refer to variants that apply to arithmetic computations over finite fields or rings.

Definition 4.3 (**ARE Compiler: Syntax**). *An* affine randomized encoding (ARE) compiler *is a PPT algorithm T with the following inputs and outputs.*

- *Inputs: a security parameter 1^κ, an arithmetic circuit C over the integers, and a positive integer U bounding the values of circuit wires.*
- *Outputs: randomized* affine *circuits* Enc *(encoder) and* Sim *(simulator), and a* boolean *circuit* Dec *(decoder). By default, we require that* Enc *output short keys. That is, each key length ℓ_i (see Definition 4.2) should be bounded by a fixed polynomial in κ and $\log U$, independently of $|C|$. We will sometimes refer to* Enc *as an affine randomized encoding (or ARE) of C.*

We require an ARE compiler to satisfy the following correctness and privacy requirements.

Definition 4.4 (**ARE Compiler: Correctness**). *A polynomial-time algorithm T, as above, is (perfectly)* correct *if for every positive integer κ, arithmetic circuit C with n inputs, and positive integer U, the following holds. For every input $x \in \mathbb{Z}^n$ such that the value of each wire of C on input x is in the interval $[\pm U]$,*

$$\Pr[\mathsf{Dec}(\mathsf{Enc}(x;r)) = C(x)] = 1,$$

where $(\mathsf{Enc}, \mathsf{Dec}, \mathsf{Sim}) \stackrel{R}{\leftarrow} T(1^\kappa, C, U)$ and the input for Dec *is given in standard binary representation.*

Towards defining privacy, we consider the following game between an adversary $\mathcal{A}$ and a challenger. (All integers are given to the adversary in a standard binary representation.) First, $\mathcal{A}$ on input 1^κ outputs an arithmetic circuit C with input length n, a positive integer U, and an input $x \in \mathbb{Z}^n$ such that the value of each wire of C on input x is in the interval $[\pm U]$. The challenger then lets $(\mathsf{Enc}, \mathsf{Dec}, \mathsf{Sim}) \stackrel{R}{\leftarrow} T(1^\kappa, C, U)$ and picks a random bit b as well as random inputs r_e for Enc and r_s for Sim. It feeds the adversary with $\mathsf{Enc}(x; r_e)$ if $b = 0$ or with $\mathsf{Sim}(C(x); r_s)$ if $b = 1$, together with the randomness used by T. The adversary outputs a guess b'. The adversary wins the game if $b' = b$.

Definition 4.5 (**ARE Compiler: Privacy**). *An ARE compiler T, as above, is (computationally)* private *if every nonuniform adversary $\mathcal{A}$ of size $\mathrm{poly}(\kappa)$ wins the above game with at most $1/2 + \kappa^{-\omega(1)}$ probability. We say that T is* statistically private *if the above holds for an unbounded $\mathcal{A}$, and is* perfectly private *if the winning probability is $1/2$.*

The above definitions can be naturally modified to capture ARE over a *finite* ring R by letting C,Enc,Dec, and Sim be defined by arithmetic circuits over R. In this case, Enc and Sim are allowed to sample uniform elements from R and we also eliminate the bound U on the values of circuit wires.

Remark 4.6 (Applicability of previous techniques). *Previous randomized encoding techniques from the literature [26], [12], [23], [13] imply ARE compilers over finite rings that are weaker from our main notion of ARE compiler in two ways. First, these compilers can only efficiently apply to restricted classes of circuits such as arithmetic formulas and branching programs. Second, even for these classes, the previous constructions yield AREs in which the length of each key is bigger than the representation size of the function (i.e., formula or branching program size), contradicting our requirement of having short keys. See Section 5 for further details on these previous constructions. In Section 8, we describe an indirect approach for combining these previous techniques with Yao's construction for boolean circuits to yield a general ARE compiler that meets the above definition. However, this compiler does not have the efficiency*

and generality advantages of the main compiler we present in Section 7.

We will rely on natural concatenation and composition properties of randomized encodings: a circuit C with multiple outputs can be encoded by concatenating the encodings for each output [4, Lemma 4.9], and two encoders can be composed by using one to encode the function computed by the other [3, Lemma 3.5]. In fact, we will use an extension of these properties that allows to concatenate or compose polynomially many encodings, which follows from a standard hybrid argument.

In the following, we will be more loose in our use of notation. We will sometimes implicitly identify a concrete function f with an arithmetic circuit computing it and (following [4]) will denote by $\hat{f}$ the encoder Enc corresponding to f. We will also replace arithmetic circuits by weaker arithmetic computational models such as arithmetic branching programs. Finally, while our presentation does not make the compiler T explicit, it is easy to verify that the encoder, decoder, and simulator we describe can be efficiently constructed given the inputs for T.

5. Affinization Gadgets

The first building block of our ARE compiler is an *affinization* gadget, which provides a statistical ARE for *simple* arithmetic functions. To this end, we rely on perfect ARE compilers for functions in low arithmetic complexity classes that are implicit in the secure computation literature. In particular, any arithmetic circuit C over a finite ring R for which each output depends on at most d inputs admits a perfect ARE with key length $2^{O(d)}$. More generally:

Fact 5.1 (Affinization gadget over finite rings)**.** *There is a perfect ARE compiler for arithmetic branching programs or arithmetic formulas over any finite ring R. For a formula or branching program of size s, the key is of length $O(s^2)$.*

The above result can be obtained via a randomization of an iterated matrix product [26], [12], [14], [13] or the determinant [21], [23], [13]. See [13], [6] for further discussion and pointers.

We would like to obtain a similar variant over the integers. For this, we first embed the computation in a sufficiently large finite ring $\mathbb{Z}_p$ and then encode the computation modulo p over the integers by adding a sufficiently large random multiple of p. The latter step relies on the following fact.

Lemma 5.2 (Encoding modular reduction)**.** *Let X, p, μ be positive integers such that $X > p$. Define $f : [-X, X] \to \mathbb{Z}_p$ by $f(x) = x \bmod p$. Then, for a uniformly random $r \in [0, \mu]$, the function $\hat{f}(x; r) = x + rp$ (computed over the integers) is a randomized encoding of f with statistical privacy error $\varepsilon \leq 2X/(p\mu)$.*

Combining Fact 5.1 and Lemma 5.2, we get our main affinization gadget.

Corollary 5.3 (Affinization gadget over the integers)**.** *There is a perfectly correct, statistically private ARE compiler for arithmetic branching programs or arithmetic formulas over the integers. For a formula or branching program f of size s, the resulting keys have length $O(s^2)$ and bit-length $O(\log U + \kappa)$, where U is an upper bound on the absolute values of the inputs and outputs, and κ is a statistical security parameter.*

Proof: We start by viewing f as a function over $\mathbb{Z}_p$, choosing large enough p (say, $p = O(U)$) to avoid modular reduction. We then apply the ARE of Fact 5.1, yielding a perfect ARE $\hat{f}'$ over $\mathbb{Z}_p$ with key length $O(s^2)$. Viewing the key functions of this ARE as functions over the integers, their outputs have bit-length $O(\log U)$. Applying Lemma 5.2 to each entry of $\hat{f}'$, we get a statistical ARE $\hat{f}$ over the integers with key-length $O(s^2)$ and bit-length $O(\log U + \kappa)$. ■

We will mainly apply the affinization gadget to the function $f(x_1x_2 + x_3)$, where $x_i \in [0, 2^\lambda]$ and operations are over the integers. In the full version we give a self-contained description of this instance of the affinization gadget. Specifically, we show that f admits a statistically $2^{-\kappa}$-private ARE over the integers with non-negative keys a_i, b_i of length $\ell_i \leq 2$ and bit-length $2\lambda + \kappa + 2$. We refer to this version of the affinization gadget as the *basic* gadget and to the version of Corollary 5.3 as the *generalized* gadget.

6. The Key-Shrinking Gadget

In this section we describe an efficient LWE-based implementation of the key-shrinking gadget: a computationally private randomized encoding of the function $g(y, c, d) = yc + d$, where $c, d \in \mathbb{Z}^n$ and $y \in \mathbb{Z}$ (see below for restrictions on the input domain), by a function of the form

$$\hat{g}(y, c, d; r) = (W(c, d, r), y \cdot a(r) + b(r))$$

where the outputs of a and b are shorter than c and d. More concretely, the length requirement is that a and b output vectors in $\mathbb{Z}^k$ for k which can depend (polynomially) on κ and $\log U$ but not on n. In the main instance of our final construction, n will be a constant multiple of k. We will also need the bit-length of each entry of a (resp., b) to be smaller by a constant factor than the maximal bit-length of each entry of c (resp., d). In contrast to our main notion of ARE, we do not restrict the functions W, a, b other than being computable in polynomial time from the *bits* of their inputs. Since we will only apply the gadget on inputs c, d that are derived from bits of randomness (rather than from an input x_i of the original circuit), this will not violate the syntactic requirements of ARE in our final construction.

6.1. The Construction

Intuition. The encoding $\hat{g}$ follows the outline described in Section 2. First, c and d are represented by u and v via

an invertible randomized mapping, i.e., by multiplying by a large constant Δ and adding some noise. (The mapping is invertible because of the low magnitude of the noise). Then the vectors u and v are planted in a random linear space W of a low dimension k. The space W is made public. Now every linear combination of u and v lies in W, and so it can be succinctly described by its coefficients with respect to W. In particular, to reveal the output $yc+d$, it suffices for the encoding to reveal the coefficients of its representation $yu + v$. We proceed with a formal description.

Parameters. Positive integers $k, \rho_a, \rho_b, \mu_c, \mu_d, \Delta$ and a prime number q. The parameter k determines the key length of the encoding and ρ_a, ρ_b determine a bound on the key entries. The parameters Δ and μ_c, μ_d are used to define the (randomized) embedding of c and d in the (larger) vectors u and v. We assume that $k < n$, $\rho_c \geq \rho_a$, and $\rho_d \geq \rho_b$, where ρ_c, ρ_d upper-bound the values of c, d. (These upper-bounds are given as part of the input.) All parameters can be chosen as functions of κ and U under constraints that will be specified later (see Lemma 6.1 and the following discussion). Let Φ_α denotes the uniform distribution over the integers in the interval $[\pm\alpha]$.

We construct the encoding $\hat{g}$ as follows.

Input: (y, c, d) where $y \in [\pm U], c \in [0, \rho_c]^n, d \in [0, \rho_d]^n$.

1) Choose $e_c \xleftarrow{R} \Phi^n_{\mu_c}$ and $e_d \xleftarrow{R} \Phi^n_{\mu_d}$.
2) Let $u = c \cdot \Delta + e_c$ and $v = d \cdot \Delta + e_d$.
3) Choose $a \xleftarrow{R} [0, \rho_a]^k$ and $b \xleftarrow{R} [0, \rho_b]^k$.
4) Choose $W \xleftarrow{R} \mathcal{U}_q^{n \times k}$ conditioned on $Wa = u \pmod q$ and $Wb = v \pmod q$.

Output: $(W, ya + b)$.

Lemma 6.1. *Suppose that* $\mathsf{DLWE}(k, q, \Phi_{\mu_c}, \Phi_{\rho_a/2})$ *holds, and that* $k \leq \mathrm{poly}(\kappa), n \leq \mathrm{poly}(k)$, $U\rho_a/\rho_b \leq \mathrm{neg}(\kappa)$, $U\mu_c/\mu_d \leq \mathrm{neg}(\kappa)$, $\Delta > 3\mu_d$, *and* $q > 3\Delta(U\rho_c + \rho_d)$. *Then, there is an efficient decoder* Dec *such that* $\mathsf{Dec}(\hat{g}(y, c, d)) = yc + d$ *and an efficient simulator* Sim *such that* $\mathsf{Sim}(yc + d)$ *is computationally indistinguishable from* $\hat{g}(y, c, d)$.

Sketch: We decode $yc + d$ from $(W, f = ya + b)$ as follows. (1) Compute $W \cdot f$ modulo q and represent the entries of the resulting vector as integers in the interval $[\pm(q-1)/2]$. (2) Divide each entry of the vector by Δ, and round the result to the nearest integer. To see that the decoder is perfectly correct note that: (a) By linear algebra, the outcome of step (1) is equal to $yu + v$ modulo q; (b) The above equality holds over the integers as well (since q is large enough to avoid wraparound); (c) Δ is large enough to ensure that there is no rounding error in step (2).

Given $z = yc + d$ we simulate the output (W, f) of $\hat{g}(y, c, d)$ as follows. Choose $f \xleftarrow{R} [0, \rho_b]^k$ and a random matrix $W \xleftarrow{R} \mathcal{U}_q^{n \times k}$ conditioned on $Wf = w \pmod q$ where $w = z\Delta + e_d$ and $e_d \xleftarrow{R} \Phi^n_{\mu_d}$. In the full version we use a hybrid argument to show that, under DLWE, the above distribution is computationally indistinguishable from $\hat{g}(y, c, d)$. Roughly speaking, we first show that $\hat{g}(y, c, d)$ is statistically close to a distribution $D(y, c, d)$ in which instead of planting the vectors u and v in $\mathrm{span}(W)$, we plant the vectors u and w (as defined above). Then, we show that, under DLWE, the vector u is hidden in $\mathrm{span}(W)$ and W is indistinguishable from a random matrix which spans w. ■

Setting the parameters. Suppose that $\rho_a = 2^\kappa$, $\rho_b = U \cdot 2^{2\kappa}$, $\mu_c = 2^\kappa$, $\mu_d = U \cdot 2^{2\kappa}$, $\Delta = U \cdot 2^{2\kappa+2}$, $q = O(2^{2\kappa} U(U\rho_c + \rho_d))$ is prime, $k = \log^\gamma q$ (for $\gamma > 1$), and $n \leq \mathrm{poly}(k)$. With this choice of parameters, the conditions of Lemma 6.1 are met under a conservative instance of the DLWE assumption. Note that in the above choice of parameters, both the *key-length* (i.e., number of entries) and the *bit-length* of the keys a, b are independent of the corresponding parameters of the inputs c, d. In particular, they can be polynomially smaller.

Complexity. Let $\tau = \lceil \log q \rceil = O(\kappa + \log U + \log \rho_c + \log \rho_d)$, and let W_i denote the i-th column of the matrix$W$. We partition the output of the gadget into three parts: (1) Elements that depend on the "key selector" y, namely the vector $y \cdot a(r) + b(r)$ which consist of $k = \tau^\gamma$ elements each of bit length smaller than τ. (2) The $k-2$ columns $W_1, \ldots, W_{k-2}$ which consist of $n(k-2) = O(n\tau^\gamma)$ elements of bit length τ; and (3) the columns W_{k-1}, W_k which consist of $O(n)$ elements of bit length τ. We distinguish the $k-2$ leftmost columns from the other columns because the former part of the encoding consist only of public coins, and can be reused among different instantiations of the gadget. (See full version for more details.) This will allow us to reduce the length of the ARE when employing many copies of the key-shrinking gadget.

7. Garbling Arithmetic Circuits

In this section we combine the affinization gadget and the key-shrinking gadget for obtaining our main result: an efficient ARE compiler for general arithmetic circuits over integers from a bounded range. Some optimizations for specific families of functions (e.g., low degree polynomials) will be presented in Section 7.2.

7.1. The Main Construction

Let C be an arithmetic circuit. Instead of individually considering each wire and gate of C as in the original garbled circuit construction, we will build the encoder by processing one *layer* at a time. For simplicity, we assume that C is already given in a layered form. That is, $C(x) = B_1 \circ B_2 \circ \cdots \circ B_h(x)$ where each B_i is a depth-1 circuit.[5] We

[5]The construction can be generalized by letting each B_i compute any "simple" arithmetic function that can be handled by the generalized affinization gadgets of Corollary 5.3 (e.g., an arithmetic $\mathbf{NC^1}$ circuit or a sequence of polynomial-size arithmetic branching programs). This generalization can give useful efficiency tradeoffs, see Section 7.2 below.

denote by y^i (values of) variables corresponding to the input wires of layer B_i. That is, $y^i = B_{i+1} \circ \cdots \circ B_h(x)$, where $y^0 = C(x)$ and $y^h = x$. We denote by C^i the function mapping y^i to the output of C; that is, $C^i(y^i) = B_1 \circ \ldots \circ B_i(y^i)$, where $C^0(y^0)$ is the identity function on the outputs.

The ARE compiler, on inputs κ, U, C, starts by setting the parameters for the key-shrinking gadget. The bounds ρ_c, ρ_d are set to $2^{\eta(\kappa+\log U)}$ for some constant $\eta > 1$. (The constant η is derived from the parameters of the affinization gadget; for our basic affinization gadget, $\eta = 4$ suffices.) The remaining parameters are picked as discussed following Lemma 6.1. The value of n may vary in different invocations of the gadget, but will be at most $2k\phi$, where ϕ is the maximal fan-out of gates of C.

The compiler builds the encoding Enc in an iterative fashion, processing the layers of C from top (outputs) to bottom (inputs). It starts with a trivial encoding of the identity function C^0. In iteration i, $i = 1, 2, \ldots, h$, it transforms an ARE for $C^{i-1}(y^{i-1})$ into an ARE for $C^i(y^i)$ by first *substituting* $B_i(y^i)$ into y^{i-1}, then applying the *affinization gadget* to bring the resulting function into an affine form (at the cost of increasing the size of the keys), and finally applying the *key-shrinking* gadget to reduce the size of the keys (at the cost of generating additional outputs that do not depend on the inputs). This process terminates with an ARE of $C^h(y^h) = C(x)$.

More precisely, the encoder Enc produced by the ARE compiler is obtained as follows. For simplicity, we treat encoders as probabilistic circuits and omit their random inputs from the notation.

1) Let $\mathsf{Enc}^0(y^0) = y^0$ be the identity function on the variables y^0 (one variable for each output of C).
2) For $i = 1, 2, \ldots, h$, obtain an encoding $\mathsf{Enc}^i(y^i)$ of $C^i(y^i)$ from an encoding $\mathsf{Enc}^{i-1}(y^{i-1})$ of $C^{i-1}(y^{i-1})$ using the following three steps:
   a) **Substitution.** Let $F(y^i) = \mathsf{Enc}^{i-1}(B_i(y^i))$. It is clear that if Enc^{i-1} encodes C^{i-1}, then F encodes C^i. However, even if Enc^{i-1} is affine, F is no longer affine: for instance, if the first output of B_i is $y_1^{i-1} = y_1^i * y_2^i$, then F will contain outputs of the form $Q = a_1 * (y_1^i * y_2^i) + b_1$.
   b) **Affinization.** Turn F into an affine encoder G of the same function by applying to each output that depends on two inputs y_j^i the basic affinization gadget. For instance, a term Q as above can either be handled directly via the generalized gadget of Corollary 5.3, or by applying the basic gadget to $Q = z * y_2^i + b_1$ and substituting $a_1 y_1^i$ into z. The resulting encoding of Q can be written in the form $Q' = (a'_1 y_1^i + b'_1, a'_2 y_2^i + b'_2, w)$ where a'_i, b'_i, w are vectors in $\mathbb{Z}^2$ that depend only on random inputs and whose bit-length is $O(\log U + \kappa)$. Applying this transformation to every term Q in the output of F and concatenating different affine functions of the same input, we get an affine encoding G of C^i. However, now the keys of G are longer than those of Enc^{i-1} (by a factor which is at most twice the fan-out of B_i) and have bigger entries.
   c) **Key shrinking.** To avoid the exponential blowup of keys, the compiler applies the key-shrinking gadget. For every output $Q = c_j y_j^i + d_j$ of G that has length $n > k$ or has large key entries (i.e., $c_{j,h} > \rho_a$ or $d_{j,h} > \rho_b$ for some h), the key-shrinking gadget is applied to bring it to the form $(W, a_j y_j^i + b_j)$, where $a_j \in [0, \rho_a]^k$, $b_j \in [0, \rho_b]^k$, and a_j, b_j, W depend only on random inputs. (In fact, in our implementation of the key-shrinking gadget a_j, b_j are picked uniformly at random from their domain.) The W entries will be aggregated and will form the garbled circuit part of the output. Let $\mathsf{Enc}^i(y^i)$ be the affine encoding resulting from this step.
3) Output $\mathsf{Enc}(x) = \mathsf{Enc}^h(x)$.

The decoder and simulator are obtained by applying a similar iterative process based on the decoders and simulators of the two gadgets. Correctness and privacy follow from the natural concatenation and composition properties of randomized encodings (see full version). Indeed, for all $1 \le i \le h$, the randomized function $E^i(x) = \mathsf{Enc}^i(B_{i+1} \circ \cdots \circ B_h(x))$ is a randomized encoding of $E^{i-1}(x) = \mathsf{Enc}^{i-1}(B_i \circ \cdots \circ B_h(x))$. It follows from the composition property that $\mathsf{Enc}(x) = E^h(x)$ is a randomized encoding of $C(x) = E^0(x)$, as required. Note that since gadgets are applied on the values of all intermediate wires y_j^i, the correctness and privacy of the final encoding Enc is only guaranteed on inputs x for which the absolute value of every wire in C is bounded by U.

The security properties of the construction are captured by the following main theorem:

Theorem 7.1. *Suppose $\gamma > 1$ is such that for every choice of $q \in 2^{O(\kappa)}$ and discrete Gaussian χ over $\mathbb{Z}_q$ with standard deviation $q^{\Omega(1)}$, the assumption $\mathsf{SLWE}(k, q, \chi, \mathcal{S})$ holds with $k = \max(\log q, \kappa)^\gamma$ and a uniform information distribution $\mathcal{S}$. Then the above construction (with dimension parameter γ) yields a perfectly correct, computationally private ARE compiler as defined in Section 4.*

Complexity. We measure the output length of the encoding in terms of elements whose bit length is $\tau = O(\kappa + \log U)$. Each key consists of $k = O(\tau^\gamma)$ elements. In addition, each input or internal gate with fan-out ℓ adds an overhead of $O(k\ell\tau^\gamma)$ elements to the "garbled-circuit" part of the encoding. By reusing the public randomness of the key-shrinking gadget, this can be improved to $O(k\ell)$ plus an additional "one-time" cost of $O(kL\tau^\gamma)$ where L is the

maximal fan-out. This makes the total complexity $O(k)$ elements per wire, plus $O(kL\tau^\gamma)$ additional elements. In a typical scenario, where the integers are large enough compared to the security parameter (i.e., $\kappa = O(\log U)$) and the circuit is large enough compared to the integers (i.e., $L(\log U)^{2\gamma} = O(|C|)$), the encoding contains $O((\log U)^\gamma)$ elements of bit-length $O(\log U)$ per wire.

7.2. Constant-Rate Instances

For some natural circuit classes, we can obtain a "constant rate" encoding, namely an ARE whose output length is a constant multiple of the description size of the circuit. This type of efficiency cannot be achieved using alternative approaches (e.g., Yao's boolean construction or its ARE variant in Section 8). We demonstrate this for the case of functions whose output can be written as a constant-degree polynomial.[6] We begin with the following theorem:

Theorem 7.2. *Let $f : \mathbb{Z}^n \to \mathbb{Z}$ be a degree $d = O(1)$ polynomial defined by the sum of monomials $T_1(x) + \ldots + T_m(x)$, and let a_i be the number of monomials which depend on x_i. Define $U, \kappa, \tau = O(\kappa + \log U)$, and $k = \tau^\gamma$ as in Section 7.1. Then, under the assumption of Theorem 7.1, f has an ARE whose output consists of $(\sum a_i) + k(n + \max(a_i))$ elements of bit-length τ.*

Most polynomials of degree d have description length $\Theta(n^d \log U)$, which in a typical setting of the parameters is the same (up to a multiplicative constant) as the length of our encoding.[7] As an immediate application, let us consider the task described in the introduction, where a weak client wishes to publicly announce the value of a polynomial f applied to a sensitive input x while keeping x private. Here, a constant-rate encoding gives rise to a non-interactive solution whose offline communication complexity is just a constant multiple of the description size of f. This is essentially optimal when f is not fixed beforehand.

Proof sketch of Thm. 7.2: The proof consists of two steps. First, we show that the task of encoding f reduces to the task of encoding certain constant-depth ($\mathbf{NC^0}$) arithmetic circuits $C : \mathbb{Z}^n \to \mathbb{Z}^m$ while preserving the output complexity; then, we use the main ARE compiler to encode C with the desired complexity.

The first step relies on the locality reduction of [4, Lemma 4.17] which allows to perfectly encode f via an arithmetic $\mathbf{NC^0}$ encoding $\hat{f}(x; r)$. For any fixing of r, the function $\hat{f}_r(x)$ has the following properties: (1) Each output depends on at most d (original) inputs; (2) The input x_i affects at most a_i outputs; and (3) The output complexity is $O(m) = O(\sum a_i)$. Now, by a variant of the composition theorem from [3, Lemma 3.5], it suffices to encode this class of functions $\left\{\hat{f}_r(x)\right\}_r$ by an encoding g of output complexity $(\sum a_i) + k(n + \max(a_i))$. (Technically, the new encoding g should have a single simulator and decoder that do not depend on the choice of r; see full version for more details.)

It remains to encode the constant-depth ($\mathbf{NC^0}$) arithmetic circuit $\hat{f}_r : \mathbb{Z}^n \to \mathbb{Z}^m$. To obtain a constant-rate encoding, we combine the key-shrinking gadget with the information-theoretic encoding of Corollary 5.3. Specifically, by applying the corollary to each output bit of the circuit (which can be computed by a constant-size arithmetic formula) we get an affine encoding for $\hat{f}_r$ with key length $O(a_i)$. Applying the key-shrinking gadget we get n short keys of size k, and a garbled circuit part which consists of additional $O(\sum a_i)$ entries and a single "public" matrix of size $O(k \max a_i)$, as required. ∎

8. A CRT-Based ARE Compiler

In this section, we sketch an alternative construction of an ARE compiler over the integers, via a reduction to Yao's original garbled circuit construction. The reduction is based on the Chinese Remainder Theorem. Similarly to Yao's construction, the resulting construction can be based on any one-way function. However, it does not possess the efficiency and generality advantages of our main LWE-based construction.

Recall that Yao's construction requires to use each bit of the input for selecting one key from a pair of keys (k_0, k_1). Since we cannot directly access bits of the input in the arithmetic model, we will instead settle for *encoding* the selection. Concretely, for each input $x \in \mathbb{Z}$ and every $1 \le i \le \lceil \log U \rceil$ (where U is an upper bound on x), we will define an ARE of a function $f_i(x, k_0, k_1) = k_{x[i]}$, where $x[i]$ denotes the i-th bit in the binary representation of x. By using the Chinese Remainder Theorem (CRT), we reduce the latter task to encoding functions of the form $f_{p,i}(x, k_0, k_1) = k_{(x \bmod p)[i]}$, where p is a small prime ($p \le \mathrm{polylog}(U)$). Indeed, applying such a selector function with $O(\log U)$ distinct primes p_j for each i, we get a selection of keys corresponding to a binary CRT encoding of x. We can now feed these keys to Yao's construction applied to the following *boolean* circuit C'. The circuit C' first applies CRT decoding to the bits of the CRT encoding of each input x (whose keys are given by the outputs of $f_{p,i}$), and then computes C on the bit-representation of its inputs.

It remains to describe an ARE for the functions $f_{p,i}$. We can assume, without loss of generality, that the keys k_0, k_1 are integers in the range $[0, p-1]$. (The construction can be repeated in parallel to handle κ-bit keys.) The function

[6] The theorem generalizes to the case of multi-output functions where each output is a constant-degree polynomial. In this case, the value a_i below counts the number of *different* monomials T_j which (1) contain x_i and (2) appear as a summand in one of the outputs.

[7] E.g., when the inputs are sub-exponentially large integers in $[\pm 2^{n^\varepsilon}]$, $\kappa = n^\varepsilon$, $k = n^{\varepsilon\gamma} = o(n)$, and when $\sum a_i = O(n^d)$ and $\max(a_i) = O(n^{d-1})$; the latter two conditions hold for most polynomials.

$f_{p,i}(x, k_0, k_1)$ can now be written as

$$\sum_{0\le a<p\,:\,a[i]=0} k_0\cdot(1-(x-a)^{p-1})+\sum_{0\le a<p\,:\,a[i]=1} k_1\cdot(1-(x-a)^{p-1}),$$

with arithmetic operations modulo p. Using Lemma 5.2, the latter expression can be statistically encoded by

$$f'_{p,i}(x, k_0, k_1) = \sum_{0\le a<p\,:\,a[i]=0} k_0 \cdot (1-(x-a)^{p-1}) + \sum_{0\le a<p\,:\,a[i]=1} k_1 \cdot (1-(x-a)^{p-1}) + Rp,$$

where R is a uniformly random integer in $[0, 2^\kappa + p^p]$. Finally, since $f'_{p,i}$ can be computed by an arithmetic branching program of size $\mathrm{poly}(\log U)$ (modulo a prime of bit-length $\mathrm{poly}(\kappa, \log U)$), we can apply Fact 5.1 and again Lemma 5.2 to get a statistical ARE for $f'_{p,i}$ and hence for $f_{p,i}$.

ACKNOWLEDGMENTS

We thank Shai Halevi, Vadim Lyubashevsky and Amir Shpilka for helpful discussions.

REFERENCES

[1] B. Applebaum, "Key-dependent message security: Generic amplification and completeness," in *EUROCRYPT*, 2011.

[2] B. Applebaum, D. Cash, C. Peikert, and A. Sahai, "Fast cryptographic primitives and circular-secure encryption based on hard learning problems," in *CRYPTO*, 2009.

[3] B. Applebaum, Y. Ishai, and E. Kushilevitz, "Computationally private randomizing polynomials and their applications," *Computational Complexity*, vol. 15, no. 2, pp. 115–162, 2006.

[4] ——, "Cryptography in NC^0," *SIAM J. Comput.*, vol. 36, no. 4, pp. 845–888, 2006.

[5] ——, "Cryptography with constant input locality," *J. Cryptology*, vol. 22, no. 4, pp. 429–469, 2009.

[6] ——, "From secrecy to soundness: Efficient verification via secure computation," in *ICALP (1)*, 2010.

[7] B. Barak, I. Haitner, D. Hofheinz, and Y. Ishai, "Bounded key-dependent message security," in *EUROCRYPT*, 2010.

[8] D. Beaver, S. Micali, and P. Rogaway, "The round complexity of secure protocols (extended abstract)," in *STOC*, 1990.

[9] M. Blum and S. Micali, "How to generate cryptographically strong sequences of pseudorandom bits," *SICOMP*, vol. 13, no. 4, pp. 162–167, 1984.

[10] Z. Brakerski and V. Vaikuntanathan, "Efficient Fully Homomorphic Encryption from (Standard) LWE," in *FOCS*, 2011.

[11] C. Cachin, J. Camenisch, J. Kilian, and J. Müller, "One-round secure computation and secure autonomous mobile agents," in *ICALP*, 2000.

[12] R. Cleve, "Towards optimal simulations of formulas by bounded-width programs," in *STOC*, 1990.

[13] R. Cramer, S. Fehr, Y. Ishai, and E. Kushilevitz, "Efficient multi-party computation over rings," in *EUROCRYPT*, 2003.

[14] U. Feige, J. Kilian, and M. Naor, "A minimal model for secure computation," in *STOC*, 1994.

[15] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *CRYPTO*, 2010.

[16] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *STOC*, 2009.

[17] C. Gentry and S. Halevi, "Fully Homomorphic Encryption without Squashing Using Depth-3 Arithmetic Circuits," in *FOCS*, 2011.

[18] O. Goldreich, S. Micali, and A. Wigderson, "How to play ANY mental game," in *STOC*, 1987.

[19] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum, "Delegating computation: interactive proofs for muggles," in *STOC*, 2008.

[20] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby, "A pseudorandom generator from any one-way function," *SICOMP*, vol. 28, no. 4, pp. 1364–1396, 1999.

[21] Y. Ishai and E. Kushilevitz, "Private simultaneous messages protocols with applications," in *ISTCS*, 1997.

[22] ——, "Randomizing polynomials: A new representation with applications to round-efficient secure computation," in *FOCS*, 2000.

[23] ——, "Perfect constant-round secure computation via perfect randomizing polynomials." in *ICALP*, 2002.

[24] M. J. Jansen and B. V. R. Rao, "Simulation of arithmetical circuits by branching programs with preservation of constant width and syntactic multilinearity," in *CSR*, 2009.

[25] J. Katz, R. Ostrovsky, and A. Smith, "Round efficiency of multi-party computation with a dishonest majority," in *EUROCRYPT*, 2003.

[26] J. Kilian, "Founding cryptography on oblivious transfer," in *STOC*, 1988.

[27] Y. Lindell, "Parallel coin-tossing and constant-round secure two-party computation," in *CRYPTO*, 2001.

[28] Y. Lindell and B. Pinkas, "A proof of Yao's protocol for secure two-party computation," *J. Cryptology*, vol. 22, no. 2, pp. 161–188, 2009.

[29] V. Lyubashevsky and D. Micciancio, "On bounded distance decoding, unique shortest vectors, and the minimum distance problem," in *CRYPTO*, 2009.

[30] V. Lyubashevsky, C. Peikert, and D. Micciancio, "On Ideal Lattices and Learning with Errors Over Rings," in *EUROCRYPT*, 2010.

[31] M. Mahajan and B. V. R. Rao, "Small-space analogues of valiant's classes," in *FCT*, 2009.

[32] M. Naor, B. Pinkas, and R. Sumner, "Privacy preserving auctions and mechanism design," in *EC*, 1999.

[33] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *EUROCRYPT*, 1999.

[34] C. Peikert, "Public-key cryptosystems from the worst-case shortest vector problem," in *STOC*, 2009.

[35] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," in *STOC*, 2005.

[36] ——, "The learning with errors problem (invited survey)," in *CCC*, 2010.

[37] A. Sahai and H. Seyalioglu, "Worry-free encryption: functional encryption with public keys," in *ACM CCS*, 2010.

[38] T. Sander, A. Young, and M. Yung, "Non-interactive cryptocomputing for NC^1." in *FOCS*, 1999.

[39] L. Valiant, "Completeness classes in algebra," in *STOC*, 1979.

[40] A. C. Yao, "Protocols for secure computation," in *FOCS*, 1982.

[41] ——, "How to generate and exchange secrets," in *FOCS*, 1986.