How to Garble Arithmetic Circuits[*]

Benny Applebaum[†] Yuval Ishai[‡] Eyal Kushilevitz[§]

May 5, 2012

Abstract

Yao's garbled circuit construction transforms a boolean circuit $C : \{0,1\}^n \to \{0,1\}^m$ into a "garbled circuit" $\hat{C}$ along with n pairs of k-bit keys, one for each input bit, such that $\hat{C}$ 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 $\hat{C}$ along with n affine functions $L_i : \mathbb{Z} \to \mathbb{Z}^k$ such that $\hat{C}$ 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

[*]An extended abstract appeared in the Proceedings of the 52nd Annual Symposium on Foundations of Computer Science (FOCS), 2011.

[†]School of Electrical Engineering, Tel-Aviv University. Supported by Alon Fellowship, ISF grant 1155/11 and by the Check Point Institute for Information Security.

[‡]Department of Computer Science, Technion. Supported by ERC Starting Grant 259426, ISF grant 1361/10, and BSF grant 2008411.

[§]Department of Computer Science, Technion. Supported by ISF grant 1361/10 and BSF grant 2008411.

problems such as computing on encrypted data [38, 11], parallel cryptography [4, 3], verifiable computation [15, 5], software protection [19], functional encryption [37], and key-dependent message security [6, 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 $\mathcal{U}$ to the field in order to perform some expensive computation C on sensitive data x. The computation is too complex for $\mathcal{U}$ to quickly perform it on its own, but since the input x is sensitive $\mathcal{U}$ 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, $\mathcal{U}$ 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 $\mathcal{U}$ to select the corresponding keys K_x and send 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 $\mathcal{U}$ 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 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 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 compactly encrypt large integers using

[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.

an *additively homomorphic* encryption scheme such as Paillier'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 a *decomposable affine randomized encoding* (DARE) 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 a DARE 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. A DARE 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 DARE 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 a DARE compiler over the integers based on a general assumption (the existence of a 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 7.3). 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

[2]More precisely, the DARE 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 **VP** [39], and are believed to be strictly stronger than log-space variants that are captured by arithmetic formulas and branching programs [24, 30].

on a special "functional encryption" scheme which yields a succinct encoding 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 DARE 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 a DARE 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 $\mathbb{F}_p$). Our new framework also leads to a conceptually simpler derivation of Yao's original result for boolean circuits (see Section 2). 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 DARE for *simple* low-depth circuits. Previous randomized encoding techniques from the literature [26, 12, 23, 13] imply DARE compilers over finite rings that are weaker than our main notion of DARE 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 of 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 DARE 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 $s \Rsh (K_0, K_1) = K_s$. (Selectors and affine functions are essentially equivalent in the binary domain.) Our goal can be described as follows: We are given the function $f(y, c, d) = y \Rsh (c, d)$, where y is a bit and c, d are long keys, and we would like to encode it by a function of the form $g(y, c, d; r) = (W, y \Rsh (a, b))$. 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.[3] 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 $y \Rsh (a, b)$. 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.

It is not hard to see that g satisfies our syntactic requirements and, in addition, its output allows one to recover the value $y \Rsh (c, d)$ by decrypting the corresponding ciphertext using the key $y \Rsh (a, b)$. 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 and a, b, c, d are vectors over R. 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 $a, b \overset{R}{\leftarrow} R^k$ 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

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

(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

General. For real numbers $a < b$, let $[a, b]$ denote the set of integers i such that $a \le i \le b$. For $b > 0$, let $[b]$ denote the set $[0, b]$ and $[\pm b]$ denote the set $[-b, b]$. For $x \in \mathbb{R}$, define $\lfloor x \rceil$ as the integer closest to x or, in case two such integers exist, the smaller of the two. For an integer $p \ge 2$, we write $\mathbb{Z}_p$ for the ring $\{0, \dots, p-1\}$ with addition and multiplication modulo p. For a distribution D, let D^n denote the probability distribution over vectors of length n in which each entry is chosen independently at random from D. For a group $\mathbb{G}$, a scalar $\Delta \in \mathbb{G}$ and a distribution D over vectors in $\mathbb{G}^n$, let $D + \Delta$ denote the distribution D shifted by Δ, i.e., first choose a vector from D and then shift each of its entries by Δ.

Indistinguishability. We say that a pair of probability distributions X, Y over $\{0,1\}^n$ are (t, ε)-*indistinguishable* (denoted by $X \equiv_{t,\varepsilon} Y$) if for every circuit $\mathcal{A}$ of size at most t we have that

$$|\Pr[\mathcal{A}(X) = 1] - \Pr[\mathcal{A}(Y) = 1]| \le \varepsilon.$$

The special case where X, Y are (t, ε)-indistinguishable for *every* t (denoted by $X \equiv_\varepsilon Y$) corresponds to the case where the *statistical distance* between X and Y is ε. If, in addition, $\varepsilon = 0$ then the two distributions are identical (denoted by $X \equiv Y$). We naturally extend this notation to ensembles of distribution $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$; in this case, t and ε are viewed as functions of the security parameter n. The standard notion of *computational indistinguishability* corresponds to the case where $t = n^{\omega(1)}$ and $\varepsilon = n^{-\omega(1)}$ (denoted by $\{X_n\}_{n\in\mathbb{N}} \overset{c}{\equiv} \{Y_n\}_{n\in\mathbb{N}}$), whereas *statistical indistinguishability* corresponds to unbounded t and $\varepsilon = n^{-\omega(1)}$.

3.1 Randomized Encoding

Intuitively, a randomized encoding of a function $f(x)$ is a randomized mapping $\hat{f}(x; r)$ whose output distribution depends only on the output of f. We formalize this intuition via the notion of *computationally-private randomized encoding* (in short RE) from [3]. Unlike [3], we define here concrete security for finite functions rather than asymptotic security for infinite families of functions.

Consider a function $f : \Sigma^n \to \Sigma^l$ and a randomized function $\hat{f} : \Sigma^n \times \{0,1\}^m \to \Sigma^s$, where Σ is some finite alphabet. (We view the symbols of Σ as integers with the standard binary representation.) We say that $\hat{f}$ is a perfectly-correct (t, ε)-private encoding of f (in short, (t, ε)-encoding), if there exist a decoder algorithm Dec and a simulator Sim that satisfy the following conditions:

- **perfect correctness.** For every $x \in \Sigma^n$, $\Pr_r[\mathsf{Dec}(\hat{f}(x; r)) \neq f(x)] = 0$.
- (t, ε)**-privacy.** For every $x \in \Sigma^n$, $\hat{f}(x; r) \equiv_{t,\varepsilon} \mathsf{Sim}(f(x))$.

Our default notion of privacy is computational, but we will also consider stronger variants: an encoding is ε-private if the encoding is (t, ε)-private for any t (this corresponds to *statistically* private encoding [22]). The encoding is *perfectly* private (or just perfect) if it is 0-private. The complexity of the RE is s if the circuit size of $\hat{f}, \mathsf{Dec}$ and Sim is bounded by s.

Public randomness. We say that an RE $\hat{f}(x;r,w)$ of $f(x)$ has *public coins* w if part of the output of $\hat{f}$ consists of the coins w, i.e., if we can write $\hat{f}(x;r,w) = (w, g(x,r,w))$ for some function g. Moreover, we require a simulator Sim of a similar form; namely, given y (supposedly $y = f(x)$) and internal randomness (ρ, w), the output of $\mathsf{Sim}(y;\rho,w)$ can be written as $(w, h(y,\rho,w))$ for some function h. The notion of public coins RE has appeared in [7]. The public coins w can be chosen once and for all and be reused in different instantiations of REs as we will show in Facts 3.1–3.3 below. (This is similar to the role of public parameters in other cryptographic applications, e.g., a generator of some cyclic group.) Observe that every RE $\hat{f}(x;r)$ can be augmented with public coins w by letting g and h ignore w. We will sometimes use the notation $\hat{f}^w(x;r)$ to emphasize that w is used as public coins.

We make use of the following standard facts regarding REs (cf. [4, 3, 5]). The statements refer to REs with public coins w.

Fact 3.1 (**Substitution).** *Suppose that the function $\hat{f}^w(x;r)$ is a (t,ε)-encoding of $f(x)$ with simulator and decoder* $(\mathsf{Sim}, \mathsf{Dec})$. *Let $h(z)$ be a function of the form $f(g(z))$ where $z \in \{0,1\}^k$ and $g : \{0,1\}^k \to \{0,1\}^n$. Then, the function $\hat{h}^w(z;r) = \hat{f}^w(g(z);r)$ is a (t,ε)-encoding of h with the same simulator and the same decoder.*

Proof. Follows immediately from the definition. For correctness we have that for all z

$$\Pr_{w,r}[\mathsf{Dec}(\hat{h}^w(z;r)) \neq h(z)] = \Pr_{w,r}[\mathsf{Dec}(\hat{f}^w(g(z);r)) \neq f(g(z))] = 0,$$

and for privacy we have that for all z

$$\mathsf{Sim}^w(h(z)) \equiv \mathsf{Sim}^w(f(g(z))) \equiv_{t,\varepsilon} \hat{f}^w(g(z);r) \equiv \hat{h}^w(z;r),$$

as required. □

Fact 3.2 (**Concatenation).** *Suppose that $\hat{f}_i^w(x;r_i)$ is a (t,ε)-encoding of a function $f_i : \{0,1\}^n \to \{0,1\}^{\ell_i}$ with simulator Sim_i^w, decoder Dec_i and complexity at most s, for every $1 \leq i \leq c$. Then, the function $\hat{f}^w(x;(r_1,\dots,r_c)) = (\hat{f}_i^w(x;r_i))_{i=1}^c$ is a $(t - cs, c\varepsilon)$-encoding of $f(x) = (f_1(x),\dots,f_c(x))$ with simulator $\mathsf{Sim}^w(y) = (\mathsf{Sim}_i^w(y_i))_{i=1}^c$ and decoder $\mathsf{Dec}(\hat{y}) = (\mathsf{Dec}_i(\hat{y}_i))_{i=1}^c$.*

Proof. Perfect correctness follows from $\Pr_{r,w}[\mathsf{Dec}(\hat{f}^w(x;r)) \neq f(x)] \leq \sum_{i=1}^c \Pr_{r,w}[\mathsf{Dec}(\hat{f}_i^w(x;r_i)) \neq f_i(x)] = 0$. Privacy is proved via a standard hybrid argument. Specifically, suppose towards a contradiction, that $\mathcal{A}$ is a $(t - cs)$ size adversary that distinguishes $\hat{f}^w(x;r)$ from $\mathsf{Sim}^w(f(x);\rho)$ with advantage $c\varepsilon$. Then, by an averaging argument, for some $j \in \{1,\dots,c\}$ the adversary $\mathcal{A}$ distinguishes with advantage at least ε between the tuple

$$(\hat{f}_1^w(x;r_1),\dots,\hat{f}_{j-1}^w(x;r_{j-1}), \mathsf{Sim}_j^w(f_j(x)),\dots,\mathsf{Sim}_c^w(f_c(x)))$$

and the tuple

$$(\hat{f}_1^w(x;r_1),\dots,\hat{f}_j^w(x;r_j), \mathsf{Sim}_{j+1}^w(f_j(x)),\dots,\mathsf{Sim}_c^w(f_c(x))).$$

Now, we can define an adversary $\mathcal{B}$ that ε-distinguishes $\hat{f}_j^w(x;r_j)$ from $\mathsf{Sim}_j^w(f_j(x))$. Given a challenge $\hat{y}_j^w$, the adversary $\mathcal{B}$ samples $(\hat{f}_i^w(x;r_i))_{i<j}$ and $(\mathsf{Sim}_i^w(f_i(x)))_{i>j}$ with complexity $c \cdot s$, and invokes $\mathcal{A}$ on the resulting vector with the challenge planted in the j-th position. This gives rise to a (t,ε)-adversary, contradicting our hypothesis. □

We say that a function $g^w(x) = (y, w)$ with public randomness w is (t,ε)-encoded by $h^w(x;r)$ if privacy holds with respect to a random w, i.e., for every x the distribution $\mathsf{Sim}^w(g^w(x))$ is (t,ε)-indistinguishable from $h^w(x;r)$ where w and r are uniformly chosen. (Perfect correctness holds, as usual, for every choice of x, w and r.)

Fact 3.3 (Composition). *Suppose that:*

- $g^w(x; r_g)$ *is a* (t_1, ε_1)*-encoding of* $f(x)$ *with decoder* Dec_g *and simulator* Sim_g^w*, and*
- $h^w((x, r_g); r_h)$ *is a* (t_2, ε_2)*-encoding of the function* $g^w(x, r_g)$*, viewed as a single-argument function, with decoder* Dec_h*, simulator* Sim_h^w *and complexity* s*.*

Then, the function $\hat{f}^w(x; (r_g, r_h)) = h^w((x, r_g); r_h)$ *is a* $(\min(t_1 - s, t_2), \varepsilon_1 + \varepsilon_2)$*-encoding of* $f(x)$ *where* (r_g, r_h) *are its random inputs,* w *is the public coins, and the simulator and decoder are* $\mathsf{Sim}^w(y) = \mathsf{Sim}_h(\mathsf{Sim}_g^w(y))$ *and* $\mathsf{Dec}(\hat{y}) = \mathsf{Dec}_g(\mathsf{Dec}_h(\hat{y}))$*.*

Proof. To prove perfect correctness, note that $\Pr_{r_g, r_h, w}[\mathsf{Dec}(\hat{f}^w(x; r_g, r_h)) \neq f(x)]$ is upper-bounded by

$$\Pr_{r_g, r_h, w}[\mathsf{Dec}(h^w(x, r_g; r_h)) \neq g(x, r_g, w)] + \Pr_{r_g, w}[\mathsf{Dec}(\hat{g}^w(x; r_g)) \neq f(x)] = 0.$$

We prove privacy by noting that $\mathsf{Sim}_g^w(f(x))$ is (t_1, ε_1)-indistinguishable from $g^w(x; r_g)$. Hence, $\mathsf{Sim}_h^w(\mathsf{Sim}_g^w(f(x)))$ is $(t_1 - s, \varepsilon_1)$ indistinguishable from $\mathsf{Sim}_h^w(g^w(x; r_g))$. However, the latter distribution is (t_2, ε_2)-indistinguishable from $h^w((x, r_g); r_h)$, and so $h^w(x; (r_g, r_h))$ is $(\min(t_1 - s, t_2), \varepsilon_1 + \varepsilon_2)$-indistinguishable from $\mathsf{Sim}_g^w(f(x))$. $\square$

In Fact 3.3 (Composition), we assumed that $g(x; r_g)$ encodes f and that $h((x, r_g); r_h)$ encodes the function $g(x, r_g)$. In order to improve efficiency, one may try to replace the latter condition with the following weaker requirement: For every *fixed* r_g the function $g_{r_g}(x) = g(x; r_g)$ is encoded by some function $h_{r_g}(x; r_h)$. This variant of the composition lemma is valid provided that there exist a *universal* simulator and *universal* decoder that work for all the r_g's. In this case, we say that the collection $\{h_{r_g}(x; r_h)\}_{r_g}$ is a *universal encoding* [7] of $\{g_{r_g}\}_{r_g}$. Several known constructions of REs are universal. Let us summarize the modified version of composition.

Fact 3.4 (Composing Universal Encoding). *Suppose that:*

- $g(x; r_g)$ *is a* (t, ε)*-encoding of* $f(x)$*.*
- *For every* r_g*, the function* $h_{r_g}(x; r_h)$ *is a* (t, ε)*-encoding of the function* $g_{r_g}(x) = g(x; r_g)$*. Furthermore, all the encodings share a single (universal) decoder* Dec_h *and a single (universal) simulator* Sim_h *of complexity* s*.*

Then, the function $\hat{f}(x; (r_g, r_h)) = h_{r_g}(x; r_h)$ *is a* $(t - s, 2\varepsilon)$*-encoding of* $f(x)$*.*

The proof follows from Fact 3.3 by defining $h((x, r_g); r_h) = h_{r_g}(x; r_h)$ and noting that h is a (t, ε)-encoding of $g(x, r_g)$ with the decoder Dec_h and simulator Sim_h.

Partial functions. We will sometimes refer to randomized encodings of partial functions f which are defined over a strict subset of the input domain. In this case, we make no correctness or privacy requirements for inputs on which f is undefined. All the above properties of REs extend naturally to this case.

3.2 The Learning With Errors Problem

We recall the learning with errors (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 3.5 (**Learning with Errors**). *The* decisional learning with errors *problem* $\mathsf{LWE}(k, q, \chi, \mathcal{S})$ *is* $(t(\kappa), \varepsilon(\kappa))$ hard *if the* $\mathsf{LWE}^t(k, q, \chi, \mathcal{S})$ *distribution*

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

is (t, ε)*-indistinguishable from the uniform distribution* $(\mathcal{U}_q^{t \times k}, \mathcal{U}_q^t)$*. The* $\mathsf{LWE}(k, q, \chi, \mathcal{S})$ assumption *asserts that the above holds with* $t = \kappa^{\omega(1)}$ *and* $\varepsilon = \kappa^{-\omega(1)}$*.*

Standard choices of parameters. Typically, $\mathcal{S}$ is taken to be the uniform distribution $\mathcal{U}_q$. For the noise distribution χ, the standard choice is to use a discrete Gaussian. Recall that for any $\alpha > 0$, the density function of a one-dimensional Gaussian probability distribution over $\mathbb{R}$ is given by $\rho_\alpha(x) = \exp(-\pi(x/\alpha)^2)/\alpha$. Then, for an integer q, we define $\bar{\Psi}_{q,\alpha}$ to be the distribution over $\mathbb{Z}_q$ obtained by drawing $y \overset{R}{\leftarrow} \rho_\alpha$ and outputting $\lfloor qy \rceil \bmod q$. (We write $\bar{\Psi}_\alpha$ when q is clear from the context.) Viewing elements of $\mathbb{Z}_q$ as integers in the range $[-\lfloor q/2 \rfloor, \lfloor q/2 \rfloor]$, the random variable $\bar{\Psi}_\alpha$ has mean 0 and standard deviation roughly $\alpha q/\sqrt{2\pi}$. In typical applications of LWE, the standard deviation of the noise is q^ϵ for some $0 < \epsilon < 1$. See [35, 34, 29, 36] for a survey of LWE and related lattice problems.

Our parameters. Given a security parameter κ and an upper bound U on the range of wire values, our construction can be based on LWE with $q = (U + 2^\kappa)^{O(1)}$, $k = \log^\gamma q$ for some constant $\gamma > 1$ (see below for the choice of γ), Gaussian noise with standard deviation $q^{\Omega(1)}$, and a uniform information word $\mathcal{S}^k = \mathcal{U}_q^k$. (Hence, the modulus q is sub-exponential in the dimension, i.e., $q = 2^{(k^{1/\gamma})}$.) 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. Furthermore, by choosing q properly (e.g., a power of a small prime such as 2^ℓ) our assumption reduces to the conjectured hardness of approximating SVP to within subexponential factors [34], i.e., 2^{k^ϵ} where ϵ vanishes with γ.[4]

LWE under Rectangular Distributions. The Gaussian noise model is useful for security reductions because of its robustness under linear transformations. For instance, the sum of two independent Gaussians is a Gaussian. For our purposes, however, it will be more convenient (but not essential) to sample the noise from a "rectangular" probability distribution Φ_ρ in which an element of $\mathbb{Z}_q$ is chosen uniformly at random from the interval $[-\rho, +\rho]$, where $\rho = q^{\Omega(1)}$. Since our choice of parameters will always let q be superpolynomial in κ, this variant reduces to a similar variant with Gaussian noise of standard deviation $\alpha q = \rho^{\Omega(1)} = q^{\Omega(1)}$. Moreover, even if the information distribution $\mathcal{S}$ is sampled from the same (rectangular) distribution as the noise, this does not affect the strength of the assumption provided that q is a prime power.

Lemma 3.6. *Let* $q(\kappa) > \kappa^{\omega(1)}$ *be an integer function such that, for every* κ*, the integer* $q(\kappa)$ *is a prime power. Let* $0 < \alpha(\kappa) < 1$ *and* $0 < \rho(\kappa) < q(\kappa)$ *be integer valued functions for which* $\alpha q/\rho < \mathrm{neg}(\kappa)$*. Let* $k(\kappa) = \mathrm{poly}(\kappa)$*. If* $\mathsf{LWE}(k, q, \bar{\Psi}_\alpha, \mathcal{U}_q)$ *holds then so does* $\mathsf{LWE}(k, q, \Phi_\rho, \Phi_\rho)$*.*

The proof is deferred to Section 3.2.1. From here on, we will only use distributions of noise and information words that are uniform over intervals of length $q^{\Omega(1)}$, and a modulus q which is a power of 2.

[4]The reduction consists of two steps. First show that, for appropriate modulus (e.g., $q = 2^\ell$), the hardness of decisional-LWE with Gaussian noise $\bar{\Psi}_\alpha$ follows from the hardness of search-LWE with similar parameters and Gaussian noise $\bar{\Psi}_\beta$ where $\alpha < 1/\omega(\sqrt{\log k})$ and $\beta \geq \alpha(\omega(\sqrt{\log k}))^2$ [31, Thm. 3.1]. Then, show that the latter is as hard as approximating k-dimensional SVP within a factor of $q \cdot \alpha \cdot \mathrm{poly}(k) = 2^{O(k^{1/\gamma})}$ via the reduction of Peikert [34, Thm. 3.1].

3.2.1 Proof of Lemma 3.6

We will need the following simple facts.

Fact 3.7 (**Shifted Rectangle**). *For any integers $0 < \rho < q$ and $0 < \beta < \rho$, the statistical distance between Φ_ρ and $\Phi_\rho + \beta$ is exactly β/ρ.*

Claim 3.8. *Let q be an integer, $\rho < q$ be an integer and $0 < \alpha < \rho/q$ be a real. Then, the statistical distance between Φ_ρ and $A = \Phi_\rho + \bar{\Psi}_\alpha$ is at most $\frac{1}{t}\exp(-t^2/2) + t\alpha q/\rho$, for every t.*

Proof. We write A as a convex combination of two probability distributions: A_1 which is A conditioned on the event E that the outcome of $\bar{\Psi}_\alpha$ is not in $[-t\alpha q, t\alpha q]$, and A_2 which is A conditioned on the complementary event $\bar{\Psi}_\alpha$. The statistical distance between A and Φ_ρ is bounded by

$$\Pr[E] \cdot \|A_1 - \Phi_\rho\| + (1 - \Pr[E]) \cdot \|A_2 - \Phi_\rho\|,$$

where $\|X - Y\|$ denotes the statistical distance between X and Y. By a standard tail inequality, the probability of the event E is at most $\frac{1}{t}\exp(-t^2/2)$. On the other hand, $\|A_2 - \Phi_\rho\|$ is bounded by $t\alpha q/\rho$. To see this, note that as in Fact 3.7, this quantity bounds the statistical distance for every fixing of $\bar{\Psi}_\alpha$ conditioned on $\bar{E}$. Hence, the claim follows. □

We can now prove Lemma 3.6.

Proof of Lemma 3.6. First, we show that $\mathsf{LWE}(k, q, \bar{\Psi}_\alpha, \mathcal{U}_q)$ implies $\mathsf{LWE}(k, q, \Phi_\rho, \mathcal{U}_q)$. Indeed, given an instance (M, r) for the $\mathsf{LWE}(k, q, \bar{\Psi}_\alpha, \mathcal{U}_q)$ problem, sample a noise vector e' from Φ_ρ^n and add it to r. This mapping takes a uniform instance into a uniform instance, and an LWE instance $r = Ms + e$ where $s \overset{R}{\leftarrow} \mathcal{U}_q$ and $e \overset{R}{\leftarrow} \bar{\Psi}_\alpha$ into $r' = Ms + e + e'$. The claim now follows by Claim 3.8, as $e + e'$ is statistically indistinguishable from Φ_ρ^n.

The next step is to show that the $\mathsf{LWE}(k, q, \Phi_\rho, \mathcal{U}_q)$ assumption implies the $\mathsf{LWE}(k, q, \Phi_\rho, \Phi_\rho)$ assumption. This follows by the more general results of [2, Lemma 2], which show that when q is a prime power the error distribution can be shifted to $\mathcal{S}$. □

4 Decomposable 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] (see Section 3.1). 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 the 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 (**Decomposable affine circuits**). *We say that a randomized arithmetic circuit C is* decomposable *if each output depends on at most a single deterministic input x_i (but possibly on many random inputs). The circuit is* affine *if each of its outputs is either constant in x (i.e., depends only on the r_j) or is a degree-1 polynomial in the deterministic inputs $x = (x_1, \ldots, x_n)$ (viewing the r_j as constants). Taken together, affinity and decomposability implies that each output can be written as a degree-1 polynomial in a single input x_i. This is syntactically enforced by partitioning the output of C 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, $1 \leq i \leq n$, as* keys. *For $1 \leq i \leq n$, we denote the i-th key length $|y_i|$ by ℓ_i.*

We now define the notion of a decomposable 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 (**DARE Compiler: Syntax**). *A* decomposable affine randomized encoding (DARE) 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* decomposable affine *circuit* Enc *(encoder), randomized arithmetic circuit* Sim *(simulator), and* 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 a decomposable affine randomized encoding (or DARE) of C.*

We note that while the above definition allows the compiler T to be randomized, our main compiler will be deterministic.

We require a DARE compiler to satisfy the following correctness and privacy requirements.

Definition 4.4 (**DARE 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 $(\mathsf{Enc}, \mathsf{Dec}, \mathsf{Sim}) \overset{R}{\leftarrow} T(1^\kappa, C, U)$ *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 $(\mathsf{Enc}, \mathsf{Dec}, \mathsf{Sim}) \overset{R}{\leftarrow} T(1^\kappa, C, U)$ 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 (**DARE Compiler: Privacy**). *A DARE 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 DARE 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.

We now make a few remarks on other variants of the above definition that can be satisfied by our construction or by previous techniques.

Remark 4.6 (Applicability of previous techniques). *Previous randomized encoding techniques from the literature [26, 12, 23, 13] imply DARE compilers over finite rings that are weaker than our main notion of DARE 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 DAREs 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 DARE 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.*

Remark 4.7 (An alternative notion of privacy). *A seemingly stronger variant of privacy allows the adversary to first choose the circuit C and the bound U, and specify the input x only after seeing the output of the compiler* $(\mathsf{Enc}, \mathsf{Dec}, \mathsf{Sim})$. *Equivalently, we may require that the output of the compiler* $(\mathsf{Enc}, \mathsf{Dec}, \mathsf{Sim})$ *is always a $(\kappa^{\omega(1)}, 1/\kappa^{\omega(1)})$-encoding of C (restricted to inputs that do not violate the bound U). Our construction satisfies this variant since we get a deterministic compiler.*

Remark 4.8 (On alternative notions of arithmetic simplicity). *As in the case of boolean randomized encodings, different applications motivate different restrictions on the encoder. Here we discuss three such alternative notions of "arithmetic simplicity" that may replace our default requirement. The first two seem more liberal and the last more strict.*

1. *A more liberal notion of decomposable affine encoding allows the length of the keys y_i to grow with the circuit size. (Previous affine encodings for arithmetic formulas and branching programs suffer from this disadvantage.) Our technique will in fact show that any such decomposable affine encoding can be efficiently converted into one that satisfies our stricter definition (settling for computational privacy under the LWE assumption).*

2. *Another relaxation is to allow each output of* Enc *to be a degree-1 polynomial in* all inputs x_i *rather than just one input (i.e., require affinity but not decomposability). This notion can be shown equivalent to our more stringent notion by adapting the locality reduction technique of [4, Lemma 4.17] to computations over the integers.*

3. *Finally, we will also consider a version of decomposable affine encodings where the total degree of each output of* Enc*, as a polynomial in both the inputs* x_i *and the random inputs* r_j*, is bounded by a constant. This coincides with the notion of randomizing polynomials from [22, 13], except that here the random inputs* r_i *are picked uniformly from* $\{0,1\}$ *rather than from the entire ring, and computation is done over integers from a bounded range rather than over a finite ring. Our DARE compiler can be combined with previous results on randomizing polynomials [3] to satisfy this notion.*

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] (see Section 3.1).

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 DARE compiler is an *affinization* gadget, which provides a statistical DARE for *simple* arithmetic functions. We start with a self-contained presentation of the simplest instance of this gadget that suffices for the main instance of our construction, and then survey some extensions which follow from previous work.

Lemma 5.1 (Affinization over finite rings)**.** *Define the following functions over a finite ring* R*:*

- $f_1(x_1, x_2) = x_1 + x_2$
- $f_2(x_1, x_2, x_3) = x_1x_2 + x_3$.

Then:

1. $\hat{f}_1(x_1, x_2; r) = (x_1 + r, x_2 - r)$ *is a perfect DARE of* f_1 *over* R *(where* r *is a uniformly random ring element).*

2. $\hat{f}_2(x_1, x_2, x_3; r_1, r_2, r_3, r_4) = (a_1x_1 + b_1, a_2x_2 + b_2, a_3x_3 + b_3)$ *is a perfect DARE of* f_2 *over* R*, where* $a_1 = (1, r_2), b_1 = (-r_1, -r_1r_2 + r_3), a_2 = (1, r_1), b_2 = (-r_2, r_4), a_3 = 1, b_3 = -r_3 - r_4$*, and* r_1, r_2, r_3, r_4 *are random and independent ring elements.*

Proof. A decoder and simulator for $\hat{f}_1$ can be defined by $\mathsf{Dec}(y_1, y_2) = y_1 + y_2$ and $\mathsf{Sim}(y) = (s, y - s)$, where s is uniformly random over R. Their correctness follows from the fact that the output of $\hat{f}_1$ is uniformly random over pairs (y_1, y_2) such that $y_1 + y_2 = x_1 + x_2$.

To prove the correctness of $\hat{f}_2$, we use the following intermediate encoding of f_2 (which does not have the required affine form):

$$\hat{f}'_2(x_1, x_2, x_3\,;\, r_1, r_2) = (x_1 - r_1\,,\, x_2 - r_2\,,\, r_2x_1 - r_1r_2 + r_1x_2 + x_3).$$

It is not hard to see that $\hat{f}'_2$ is a perfect DARE for f_2: the decoder and simulator can be defined by $\mathsf{Dec}(y_1, y_2, y_3) = y_1y_2 + y_3$, and $\mathsf{Sim}(y; s_1, s_2) = (s_1, s_2, y - s_1s_2)$. Their correctness follows from the fact that the first two outputs y_1, y_2 of $\hat{f}'_2$ are uniformly random and independent, and the third output is $y_3 = f_2(x_1, x_2, x_3) - y_1y_2$. The encoding $\hat{f}_2$ is obtained from $\hat{f}'_2$ by first applying the encoding $\hat{f}_1$ (twice, using the additional randomness r_3, r_4) to break the terms in the last entry of $\hat{f}'_2$ and then arranging the outputs in a canonic affine form. □

Our main affinization gadget is a DARE of the above f_2 when viewed as a function 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 easy 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 $\varepsilon \leq 2X/(p\mu)$.*

We will sometimes need a variant of the above construction in which the output of $\hat{f}$ should be non-negative. This can be easily achieved by adding the public constant X to the output.

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

Claim 5.3 (Affinization gadget over the integers)**.** *Let $f(x_1, x_2, x_3) = x_1x_2 + x_3$, where $x_i \in [0, 2^\lambda]$. Then, f admits a statistically $2^{-\kappa}$-private DARE over the integers with non-negative keys a_i, b_i of length $\ell_i \leq 2$ and bit-length $O(\lambda) + \kappa$ (concretely, $2\lambda + \kappa + 2$ bits suffice).*

Proof. We start by viewing f as a function over $\mathbb{Z}_p$, choosing large enough p (say, $p = 2^{2\lambda+2}$) to avoid modular reduction. We then apply the DARE of Lemma 5.1, yielding a perfect DARE over $\mathbb{Z}_p$ with key length 2. Viewing this DARE as a function over the integers, its outputs have bit-length $O(\lambda)$. Applying Lemma 5.2, we get a statistical DARE over the integers with key-length 2 and bit-length $O(\lambda) + \kappa$. □

While the above gadget suffices for the basic variant of our DARE compiler, one can sometimes get better efficiency by relying on more complex affinization gadgets. These can be based on perfect DARE 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 DARE with key length $2^{O(d)}$. More generally:

Fact 5.4 (Generalized affinization gadget over finite rings)**.** *There is a perfect DARE compiler for arithmetic branching programs or arithmetic formulas over any finite ring R. For a formula or branching program of size s, the resulting key length is $O(s^2)$.*

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, 5] for further discussion and pointers. Combining Fact 5.4 with Lemma 5.2, we get a similar variant over the integers:

Corollary 5.5 (Generalized affinization gadget over the integers)**.** *There is a perfectly correct, statistically-private DARE compiler for arithmetic branching programs or arithmetic formulas over the integers. For a formula or branching program 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 value of the output, and κ is a statistical security parameter. Furthermore, the encoding is* universal *(see Section 3.1) for the class of arithmetic formulas (resp., branching program) of size at most s and output bounded by U.*

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 may 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 DARE, 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 DARE 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 $\hat{c}$ and $\hat{d}$ 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 $\hat{c}$ and $\hat{d}$ are planted in a random linear space W of a low dimension k. The space W is made public. Now every linear combination of $\hat{c}$ and $\hat{d}$ 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 $y\hat{c} + \hat{d}$. A formal description is given in Figure 1.[5]

Decoder and Simulator. In order to decode $yc+d$ from $(W, f = ya+b)$, we define the following decoder Dec: (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. Next, we define the simulator. Given $z = yc+d$ and public randomness $W_1 \overset{R}{\leftarrow} \mathcal{U}_q^{n\times k}$, the simulator Sim simulates the output (W, f) of $\hat{g}(y, c, d)$ as follows: (1) Let $\hat{z} := z\Delta + e_z$ where $e_z \overset{R}{\leftarrow} \Phi_{\mu_d}^n$; (2) Choose $f = (f_1, f_2, f_3) \overset{R}{\leftarrow} ([\rho_b]^k, [\rho_b], 1)$ and let $W = (W_1|W_2|W_3)$ where $W_2 \overset{R}{\leftarrow} \mathcal{U}_q^n$ and $W_3 := \hat{z} - (W_1 f_1 + W_2 f_2)$. The analysis of the decoder and simulator is summarized in the following lemma:

[5]The definition of $\hat{g}$ here is slightly different than the one which appears in the conference version of this paper. Both versions are correct but the current one allows to employ a non-prime modulus q.

The Encoding $\hat{g}$

Input: (y, c, d) where $y \in [\pm U], c \in [0, \rho_c]^n, d \in [0, \rho_d]^n$.
Public coins: $W_1 \overset{R}{\leftarrow} \mathcal{U}_q^{n \times k}$.

1. Choose $e_c \overset{R}{\leftarrow} \Phi^n_{\mu_c}$ and $e_d \overset{R}{\leftarrow} \Phi^n_{\mu_d}$.
2. Let $\hat{c} := c \cdot \Delta + e_c$ and $\hat{d} := d \cdot \Delta + e_d$.
3. Choose $a = (a_1, a_2, a_3) \overset{R}{\leftarrow} ([\rho_a]^k, 1, 0)$ and $b = (b_1, b_2, b_3) \overset{R}{\leftarrow} ([\rho_b]^k, [\rho_b], 1)$.
4. Let $W_2 := \hat{c} - W_1 \cdot a_1 \pmod q$ and $W_3 := \hat{d} - (W_1 \cdot b_1 + W_2 \cdot b_2) \pmod q$.

Output: $(W = (W_1|W_2|W_3), ya + b)$.

Figure 1: The Key-Shrinking Gadget. The construction is parameterized with positive integers $k, \rho_a, \rho_b, \mu_c, \mu_d, \Delta$ and modulus q (that does not have to be a prime number). 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 $\hat{c}$ and $\hat{d}$. 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.) Recall that Φ_α denotes the uniform distribution over the integers in the interval $[\pm\alpha]$.

Lemma 6.1. *Suppose that* $\mathsf{LWE}(k, q, \Phi_{\mu_c}, \Phi_{\rho_a/2})$ *holds and that*

$$k \leq \text{poly}(\kappa), \qquad n \leq \text{poly}(k), \qquad U\rho_a/\rho_b \leq \text{neg}(\kappa)$$
$$U\mu_c/\mu_d \leq \text{neg}(\kappa), \qquad 3\mu_d < \Delta, \qquad 3\Delta(U\rho_c + \rho_d) < q.$$

Then, for every y, c and d:

$$\Pr[\mathsf{Dec}(\hat{g}(y, c, d)) = yc + d] = 1, \text{ and } \mathsf{Sim}(yc + d) \overset{c}{\equiv} \hat{g}(y, c, d).$$

Proof. To see that the decoder is perfectly correct, note that: (a) By linear algebra, the outcome of step (1) is equal to $y\hat{c} + \hat{d}$ modulo q; (b) The above equality holds over the integers as well; Indeed, the lower bound on q ensures that the absolute value of each entry in $y\hat{c} + \hat{d}$ (computed over the integers) is smaller than $(q-1)/2$ and therefore there is no wraparound. (c) Δ is large enough to ensure that there is no rounding error in step (2). To see this, note that

$$y\hat{c} + \hat{d} = y(c\Delta + e_c) + (d\Delta + e_d) = \Delta(yc + d) + (ye_c + e_d),$$

where the absolute value of each entry in $ye_c + e_d$ is at most $U\mu_c + \mu_d < 1.5\mu_d < \Delta/2$.

In Section 6.2 we use a hybrid argument to show that, under LWE, 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 $\hat{c}$ and $\hat{d}$ in span(W), we plant the vectors $\hat{c}$ and $\hat{z}$ (as defined above). Then, we show that, under LWE, the vector $\hat{c}$ is hidden in span(W) and W is indistinguishable from a random matrix which spans $\hat{z}$. □

Setting the parameters. Fix $\rho > 1$ corresponding to the strength of the LWE assumption. Suppose that $\rho_a = 2^{\kappa+1}$, $\rho_b = U \cdot 2^{2\kappa}$, $\mu_c = 2^\kappa$, $\mu_d = U \cdot 2^{2\kappa}$, $\Delta = U \cdot 2^{2\kappa+2}$, q is a power of two where $q = \Theta(2^{2\kappa} U(U\rho_c + \rho_d))$, $k = \log^{\gamma} q$, 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 LWE assumption. (Namely, $\mathsf{LWE}(k, q, \Phi_{2^\kappa}, \Phi_{2^\kappa})$.) 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 $\tau = \log(q) = O(\kappa + \log U + \log\rho_c + \log\rho_d)$. We partition the output of the gadget to three parts: (1) Elements that depend on the "key selector" y, namely the vector $y \cdot a(r) + b(r)$ which consists of $k = \tau^\gamma$ elements each of bit length smaller than τ. (2) The submatrix W_1 which consists of $n(k-2) = O(n\tau^\gamma)$ elements of bit length τ; and (3) the submatrix $W_2|W_3$ which consists of $O(n)$ elements of bit length τ. We distinguish the submatrix W_1 from the other entries, since this part of the encoding consists only of public coins, and can be reused among different instantiations of the gadget. (See Section 3.1 for more details.) This will allow us to improve the efficiency of our DARE as our general construction employs many copies of the key-shrinking gadget.

6.2 Computational Privacy

We establish the correctness of the simulator Sim defined in Section 6.1. Namely, we show that, under the assumptions of Lemma 6.1, the distribution $\mathsf{Sim}(cy + d; W_1)$ is computationally indistinguishable from $\hat{g}(y, c, d; W_1)$ for every fixing of y, c, d. The proof follows from a hybrid argument as shown in Claims 6.2–6.5. The simulator and the hybrids are described in Table 1.

Step	$D_1(y,c,d;W_1)$	$D_2(y,c,d;W_1)$	$D_3(y,c,d;W_1)$	$\mathsf{Sim}(yc+d;W_1)$
1	$e_c \overset{R}{\leftarrow} \Phi^n_{\mu_c}$ $e_d \overset{R}{\leftarrow} \Phi^n_{\mu_d}$	$e_z \overset{R}{\leftarrow} \Phi^n_{\mu_d}$		
2	$\hat{c} := c\Delta + e_c$ $\hat{d} := d\Delta + e_d$ $\hat{z} := y\hat{c} + \hat{d}$	$\hat{z} := (cy + d)\Delta + e_z$		
3	$a \overset{R}{\leftarrow} ([\rho_a]^k, 1, 0)$ $b \overset{R}{\leftarrow} ([\rho_b]^k, [\rho_b], 1)$ $f := ya + b$		$f \overset{R}{\leftarrow} ([\rho_b]^k, [\rho_b], 1)$	
4	$W_2 := \hat{c} - W_1 a_1$ $W_3 := \hat{z} - (W_1 f_1 + W_2 f_2)$			$W_2 \overset{R}{\leftarrow} \mathcal{U}^n_q$
Output:	$(W = (W_1 \| W_2 \| W_3), f)$			
Remark:	Linear algebra	Linearity+Fact 3.7	Fact 3.7	LWE

Table 1: The hybrids. All hybrids take W_1 as public randomness. To avoid clutter only the modifications appear, and blanks indicate that no change has occurred wrt the previous hybrid. (For example, the vector e_c in D_2 is defined exactly as in D_1.) In Step 4, arithmetic is performed modulo q.

Claim 6.2. *$\hat{g}(y, c, d; W_1) \equiv D_1(y, c, d; W_1)$, for every fixed y, c, d, W_1.*

Proof. Fix y, c, d, W_1 and all the random coins in both experiments. The two experiments are defined similarly, except that in D_1 the value of W_3 is defined to be $\hat{z} - (W_1 \cdot f_1 + W_2 \cdot f_2)$ where

$f_1 = ya_1 + b_1$ and $f_2 = y + b_2$. (All arithmetic is performed modulo q.) By linear algebra, this is equal to $\hat{d} - (W_1W_1 + W_2b_2)$ which is the value given to W_2 in $\hat{g}$. Indeed, $\hat{z} - (W_1 \cdot f_1 + W_2 \cdot f_2)$ can be written as

$$(y\hat{c}+\hat{d})-W_1(ya_1+b_1)-W_2(y+b_2) = y(\hat{c}-(W_1a_1+W_2))+\hat{d}-(W_1b_1+W_2b_2) = \hat{d}-(W_1b_1+W_2b_2),$$

as required. □

Claim 6.3. *$D_1(y, c, d; W_1) \equiv_{n \cdot U\mu_c/\mu_d} D_2(y, c, d; W_1)$ for every fixed y, c, d, W_1.*

Recall that Lemma 6.1 guarantees that $n \cdot U\mu_c/\mu_d = \text{neg}(\kappa)$.

Proof. Fix y, c, d, W_1 and all the random coins in both experiments except e_d and e_z. Let us compare the distribution of the vector $\hat{z}$ in both experiments. In D_1, we have

$$\hat{z} = y\hat{c} + \hat{d} = (cy + d)\Delta + (e_d + ye_c),$$

while in D_2 we changed $\hat{z}$ to $(cy + d)\Delta + e_z$. Since c, y, d and e_c are fixed, it suffices to show that e_z is statistically indistinguishable from $(e_d + ye_c)$. Indeed, by noting that ye_c is a fixed vector whose entries are in $[\pm U\mu_c]$, we can apply Fact 3.7 and show that the statistical distance is at most $n \cdot U\mu_c/\mu_d$. □

Claim 6.4. *$D_2(y, c, d; W_1) \equiv_{kU\rho_a/\rho_b} D_3(y, c, d; W_1)$, for every fixed y, c, d, W_1.*

Recall that Lemma 6.1 guarantees that $kU\rho_a/\rho_b = \text{neg}(\kappa)$ (this follows from the first three items).

Proof. Fix y, c, d, W_1 and all the random coins in both experiments except b and f. We compare the distribution of the vector f in both experiments. In D_3, the vector $f \xleftarrow{R} ([\rho_b]^k, [\rho_b], 1)$, while in D_2 the f is taken to be $ya+b$, where $b \xleftarrow{R} ([\rho_b]^k, [\rho_b], 1)$ and ya is a fixed vector in $([0, U\rho_a]^k, 1, 0)$. Hence, by Fact 3.7, the statistical distance between the two distributions is at most $kU\rho_a/(0.5\rho_b)$. □

Finally, we derive the following claim:

Claim 6.5. *If $\mathsf{LWE}(k, q, \Phi_{\mu_c}, \Phi_{\rho_a/2})$ is (t, ε)-hard then $D_3(y, c, d; W_1) \equiv_{t-O(kn \lg^2 q),\varepsilon} \mathsf{Sim}(yc+d; W_1)$, for every fixed y, c, d.*

Proof. First observe that the coins e_d and b have no affect on the output in both distribution and therefore they can be ignored. Fix y, c, d and all the random coins in both experiments except W_1, a_1, e_c and W_2. Under this fixing, we can define a function $\sigma_{y,c,d,,e_z,f}(W_1, W_2)$ computable by a circuit of size $O(nk \log q)$ such that the outcome of Sim is $\sigma(\mathcal{U}_q^{n\times k}, \mathcal{U}_q^n)$ and the outcome of D_3 is $\sigma(W_1, W_2)$ where $W_1 \xleftarrow{R} \mathcal{U}_q^{n\times k}$ and

$$W_2 = \hat{c} - W_1a_1 = W_1 \cdot (-a_1) + e_c + c\Delta, \text{ where } a_1 \xleftarrow{R} [\rho_a]^k, e_c \xleftarrow{R} \Phi_{\mu_c}^n.$$

(All arithmetic is performed modulo q.) Hence, it suffices to show that the distribution

$$(W_1, W_1 \cdot (-a_1) + e_c + c\Delta) \tag{1}$$

is $(t - O(kn \log^2 q), \varepsilon)$-indistinguishable from the uniform distribution. This distribution is quite similar to the LWE distribution (which is assumed to be pseudorandom), except that a_1 is uniform over $[\rho_a]^k$ and not over $\Phi_{\rho_a/2}^k$. We now show that the two variants are actually equivalent.

We define an efficient mapping $\alpha : \mathbb{Z}_q^{n\times k} \times \mathbb{Z}_q^n \to \mathbb{Z}_q^{n\times k} \times \mathbb{Z}_q^n$ which maps (M, v) to $(M, v + c\Delta - M \cdot 1_{\rho_a/2})$ where $1_{\rho_a/2}$ denotes the length-k vector whose entries are all equal to $\lfloor \rho_a/2 \rfloor$. (Recall that ρ_a is assumed to be even.) Observe that α takes the uniform distribution to itself. In addition the LWE distribution $M \xleftarrow{R} \mathcal{U}_q^{n\times(k)}, v = Ms + e$ where $s \xleftarrow{R} \Phi^k_{\rho_a/2}$ and $e \xleftarrow{R} \Phi^n_{\mu_c}$ is mapped by α to the distribution $(M \xleftarrow{R} \mathcal{U}_q^{n\times(k)}, v')$ where

$$v' = Ms + e + c\Delta - M \cdot 1_{\rho_a/2} = M(s - 1_{\rho_a/2}) + e + c\Delta = Ms' + e + c\Delta,$$

where $s' \xleftarrow{R} [\rho_a/2]^k$ and $e \xleftarrow{R} \Phi^n_{\mu_c}$. The latter distribution is exactly the distribution of Eq. 1. Since α is computable by a circuit of size $O(nk \lg^2 q)$, it follows that $(t - O(nk \lg^2 q), \varepsilon)$ distinguishing (1) from the uniform distribution allows to (t, ε) distinguish $\mathsf{LWE}(k, q, \Phi_{\mu_c}, \Phi_{\rho_a/2})$ from the uniform distribution, contradicting our hypothesis. $\square$

It follows from Claims 6.2–6.5, that, under the conditions of Lemma 6.1, the distribution $\mathsf{Sim}(cy + d; W_1)$ is computationally indistinguishable from $\hat{g}(y, c, d; W_1)$ for every fixing of y, c, d.

7 Garbling Arithmetic Circuits

In this section we combine the affinization gadget and the key-shrinking gadget for obtaining our main result: an efficient DARE 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 7.3.

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.[6] We 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 DARE 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 a DARE for $C^{i-1}(y^{i-1})$ into a DARE 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 a decomposable affine form (at the cost of increasing the size of the keys), and finally applying

[6] 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 5.5 (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 7.3 below.

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 a DARE of $C^h(y^h) = C(x)$.

More precisely, the encoder Enc produced by the DARE 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 a decomposable affine encoder G of the same function by applying to each output that depends on two inputs y_j^i the basic affinization gadget (Claim 5.3). For instance, a term Q as above can either be handled directly via the generalized gadget of Corollary 5.5, or by applying the basic gadget of Claim 5.3 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 a decomposable 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 decomposable affine encoding resulting from this step.

3. Output the arithmetic circuit computing $\mathsf{Enc}(x) = \mathsf{Enc}^h(x)$.

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 Section 3.1). 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 $q(\kappa) \in 2^{O(\kappa)}$ and discrete Gaussian χ over $\mathbb{Z}_q$ with standard deviation $q^{\Omega(1)}$, the assumption $\mathsf{LWE}(k, q, \chi, \mathcal{S})$ 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 DARE 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 Proof of Theorem 7.1

Fix κ, U and C, let b_i be the number of entries in y^i, and let c_i be the size of the circuit C^i. We assume that the affinization gadget has (statistical) $\varepsilon_{\mathsf{aff}}$-privacy and complexity s_{aff} and that the key-shrinking gadget is a $(t_{\mathsf{shr}}, \varepsilon_{\mathsf{shr}})$-encoding with complexity s_{shr}. The proof of Theorem 7.1 will follow from the following lemma.

Lemma 7.2. *For every $1 \le i \le h$, the function $\mathsf{Enc}^i(y^i)$ is a (t_i, ε_i)-encoding of $C^i(y^i)$ with simulator Sim^i and decoder Dec^i of size s_i, where $t_i \ge t_{\mathsf{shr}} - c_i(s_{\mathsf{aff}} + s_{\mathsf{shr}}), \varepsilon_i \le c_i(\varepsilon_{\mathsf{aff}} + \varepsilon_{\mathsf{shr}})$ and $s_i \le c_i(s_{\mathsf{aff}} + s_{\mathsf{shr}})$.*

With our choice of parameters we have $\varepsilon_{\mathsf{aff}} = 2^{-\kappa}$, $s_{\mathsf{aff}} = \mathrm{poly}(\kappa)$, $s_{\mathsf{shr}} = \mathrm{poly}(\kappa, \log U)$, and $(t_{\mathsf{shr}}, \varepsilon_{\mathsf{shr}}) = (\kappa^{\omega(1)}, 1/\kappa^{\omega(1)})$ assuming that LWE is hard.

Proof. It will be convenient to prove a stronger claim, namely that $t_i = t_{\mathsf{shr}} - (c_i(s_{\mathsf{aff}} + s_{\mathsf{shr}}) - b_i s_{\mathsf{aff}}), \varepsilon_i = c_i(\varepsilon_{\mathsf{aff}} + \varepsilon_{\mathsf{shr}}) - b_i\varepsilon_{\mathsf{aff}}$ and $s_i = c_i(s_{\mathsf{aff}} + s_{\mathsf{shr}}) - b_i s_{\mathsf{aff}}$. The proof is by induction on i. For $i = 0$, the identity function $C^0(y^0)$ is perfectly encoded by the identity function $\mathsf{Enc}^0(y^0)$ and so the claim is trivially correct by letting Sim^0 and Dec^0 be the identity functions over $[\pm U]^{b_0}$.

Suppose that the function $\mathsf{Enc}^i(y^{i-1})$ is a $(t_{i-1}, \varepsilon_{i-1})$-encoding of $C^{i-1}(y^{i-1})$ with simulator Sim^{i-1} and decoder Dec^{i-1} of circuit size s_{i-1}. By Fact 3.1 (substitution), we have

Claim 7.3. *The function F defined in the i-th iteration is a $(t_{i-1}, \varepsilon_{i-1})$-encoding of $C^i(y^i)$ with the simulator Sim^{i-1} and decoder Dec^{i-1}.*

Recall that the function G of the i-th iteration is defined by applying an $\varepsilon_{\mathsf{aff}}$-encoding to each of the b_{i-1} outputs of F. (The complexity of each of the encodings and the simulator and decoder is s_{aff}.) Hence, by Fact 3.2 (concatenation), the function G is a $b_{i-1} \cdot \varepsilon_{\mathsf{aff}}$-encoding of F whose complexity (resp., simulator and decoder complexity) is $b_{i-1} \cdot s_{\mathsf{aff}}$. By Fact 3.3 (composition) we get that:

Claim 7.4. *The function G is $(t_{i-1} - b_{i-1} \cdot s_{\mathsf{aff}}, \varepsilon_{i-1} + b_{i-1} \cdot \varepsilon_{\mathsf{aff}})$-encoding of $C^i(y^i)$ with simulator and decoder of complexity at most $s_{i-1} + b_{i-1} \cdot s_{\mathsf{aff}}$.*

Recall that Enc^i is defined by viewing G as the concatenation of b_i functions and by encoding each of these functions by a $(t_{\mathsf{shr}}, \varepsilon_{\mathsf{shr}})$-encoding whose complexity for encoding, simulation and decoding is s_{shr}. Hence, by Fact 3.2 (concatenation), Enc^i is a $(t_{\mathsf{shr}} - b_{i-1}s_{\mathsf{shr}}, b_i\varepsilon_{\mathsf{shr}})$-encoding of complexity $b_i \cdot s_{\mathsf{shr}}$. By Fact 3.3 (composition) we get that:

Claim 7.5. *The function* Enc^i *is* $(t_{i-1} - b_{i-1} \cdot s_{\mathsf{aff}} - b_i \cdot s_{\mathsf{shr}}, \varepsilon_{i-1} + b_{i-1} \cdot \varepsilon_{\mathsf{aff}} + b_i \cdot \varepsilon_{\mathsf{shr}})$*-encoding of* $C^i(y^i)$ *with simulator and decoder of complexity at most* $s_{i-1} + b_{i-1} \cdot s_{\mathsf{aff}} + b_i \cdot s_{\mathsf{shr}}$*.*

By noting that the above quantities equal to t_i, ε_i and s_i (as follows from the equality $b_0 + \ldots + b_i = c_i$), we complete the induction. □

Uniformity of the simulator and decoder. The circuits for the simulator Sim^i and decoder Dec^i are *uniform* in the following sense: There exists a polynomial-time algorithm M that given the circuit C^i, the security parameter 1^κ, and the upper-bound U (in binary representation) outputs the circuits Sim^i and Dec^i. This follows from the proof of Lemma 7.2 by noting that: (1) the above claim is trivially true for $i = 0$; and (2) there is a polynomial-time algorithm T which takes as an input the parameters $(1^\kappa, U)$, together with B_i (the i-th layer of C^i) and the circuits $(\mathsf{Sim}^{i-1}, \mathsf{Dec}^{i-1})$, and outputs the circuits $(\mathsf{Sim}^i, \mathsf{Dec}^i)$. The description of T is implicit in Facts 3.1–3.3 and in the definitions of the affinization and shrinking gadgets.

7.3 Constant-Rate Instances

For some natural circuit classes, we can obtain a "constant rate" encoding, namely a DARE whose output length is a constant multiple of the description size of the circuit. This type of efficiency cannot be achieved using alternative approaches we are aware of (including Yao's boolean construction or its DARE variant in Section 8). We demonstrate this for the case of functions whose output can be written as a constant-degree polynomial in the inputs.

Theorem 7.6. *Let* $f : \mathbb{Z}^n \to \mathbb{Z}$ *be a polynomial of constant degree* d *defined by the sum of monomials* $T_1(x) + \ldots + T_m(x)$*. Let* a_i *be the number of monomials which depend on* x_i*. Let* $\tau = \kappa + \log U$ *and* $k = \tau^\gamma$ *as in Section 7.1, where* κ *is the security parameter and* U *upper-bounds the value of the inputs, output and the intermediate values* $T_i(x)$*. Then, under the assumption of Theorem 7.1,* f *has a DARE whose output consists of* $k(n + \max(a_i)) + \sum a_i$ *elements of bit-length* $O(\tau)$*.*

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 of Theorem 7.6. 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 DARE compiler to encode C with the desired complexity.

The first step relies on the locality reduction of [4, Appendix B] which allows to perfectly encode f via the arithmetic encoding $\hat{f}(x; r) = (T_1(x) - r_1, \ldots, T_m(x) - r_m, \sum r_i)$, where addition and subtraction are performed modulo $2U+1$. (Recall that $T_i(x)$ and $\sum T_i(x)$ are all in the interval $[\pm U]$, hence there is no wraparound.) Observe that when r is fixed $\hat{f}_r(x)$ is an $\mathbf{NC^0}$ circuit in which each output depends on at most d inputs and each input x_i affects exactly a_i outputs. Furthermore, all the different functions $\left\{\hat{f}_r\right\}$ have the same input-output dependency graph (i.e., for $i \in [m]$ the

[7]For instance, this is the case when the inputs are 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.

i-th output depends on the inputs that participate in T_i, and the last output is fixed). Now, by Fact 3.4, it suffices to encode this class of functions $\left\{\hat{f}_r(x)\right\}$ by a universal encoding $\{g_r(x;r')\}$ of output complexity $k(n+\max(a_i))+\sum a_i$.

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 5.5. Specifically, by applying the corollary to each output of the circuit (which can be computed by a constant-size arithmetic formula) and concatenating the results (Fact 3.2) we get a decomposable affine encoding for $\hat{f}_r$ with key length $O(a_i)$. Applying the key-shrinking gadget and employing standard composition (Fact 3.3) we get a decomposable affine encoding with 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. To see that the encoding is universal for $\left\{\hat{f}_r(x)\right\}$ observe that the simulator and decoder depend only on the dependency graph of $\hat{f}_r$ (which is universal) and on the simulator and decoder of the information theoretic encoding for size s arithmetic formulas (Corollary 5.5) which are also universal. $\square$

We note that Theorem 7.6 can be generalized to the case of multi-output functions where each output is a constant-degree polynomial. In this case, the value a_i counts the number of *different* monomials T_j which (1) contain x_i and (2) appear as a summand in one of the outputs.

8 A CRT-Based DARE Compiler

In this section, we sketch an alternative construction of a DARE 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 a DARE 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 a DARE 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 $f_{p,i}$ can now be written as

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

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 $\text{poly}(\log U)$ (modulo a prime of bit-length $\text{poly}(\kappa, \log U)$), we can apply Fact 5.4 and again Lemma 5.2 to get a statistical DARE 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 theorems. 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*, pages 595–618, 2009.

[3] B. Applebaum, Y. Ishai, and E. Kushilevitz. Computationally private randomizing polynomials and their applications. *Computational Complexity*, 15(2):115–162, 2006.

[4] B. Applebaum, Y. Ishai, and E. Kushilevitz. Cryptography in NC^0. *SIAM J. Comput.*, 36(4):845–888, 2006.

[5] B. Applebaum, Y. Ishai, and E. Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *ICALP (1)*, pages 152–163, 2010.

[6] B. Barak, I. Haitner, D. Hofheinz, and Y. Ishai. Bounded key-dependent message security. In *EUROCRYPT*, pages 423–444, 2010.

[7] O. Barkol and Y. Ishai. Secure computation of constant-depth circuits with applications to database search problems. In *CRYPTO*, pages 395–411, 2005.

[8] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols (extended abstract). In *STOC*, pages 503–513, 1990.

[9] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudorandom bits. *SICOMP*, 13(4):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*, pages 512–523, 2000.

[12] R. Cleve. Towards optimal simulations of formulas by bounded-width programs. In *STOC*, pages 271–277, 1990.

[13] R. Cramer, S. Fehr, Y. Ishai, and E. Kushilevitz. Efficient multi-party computation over rings. In *EUROCRYPT*, pages 596–613, 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*, pages 169–178, 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*, pages 218–229, 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. *SIAM J. Comput.*, 28(4):1364–1396, 1999. Preliminary versions appeared in STOC' 89 and STOC' 90.

[21] Y. Ishai and E. Kushilevitz. Private simultaneous messages protocols with applications. In *ISTCS*, pages 174–184, 1997.

[22] Y. Ishai and E. Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *FOCS*, pages 294–304, 2000.

[23] Y. Ishai and E. Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *ICALP*, pages 244–256, 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*, pages 179–190, 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*, pages 20–31, 1988.

[27] Y. Lindell. Parallel coin-tossing and constant-round secure two-party computation. In *CRYPTO*, pages 171–189, 2001.

[28] Y. Lindell and B. Pinkas. A proof of Yao's protocol for secure two-party computation. *J. Cryptology*, 22(2):161–188, 2009.

[29] V. Lyubashevsky and D. Micciancio. On bounded distance decoding, unique shortest vectors, and the minimum distance problem. In *CRYPTO*, pages 577–594, 2009.

[30] M. Mahajan and B. V. R. Rao. Small-space analogues of valiant's classes. In *FCT*, pages 250–261, 2009.

[31] D. Micciancio and C. Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT*, pages 700–718, 2012.

[32] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *Electronic Commerce*, pages 129–139, 1999.

[33] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[34] C. Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In *STOC*, pages 333–342, 2009.

[35] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93, 2005. Full version in JACM 56(6), article 34, 2009.

[36] O. Regev. The learning with errors problem (invited survey). In *IEEE Conference on Computational Complexity*, pages 191–204, 2010.

[37] A. Sahai and H. Seyalioglu. Worry-free encryption: functional encryption with public keys. In *ACM Conference on Computer and Communications Security*, pages 463–472, 2010.

[38] T. Sander, A. Young, and M. Yung. Non-interactive cryptocomputing for NC^1. In *FOCS*, pages 554–567, 1999.

[39] L. Valiant. Completeness classes in algebra. In *STOC*, pages 249–261, 1979.

[40] A. C. Yao. Protocols for secure computation. In *FOCS*, pages 160–164, 1982.

[41] A. C. Yao. How to generate and exchange secrets. In *FOCS*, pages 162–167, 1986.