Randomization Techniques for Secure Computation

Yuval Ishai [1]
Computer Science Department, Technion
e-mail: yuvali@cs.technion.ac.il

Abstract. To what extent can a computation be made "simpler" by settling for computing a randomized encoding of the output? Originating from Yao's seminal idea of garbled circuits, answers to this question have found applications in cryptography and elsewhere. We will survey the state of the art on different flavors of this question that are motivated by different problems in secure computation and correspond to different notions of simplicity.

1. Introduction

Suppose that a weak client holds a sensitive input x and wishes to reveal $f(x)$ to a strong server by sending a single message to the server. The client cannot just send x since this reveals more than just $f(x)$. On the other hand, the client is too weak to compute f on its own. Can the client's job be made easier by settling for computing some *encoding* $\hat{y}$ of $y = f(x)$?

Using a traditional (deterministic) encoding is not very helpful in this context. For instance, if f is a boolean function, then the encodings $\hat{y}_0, \hat{y}_1$ of the two possible output values must differ in some position i. In this case, computing the i-th bit of the encoded output is as difficult as computing f. However, by allowing $\hat{y}$ to be a *randomized* encoding of y, this relaxed notion of computation turns out to be surprisingly powerful. More concretely, we would like to replace $f(x)$ by a "simpler" function $\hat{f}(x, r)$ such that a sample $\hat{y}$ from the output of $\hat{f}$, induced by a uniform choice of r, conveys the same information about x as $f(x)$. The latter can be broken into a *correctness* requirement, asserting that the output $f(x)$ can be decoded from $\hat{f}(x, r)$ (without knowing r), and a *privacy* requirement, asserting that the output distribution of $\hat{f}(x, r)$ induced by a random choice of r can be simulated given $f(x)$ alone (without knowing x). We will also consider natural relaxations of the privacy requirement to statistical or computational indistinguishability, and will typically require that a representation of $\hat{f}$ and its associated decoder and simulator (say, using boolean circuits) can be efficiently generated from a representation of f. See Section 3 for formal definitions. We refer to the function $\hat{f}$ as a *randomized encoding* of f, or simply as an encoding of f.

[1] Supported by the European Union's Seventh Framework Programme under grant agreement no. 259426 - ERC - Cryptography and Complexity.

Note that correctness alone and privacy alone can be satisfied by trivial functions $\hat{f}$ (e.g., $\hat{f}(x,r) = x$ and $\hat{f}(x,r) = 0$, respectively). However, the combination of the two requirements is a non-trivial relaxation of the usual notion of computing.

As an example, let $x \in \mathbb{F}_p$ where p is a large prime, and consider the function $f(x)$ which outputs 1 if x is a quadratic residue in the field $\mathbb{F}_p$ (namely, there exists $y \in \mathbb{F}_p$ such that $y^2 = x$) and outputs -1 otherwise. Computing $f(x)$ can be done by raising x to the power of $(p-1)/2$. A computationally simpler randomized encoding of f is $\hat{f}(x,r) = r^2 x$, where r is uniformly distributed over all nonzero elements of $\mathbb{F}$. Correctness follows from the fact that $f(x) = f(r^2 x)$ for all r (hence $f(x)$ can be decoded from $\hat{f}(x,r)$ by simply applying f), and privacy from the fact that $\hat{f}(x,r)$ is uniformly distributed over all inputs x' such that $f(x') = f(x)$.

Going back to the motivating question, if f admits an encoding $\hat{f}$ which is simple enough for the client to compute, the client can communicate $f(x)$ to the server by picking a random r and sending $\hat{y} = \hat{f}(x,r)$. By the correctness and privacy properties of $\hat{f}$, the server can recover $y = f(x)$ from $\hat{y}$ but cannot obtain additional information about x.

Randomized encodings of functions have found many other applications in cryptography and elsewhere. Different applications motivate different notions of simplicity which the encoding $\hat{f}$ should obey. In Section 2 we informally survey some of the applications in the area of secure computation, the notions of simplicity which they motivate, and known results about randomized encodings which satisfy these notions of simplicity. Following formal definitions and some preliminaries (Section 3), constructions of randomized encodings are presented in Section 4. We conclude in Section 5 by discussing some open questions.

2. Overview of Applications and Known Results

In this section we survey different types of randomized encodings and their applications, focusing mainly on the area of secure computation. We refer the reader to [Can00,Can01, Gol04] for a general background on secure computation and to [App11b,BHR12b] for a broader overview of applications of randomized encodings in other areas of cryptography.

2.1. Decomposable Encodings

The first application of randomized encodings in cryptography dates back to Yao's seminal idea of using *garbled circuits* for secure two-party computation, typically attributed to [Yao86]. The randomized encoding $\hat{f}$ implied by Yao's construction has the following property: every output bit of $\hat{f}$ depends on at most a single bit of input x_i (but may arbitrarily depend on the randomness r). Hence, the output of $\hat{f}$ can be decomposed into $\hat{f}(x,r) = (\hat{f}_0(r), \hat{f}_1(x_1,r), \ldots, \hat{f}_n(x_n,r))$. Equivalently, the randomness r determines a string $\hat{y}_0$ (referred to as a "garbled circuit") and n pairs of strings $(\hat{y}_{i,0}, \hat{y}_{i,1})$ (referred to as "keys") such that the output of $\hat{f}$ consists of $\hat{y}_0$ along with the n strings $\hat{y}_{i,x_i}$ selected by the bits of x. We refer to an encoding $\hat{f}$ which satisfies this property as a *decomposable* randomized encoding.

Yao's construction of decomposable encodings is polynomial in the circuit size of f, but it is only computationally private. (When referring to asymptotic efficiency of randomized encodings, we implicitly assume there is a compiler which converts a representation of f into a corresponding representation of $\hat{f}$.) For weaker representation models, such as formulas and branching programs, there are efficient general constructions of decomposable encodings which satisfy the perfect notion of privacy [Kil88,FKN94,IK97,IK02,Kol05]. Such constructions will be described in Section 4.

To summarize the state of the art on efficient constructions, Yao's construction implies that every polynomial-time computable function admits a polynomial-size (and polynomial-time computable) decomposable encoding with *computational privacy*, assuming the existence of a one-way function. Moreover, every function in the complexity class NC^1 (capturing logarithmic-depth circuits with bounded fan-in) or in various logarithmic space classes (such as deterministic, non-deterministic, and counting log-space) admits a polynomial-size decomposable encoding with perfect correctness and privacy. There are also some natural functions which are conjectured not to be in these classes but still admit a polynomial-size, perfectly private decomposable encoding. These include the quadratic residuosity function and other functions which have certain random self-reducibility properties [FKN94,DGRV11,IKP12].

In the following we describe applications of decomposable encodings in the area of secure computation.

Secure two-party computation in the OT-hybrid model. The decomposition property of $\hat{f}$ makes it useful for secure two-party computation of f via parallel invocations of 1-out-of-2 oblivious transfer [Rab81,EGL85]. It is convenient to describe the protocol in the so-called *OT-hybrid model*, where it is assumed that there is a trusted oblivious transfer (OT) oracle which receives a pair of messages (m_0, m_1) from one party and a choice bit b from another party, and delivers the selected message m_b to the latter party. We further assume that multiple instances of the OT oracle can be invoked in parallel. By using a suitable composition theorem [Gol04,Can00,Can01], this oracle can be replaced by a protocol which securely realizes oblivious transfer. Let $x = (x_A, x_B)$ where $x_A = (x_1, \ldots, x_a)$ is Alice's input and $x_B = (x_{a+1}, \ldots, x_n)$ is Bob's input. The following protocol reveals $f(x)$ to Bob while hiding all additional information:

- Alice picks a random input r for $\hat{f}$. She computes $\hat{y}_0 = \hat{f}_0(r)$ and n pairs of messages $\hat{y}_{i,c} = \hat{f}_i(c, r)$ for $1 \leq i \leq n$ and $c = 0, 1$.
- Alice and Bob invoke $n - a$ parallel instances of oblivious transfer. In instance i, $a + 1 \leq i \leq n$, Bob selects from the pair of messages $(\hat{y}_{i,0}, \hat{y}_{i,1})$, which are both known to Alice, the message $\hat{y}_{i,x_i}$. In parallel, Alice sends to Bob the messages $\hat{y}_0$ and $\hat{y}_{i,x_i}$ for $1 \leq i \leq a$.
- Bob, who now holds the entire encoding $\hat{y} = \hat{f}(x, r)$, decodes $f(x)$ from $\hat{y}$.

Viewed as a protocol in the OT-hybrid model, Alice does not receive any message, and the messages received by Bob (from Alice and the OT oracle) reveal the encoding $\hat{y} = \hat{f}(x, r)$ and nothing else about x_A or r. It thus follows from the privacy of $\hat{f}$ that the protocol is secure against passive corruptions. The protocol is also secure against an actively corrupted Bob since any input he feeds into the OT oracle corresponds to an honest behavior on some valid input x_B. However, the protocol may not be secure against an actively corrupted Alice, since the inputs she feeds into the OT oracle may not be

consistent with a valid output of $\hat{f}$. See [IPS08,IKO+11] for variants of this protocol which offer security against an actively corrupted Alice with no additional interaction.

Computing on encrypted data. A scheme for computing on encrypted data [RAD78, SYY99] allows Bob to publish an encryption c of his input w such that any other party Alice can efficiently compute an encryption c' of $g(w)$ for a function g of her choice. Given c' and Bob's private randomness, Bob should be able to decrypt $g(w)$ but he should learn no additional information about g (except, perhaps, an upper bound of its description size). A solution to this problem can be obtained by applying an OT-based protocol as above to the universal function $f(g, w) = g(w)$. Using a 2-message implementation of oblivious transfer [NP01,AIR01,PVW08], the resulting protocol involves a single message of Bob followed by a single message of Alice. Bob's message can then be viewed as an encryption c of his input w and Alice's message as an encryption c' of the output $g(w)$.

A two-message protocol as above can also be viewed as a *fully homomorphic encryption scheme* which has the "function privacy" property (namely, the function g remains hidden), but does not have the "compactness" property (namely, the encrypted output can be bigger than the description length of g) [Gen09]. Variants which protect each party against a malicious behavior of the other were considered in [CCKM00,HK07,IKO+11]. "Multi-hop" variants which allow a sequence of computations g_i on an encrypted input were considered in [CCKM00,GHV10].

Private simultaneous messages protocols. A different application of decomposable randomized encodings to secure computation is in the non-interactive model of Feige, Kilian, and Naor [FKN94,IK97]. In this model, the input x for f is partitioned between k parties, whose goal is to communicate $f(x)$ to an external referee without revealing additional information about x. To this end, the parties are assumed to share a common source of randomness r which is unknown to the referee. The application of decomposable encodings to protocols in this model is straightforward: for each input bit x_i, the party who holds this input sends $\hat{f}_i(x_i, r)$ to the referee. The message $\hat{f}_0(r)$ can be sent by any of the parties. This protocol too is only secure in the presence of passive corruptions. Variants of the protocol that offer security against active corruptions are given in [FKN94,IKP10].

Arithmetic variants. In the above we viewed the input x as a bit-string. In applications that involve arithmetic computations over a ring R, it is useful to consider an arithmetic generalization of decomposable encodings in which the input x and output $\hat{y}$ are viewed as vectors over R. In this case, we would like every entry of $\hat{y}$ to be an *affine* function of a single input entry x_i; that is, every output entry can be written as $a(r) \cdot x_i + b(r)$ for some input entry x_i and arbitrary functions a, b of r. (The original definition of decomposable encoding coincides with this arithmetic variant over the binary field.) Efficient implementations of decomposable arithmetic encodings for arithmetic formulas and branching programs over finite rings are given in [IK97,IK02,CFIK03] and for arithmetic circuits over the integers in [AIK11].

A decomposable arithmetic encoding can be securely evaluated by using parallel calls to an oracle which implements the following arithmetic extension of oblivious transfer: Alice's input is a pair of ring elements (a, b), Bob's input is a ring element x, and Bob's output is $ax + b$. Efficient protocols for securely realizing such an oracle are given in [NP06,IPS09,DPSZ12].

Low online complexity. Settling for computational privacy, Yao's decomposable encoding has the following useful feature: the time required for computing the outputs $\hat{f}_1(x_1, r), \ldots, \hat{f}_n(x_n, r)$ which depend on x is independent of the complexity of f. This implies that an offline preprocessing phase in which $\hat{f}_0(r)$ is computed before the input x is available can be used to make the *online complexity* of computing $\hat{f}(x, r)$ much smaller than the complexity of computing $f(x)$. (In fact, it is easy to convert any decomposable encoding $\hat{f}$ into an encoding $\hat{f}'$ which has this feature by letting $\hat{f}'_0(r')$ include permuted, encrypted copies of the outputs $\hat{f}_i(0, r)$, $\hat{f}_i(1, r)$, and letting $\hat{f}'_i(x_i, r')$ reveal a short key which decrypts $\hat{f}_i(x_i, r)$.) The low online complexity of decomposable encodings can be used towards minimizing the online complexity of secure computation and interactive proofs [GGP10,AIK10b]. It should be noted, however, that the traditional privacy definition of decomposable randomized encodings only guarantees privacy under the assumption that the online input x is chosen independently of the offline output $\hat{f}_0(r)$. See [BHR12a,AIKW12] for positive and negative results on decomposable encodings which remain private even when x can be adaptively chosen based on $\hat{f}_0(r)$.

Applications of decomposable encodings beyond secure computation include functional encryption [SS10,GVW12], one-time programs [GKR08,GIS$^+$10,BHR12a], encryption schemes with security against key-dependent messages [BHHI10,App11a, BHR12b], and others. We refer the reader to [App11b,BHR12b] for a more extensive survey of decomposable encodings and their applications.

2.2. Low-Degree Encodings

A different notion of simplicity which is useful for secure computation is a low algebraic degree. Here we consider both the input x and the random input r as vectors over a finite field $\mathbb{F}$, and view $\hat{f}(x, r)$ as a mapping $\hat{f} : \mathbb{F}^n \times \mathbb{F}^m \to \mathbb{F}^s$. We say that $\hat{f}$ has degree d if each of the s outputs of $\hat{f}$ can be written as a polynomial in x and r whose total degree is at most d. (Note that both x and r count towards the degree.) We will often consider the question of encoding a boolean function $f : \{0,1\}^n \to \{0,1\}^l$, in which case the correctness and privacy of $\hat{f}$ are only required for inputs $x \in \{0,1\}^n$, but we will also consider arithmetic functions $f : \mathbb{F}^n \to \mathbb{F}^l$.

Randomized encodings with a low algebraic degree are also referred to as *randomizing polynomials*. Their study was initiated in [IK00] and later extended to stronger notions of simplicity [AIK06b,AIK09,AIK10b]. Ignoring efficiency, it is known that a degree-3 encoding exists for every function f over any finite field $\mathbb{F}$. Moreover, regardless of the field $\mathbb{F}$ or the encoding size, it is known that for most functions f, the degree of $\hat{f}$ cannot be lower than 3 if one insists on perfect privacy [IK00]. It is open whether a degree-2 encoding with statistical or computational privacy exists for all functions f. The question is equivalent to the existence of such encodings for the concrete Boolean function $f(x_1, x_2, x_3, x_4) = (x_1 \wedge x_2 \wedge x_3) \oplus x_4$.

For any constant degree bound d, a degree-d encoding can be converted to a degree-3 encoding of the same function with polynomial overhead. Hence, we will use the terms "low-degree encoding" and "degree-3 encoding" interchangeably. The natural complexity classes for which *polynomial-size* low-degree encodings are known to exist are similar to the classes for which polynomial-size decomposable encodings are known to exist (see Section 2.1 above). This is not a mere coincidence: every low-degree encoding can be converted into a decomposable encoding with a polynomial overhead. One minor

difference is that the existence of polynomial-size (computationally-private) low-degree encodings for all polynomial-time computable functions relies on the existence of a one-way function in NC^1 or in similar classes [AIK06a,HRV10], whereas for decomposable encodings any one-way function suffices. Also, there are natural complexity classes (such as non-deterministic logspace) for which all known constructions of low-degree encodings require to settle for either statistical correctness or statistical privacy [IK02], whereas perfect correctness and privacy are possible for decomposable encodings. Whether efficient statistical (or perfect) low-degree encodings exist for all polynomial-time computable functions is a major open question. There is evidence against this possibility for a useful special class of degree-3 encodings, namely those that have degree 2 in the randomness [IKP12].

Low-degree randomized encodings were initially motivated by the goal of minimizing the round complexity of secure computation [IK00]. The underlying observation is that in most natural protocols, the round complexity grows linearly with the *multiplicative depth* of an arithmetic circuit (with fan-in 2) computing the functionality. This holds both for protocols which assume an honest majority [BGW88,CCD88] and for protocols which do not assume an honest majority [GMW87]. Hence, by securely evaluating a low-degree encoding $\hat{f}$ of a higher degree functionality f one can reduce the number of rounds required for computing f. More specifically, a degree-3 encoding of f can be combined with a secure protocol for evaluating degree-3 polynomials to yield a secure protocol for f. As a concrete application, this yields general 2-round k-party protocols (over secure point-to-point channels) which are secure against $t < k/3$ passively corrupted parties. The same result holds in the presence of active corruptions, assuming that the adversary is allowed to choose which subset of the honest parties receive their output [IKP10].

We now give a more detailed overview of the application to round-efficient secure computation. Fix a finite field $\mathbb{F}$ and consider functionalities defined over inputs in $\mathbb{F}^n$. For simplicity, we assume here that all functionalities receive an input from each party and deliver the same output to all parties; this can be extended, via a standard reduction, to functionalities which deliver different outputs to different parties.

As noted above, a low-degree randomized encoding $\hat{f}$ of f reduces the task of securely computing f to the task of securely computing the lower-degree functionality $\hat{f}$. However, here $\hat{f}$ must be treated as a randomized functionality, since its random input r must remain secret. We show that it is in fact sufficient to securely compute a low-degree *deterministic* functionality related to $\hat{f}$ in order to securely compute f.

Concretely, suppose that every deterministic k-party functionality g of algebraic degree d can be realized by a protocol Π_g using ρ communication rounds. Now, suppose we are given a k-party functionality $f(x_1, \ldots, x_k)$ that has degree $d_f > d$. A ρ-round protocol Π_f for f can be obtained from a degree-d encoding $\hat{f}((x_1, \ldots, x_k), r)$ of f as follows. We view $\hat{f}$ as a randomized k-party functionality whose internal randomness is r. To reduce the secure evaluation of $\hat{f}$ to that of a *deterministic* functionality of the same algebraic degree, we use a standard reduction of randomized functionalities to deterministic functionalities (cf. [Gol04]). Define a deterministic k-party functionality g by $g((x_1, r_1), \ldots, (x_k, r_k)) = \hat{f}((x_1, \ldots, x_k), \sum_{i=1}^{k} r_i)$, where each r_i is a vector over $\mathbb{F}$ of the same length as r. Note that the degree of g is the same as that of $\hat{f}$. Given an ideal oracle access to g, a secure protocol Π_f for f may proceed as follows:

- Each party P_i, on input x_i, picks r_i uniformly at random and provides g with the input (x_i, r_i).
- Let $\hat{y}$ denote the output of g. The output y of Π_f is obtained by applying the decoder of $\hat{f}$ to $\hat{y}$.

The security of Π_f holds for any number of (passively or actively) corrupted parties, in almost every model for secure computation from the literature.[2] Using suitable composition theorems [Can00,Can01,Gol04], we can obtain a ρ-round protocol for f by invoking the given ρ-round protocol Π_g for g to emulate the ideal oracle call to g.

2.3. Local Encodings

A final simplicity feature we will discuss is "locality". This feature is motivated by the goal of minimizing the *parallel time complexity* of cryptographic computations.

For a positive integer d, we say that a function $g : \{0,1\}^n \to \{0,1\}^l$ has *output locality* d if each of its output bits depends on at most d input bits, and it has *input locality* d if each of its input bits influences at most d output bits. If $g : \{0,1\}^* \to \{0,1\}^*$ has constant output locality (namely, there is a constant d such that each of its restrictions $g_n : \{0,1\}^n \to \{0,1\}^{l(n)}$ has output locality d), then g is said to be in NC^0. The class NC^0 can be considered to capture "constant parallel time," since a function in this class can be computed by a family of boolean circuits with bounded fan-in and constant depth. If g further has constant input locality, then this constant-depth family can additionally have bounded fan-out.

We say that the randomized encoding $\hat{f}(x, r)$ has (input or output) locality d, if it has locality d when viewed as a function of both x and r. Note that in contrast to the previous notion of decomposable encodings, the bits of r also count towards the locality. In fact, a decomposable encoding as in Section 2.1 can be equivalently defined as an encoding $\hat{f}(x, r)$ such that for any fixed r_0 the function $g(x) = \hat{f}(x, r_0)$ has output locality 1.

It is known that every function $f(x)$ admits a randomized encoding $\hat{f}(x, r)$ with output locality 4 [AIK06b]. Moreover, the efficiency of such encodings is equivalent to that of low-degree encodings (see Section 2.2) up to a polynomial overhead. In contrast, encodings with constant input locality only exist for restricted function classes which include all linear functions but do not include all quadratic functions [AIK09]. An even stronger notion of locality, which takes the physical distance between inputs and outputs into account, was considered in [AIK10a] and motivated by the goal of efficiently implementing cryptographic primitives on cellular automata.

The application of local encodings for reducing the parallel complexity of cryptographic computations is as follows. Suppose that f is a cryptographic function; for instance, f may compute a one-way function, a public key encryption, or the next message function of a secure computation protocol. The idea is that instead of computing $y = f(x)$, one can pick r at random and compute $\hat{y} = \hat{f}(x, r)$. In cases where some

[2] An exception is the case of security against an *adaptive* adversary who may eventually corrupt all k parties [CFGN96,GS12]. Unless honest parties are allowed to erase their randomness r_i, a simulator for proving the security of Π_f should first simulate $\hat{y}$ given y and then, after learning the inputs x_i, generate simulated random inputs r_i (which determine the randomness r for $\hat{f}$) consistently with x_i and $\hat{y}$. While this can be done for statistically private low-degree encodings that apply to low complexity classes such as NC^1, it is not known to be possible for any (computationally private) low-degree encoding scheme which efficiently applies to all polynomial-time computable functions.

receiver should make use of y, the receiver applies the decoder of $\hat{f}$ to compute y from $\hat{y}$. But in some cases the output $\hat{y}$ can be used without decoding. For instance, when f is a one-way function, the function $\hat{f}$ is a one-way function even when viewed as a deterministic function of (x, r). The same holds for pseudorandom generators and collision-resistant hash functions, assuming that $\hat{f}$ satisfies some additional regularity properties [AIK06b]. One corollary is that the existence of one-way functions and non-trivial pseudorandom generators in NC^0 follows from the existence of one-way functions in NC^1, which in turn follows from most standard intractability assumptions in cryptography.

Local randomized encodings are also useful for obtaining parallel implementations of primitives outside the domain of cryptography, such as small-bias generators [AIK06b], proof verification and decoding [GGH$^+$07], and program checking [GGH$^+$08,AIK10b].

We finally note that by applying the above encoding techniques on top of each other, it is possible to encode any function $f : \{0,1\}^n \to \{0,1\}^l$ by a randomized encoding $\hat{f}(x, r)$ which simultaneously enjoys all of the simplicity features we discussed: it is decomposable, has algebraic degree 3 over the binary field, and has output locality 4.

3. Definitions

In this section we define the general notion of randomized encodings of functions and discuss some of its useful properties. We start with the simplest variant, of a perfectly correct and private encoding.

Definition 3.1 (Randomized encoding) *Let $X, Y, \hat{Y}, R$ be finite sets and let $f : X \to Y$. We say that a function $\hat{f} : X \times R \to \hat{Y}$ is a* randomized encoding *of f, if it satisfies the following requirements:*

- **Correctness.** *There exists a function* Dec*, called a* decoder*, such that for every $x \in X$ and $r \in R$ we have $\mathrm{Dec}(\hat{f}(x, r)) = f(x)$. Equivalently, for any x, x' such that $f(x) \neq f(x')$, the random variables $\hat{f}(x, r)$ and $\hat{f}(x', r')$ induced by a uniform choice of r and r' from R have disjoint support sets.*
- **Privacy.** *There exists a randomized function* Sim*, called a* simulator*, such that for every $x \in X$, the distribution $\mathrm{Sim}(f(x))$ is identical to the distribution of $\hat{f}(x, r)$ induced by a uniformly random choice of r from R. Equivalently, for any x, x' such that $f(x) = f(x')$, the random variables $\hat{f}(x, r)$ and $\hat{f}(x', r')$ are identically distributed.*

The above definition treats f as a finite function and is not concerned with the efficiency of $\hat{f}$, Dec and Sim. In cryptographic applications, one is typically interested in obtaining an *efficient compiler* which maps a representation of f to representations of $\hat{f}, \mathrm{Dec}$, and Sim. Most of the constructions we will describe in Section 4 implicitly define a polynomial-time compiler which given some representation of f (e.g., by a circuit, a formula, or a branching program) outputs a circuit representation of $\hat{f}, \mathrm{Dec}$, and Sim.

It is often useful to relax the correctness and privacy guarantees of randomized encodings. We define such relaxations below.

Definition 3.2 (Relaxed correctness and privacy) *Let $X, Y, R, \hat{Y}, f, \hat{f}$ be as in Definition 3.1. We define the following relaxed correctness and privacy properties of $\hat{f}$:*

- **(Statistical) δ-correctness.** *There exists a probabilistic function* Dec *such that for every $x \in X$, $\Pr[\mathrm{Dec}(\hat{f}(x,r)) \neq f(x)] \leq \delta$, where the probability is taken over a uniform choice of r from R and the randomness of* Dec.
- **(Statistical) ε-privacy.** *There exists a randomized function* Sim *such that for every $x \in X$,*

$$\mathrm{SD}(\mathrm{Sim}(f(x)), \hat{f}(x,r)) \leq \varepsilon,$$

  where r is chosen uniformly from R and where SD *denotes statistical distance.*
- **(Computational) (ε, t)-privacy.** *Suppose that $\hat{Y} = \{0,1\}^m$. Then $\hat{f}$ is said to be (ε, t)-private (with respect to f) if there exists a randomized function* Sim *outputting strings from $\{0,1\}^m$, such that for every $x \in X$ and boolean circuit D of size t,*

$$\left| \Pr[D(\mathrm{Sim}(f(x))) = 1] - \Pr[D(\hat{f}(x,r)) = 1] \right| \leq \varepsilon.$$

When considering an efficient compiler which produces imperfect encoder, decoder, and simulator, the compiler may take a security parameter k as an additional input. In this case, the typical requirement is that the compiler run in time polynomial in k and provide the guarantees that δ, ε are negligible in k (in the statistical case) and additionally t is super-polynomial in k (in the computational case).

It is often convenient to manipulate randomized encodings by *concatenating* encodings of two or more functions and by *composing* encodings, namely applying one encoding on top of another. For simplicity we restrict the attention here to the perfect case. We refer the reader to [AIK06b,AIK11] for the general case.

Lemma 3.3 (Concatenation) *Suppose $\hat{f}_i(x, r_i)$ is a randomized encoding of $f_i(x)$ for $i = 1, \ldots, k$. Then the function $\hat{f}(x, (r_1, \ldots, r_k)) \stackrel{def}{=} (\hat{f}_1(x, r_1), \ldots, \hat{f}_k(x, r_k))$ is a randomized encoding of $f(x) \stackrel{def}{=} (f_1(x), \ldots, f_k(x))$.*

Lemma 3.4 (Composition) *Suppose $\hat{f}(x, r)$ is a randomized encoding of $f(x)$ and $\hat{f}'((x, r), r')$ is a randomized encoding of $\hat{f}(x, r)$ (viewing the latter as a deterministic function of (x, r)). Then $\hat{f}''(x, (r, r')) \stackrel{def}{=} \hat{f}'((x, r), r')$ is a randomized encoding of $f(x)$.*

4. Constructions of Randomized Encodings

In this section we describe different constructions of randomized encodings, grouped according to the notions of simplicity they satisfy.

4.1. Decomposable Encodings

We start by defining a generalization of the notion of decomposable encodings described in Section 2. This definition coincides with the "simultaneous messages" model of secure computation put forward by Feige, Kilian, and Naor [FKN94].

Definition 4.1 (Decomposable randomized encoding) *For $f : X_1 \times \cdots \times X_n \to Y$, a* decomposable randomized encoding *of f is one that has the form*

$$\hat{f}((x_1, \ldots, x_n), r) = (\hat{f}_1(x_1, r), \ldots, \hat{f}_n(x_n, r))$$

for some functions $\hat{f}_i : X_i \times R \to \hat{Y}_i$.

In the rest of this section we present three different constructions. Each of these constructions leads to the conclusion that every finite function $f : X_1 \times \cdots \times X_n \to Y$ admits a decomposable encoding. However, the constructions have very different efficiency features. The first construction, based on [FKN94], relies on a truth-table representation and will therefore typically be inefficient. The second construction, based on [AIK11], is a modular variant of Yao's garbled circuit construction which relies a representation of f by a circuit or formula. (A formula is a circuit with fan-out 1.) Requiring perfect correctness and privacy, the complexity of this construction is exponential in the circuit depth or polynomial in the formula size. This applies both to boolean and arithmetic circuits. Settling for computational privacy and assuming the existence of a pseudorandom generator, the complexity of the construction can be made linear in the circuit size. The third construction, based on [Kil88], relies on a representation of f using an iterated group product. Combined with a result of Barrington [Bar86], it yields an encoding whose complexity is polynomial in the formula size.

4.1.1. Decomposable encodings from truth-tables

In this section we give a modular presentation of a construction from [FKN94], whose complexity is roughly linear in the truth-table size of f (more precisely, in the number of nonzero truth-table entries). The first building block, which is of independent interest, is a simple decomposable encoding for summation in finite abelian groups.

Claim 4.2 (Decomposable encoding for group summation) *Let G be a finite abelian group and let $f_{\mathrm{sum}} : G^n \to G$ be the group summation function $f_{\mathrm{sum}}(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i$. Let $R = \{(r_1, \ldots, r_n) \in G^n : \sum_{i=1}^{n} r_i = 0\}$. Then the function $\hat{f}_{\mathrm{sum}} : G^n \times R \to G^n$ defined by $\hat{f}_{\mathrm{sum}}((x_1, \ldots, x_n), (r_1, \ldots, r_n)) = (x_1 + r_1, \ldots, x_n + r_n)$ is a decomposable encoding of f_{sum}.*

Proof: It is easy to verify that $\hat{f}_{\mathrm{sum}}$ maps an input $x = (x_1, \ldots, x_n)$ to a uniformly random input $x' \in G^n$ such that $f_{\mathrm{sum}}(x') = f_{\mathrm{sum}}(x)$. Thus, we can let $\mathrm{Dec} = f_{\mathrm{sum}}$ and let $\mathrm{Sim}(y)$ output a random n-tuple in $f_{\mathrm{sum}}^{-1}(y)$. ■

Next, we use the above example to encode a boolean function f_{szt} which instead of outputting the sum of the n inputs only determines whether the sum is zero. Here it is convenient to work over a finite field $\mathbb{F}$. The idea is to first encode f_{szt} by $\hat{f}((x_1, \ldots, x_n), r) \stackrel{\mathrm{def}}{=} r \cdot (x_1 + \ldots + x_n)$, where r is a random nonzero field element (i.e., $R = \mathbb{F} \setminus \{0\}$), and then apply the encoding from Claim 4.2 (via Lemma 3.4) to decompose $\hat{f}$. Concretely, the second step encodes $r \cdot (x_1 + \ldots + x_n) = rx_1 + \cdots + rx_n$ by $(rx_1 + r_1, \ldots, rx_n + r_n)$ where $(r_1, \ldots, r_n)$ are random field elements that sum up to 0. This construction is captured by the following claim.

Claim 4.3 (Decomposable encoding for summation zero-test) *Let $\mathbb{F}$ be a finite field and let $f_{\mathrm{szt}} : \mathbb{F}^n \to \{\mathrm{Yes}, \mathrm{No}\}$ be the summation zero-test function defined by:*

$$f_{\mathrm{szt}}(x_1, \ldots, x_n) = \begin{cases} \mathrm{Yes}, & \sum_{i=1}^{n} x_i = 0, \\ \mathrm{No}, & \textit{otherwise}. \end{cases}$$

Let $R_{\mathrm{szt}} = \{(r_0, r_1, \ldots, r_n) \in \mathbb{F}^{n+1} : r_0 \in \mathbb{F} \setminus \{0\}, \sum_{i=1}^{n} r_i = 0\}$. Then the function $\hat{f}_{\mathrm{szt}} : \mathbb{F}^n \times R_{\mathrm{szt}} \to \mathbb{F}^n$ defined by $\hat{f}_{\mathrm{szt}}((x_1, \ldots, x_n), (r_0, r_1, \ldots, r_n)) = (r_0 x_1 + r_1, \ldots, r_0 x_n + r_n)$ is a decomposable encoding of f_{szt}.

Proof: As noted above, $\hat{f}_{\mathrm{szt}}$ can be derived by applying the Composition Lemma (Lemma 3.4) to Claim 4.2 and a simple non-decomposable encoding of f_{szt}. To analyze the construction directly, note that $\hat{f}_{\mathrm{szt}}$ maps any fixed input $x \in \mathbb{F}^n$ to a uniformly random input $x' \in \mathbb{F}^n$ such that $f_{\mathrm{szt}}(x') = f_{\mathrm{szt}}(x)$. Thus the decoder can be defined by $\mathrm{Dec}(\hat{y}_1, \ldots, \hat{y}_n) = f_{\mathrm{szt}}(\hat{y}_1, \ldots, \hat{y}_n)$ and the simulator, on input $y \in \{\mathrm{Yes}, \mathrm{No}\}$, can output a random n-tuple from $f_{\mathrm{szt}}^{-1}(y)$. ■

Given Claim 4.3, we can easily obtain a decomposable encoding any *indicator function* which tests whether the inputs take specific values.

Definition 4.4 (Indicator function) *For $a \in X$, the* indicator function $I_a : X \to \{\mathrm{Yes}, \mathrm{No}\}$ *outputs* Yes *if the input is equal to a and outputs* No *otherwise. We will identify* Yes *with the value* 0 *and* No *with the value 1.*

Claim 4.5 (Decomposable encoding for indicator functions) *Let $X = X_1 \times \cdots \times X_n$ and $a = (a_1, \ldots, a_n) \in X$. Let p be a prime such that $p > n$, and let R_{szt} and $\hat{f}_{\mathrm{szt}}$ be as in Claim 4.3 over a field $\mathbb{F}$ of size p. Then the function $\hat{I}_a : X \times R_{\mathrm{szt}} \to \mathbb{F}^n$ defined by*

$$\hat{I}_a((x_1, \ldots, x_n), r) = \hat{f}_{\mathrm{szt}}((I_{a_1}(x_1), \ldots, I_{a_n}(x_n)), r)$$

is a decomposable encoding of I_a.

Proof: Since $p > n$, we have $\sum_{i=1}^{n} I_{a_i}(x_i) \equiv 0 \mod p$ if and only if $I_{a_i}(x_i) = 0$, namely $x_i = a_i$, for all i. Thus, $f_{\mathrm{szt}}(I_{a_1}(x_1), \ldots, I_{a_n}(x_n)) = I_a(x_1, \ldots, x_n)$. It follows that $\hat{f}_{\mathrm{szt}}((I_{a_1}(x_1), \ldots, I_{a_n}(x_n)), r)$ encodes I_a. ■

As special cases, this gives very efficient decomposable encodings for the boolean AND and OR functions. If one is willing to increase the encoding size by a factor equal to the truth-table size (or, more precisely, the number of nonzero entries), it is possible to extend this solution to one that applies to every function $f : X_1 \times \cdots \times X_n \to \{0, 1\}$. The encoding of f is obtained by concatenating encodings of all indicator functions I_a for a such that $f(a) = 1$. In case $f(x) = 1$, we need to hide the identity of the indicator function which is satisfied. This is done by randomly permuting the encodings $\hat{I}_a$.

Construction 4.6 (Decomposable encoding for binary truth-tables) *Let $X = X_1 \times \cdots \times X_n$, let $f : X \to \{0, 1\}$, and let $A = \{a \in X : f(a) = 1\}$. Let P be a set of permutations over A with the property that for any $a \in A$, a uniformly chosen $\pi \in P$ makes $\pi(a)$ uniformly distributed over A (e.g., P can be the set of all "cyclic shifts" of A with respect to some linear order). Let $R = P \times R_{\mathrm{szt}}^{|A|}$ where R_{szt} is as in Claim 4.3. For $r = (\pi, (r_a)_{a \in A}) \in R$, define the encoding:*

$$\hat{f}(x,r) = \left(\hat{I}_{\pi(a)}(x,r_a)\right)_{a\in A}.$$

Claim 4.7 *For any $f : X_1 \times \cdots \times X_n \to \{0,1\}$, the function $\hat{f}$ defined in Construction 4.6 is a decomposable randomized encoding of f.*

Proof: First, observe that f is encoded by the function $g(x,\pi) = \left(I_{\pi(a)}(x)\right)_{a\in A}$, where π is drawn uniformly at random from Π. Indeed, if $f(x) = 0$ then the output of g consists only of 'No' symbols, whereas if $f(x) = 1$ then the output contains a single 'Yes' symbol in a random position. The encoding $\hat{f}$ is obtained from g and the encodings $\hat{I}_a$ by using Lemma 3.3 and Lemma 3.4. ■

Using a binary representation of the output, any function $f : X \to Y$ can be viewed as a concatenation of functions $f_i : X \times \{0,1\}$. Thus, by combining Claim 4.7 and Lemma 3.4 we get the following theorem.

Theorem 4.8 *Every finite function $f : X_1 \times \cdots \times X_n \to Y$ admits a decomposable randomized encoding.*

4.1.2. Decomposable encodings from circuits

In this section we describe a composition-based approach for decomposable encoding, which can be viewed as an abstraction of Yao's garbled circuit construction. This approach was used in [AIK11] to encode arithmetic circuits. We refer the reader to [IK02,LP09,AIK06a,BHR12b] for a description and analysis of more traditional flavors of the garbled circuit construction.

The high level idea of the composition-based approach is as follows. Suppose that we can transform a decomposable encoding of $f(x)$ to a decomposable encoding of $f'(x') = f(g(x'))$, for any "simple" function g, with only a small overhead. Then we can repeat this step for each layer g_i of a circuit, going from the top to the bottom, eventually obtaining an encoding for the function computed by the entire circuit.

To carry out the above approach, it suffices to rely in a black-box way on the existence of a decomposable encoding for every finite function, which was already established in the previous section. However, we prefer a self-contained derivation which will also be useful towards obtaining low-degree encodings. It will be convenient to rely on the following algebraic variant of decomposable encodings.

Definition 4.9 (Decomposable affine randomized encoding (DARE)) *Let $\mathbb{F}$ be a finite ring and let $f : \mathbb{F}^n \to \mathbb{F}^l$. A* decomposable affine randomized encoding (DARE) *of f is an encoding $\hat{f} : \mathbb{F}^n \times \mathbb{F}^m \to \mathbb{F}^s$ of f that has the form*

$$\hat{f}((x_1,\ldots,x_n),r) = (\hat{f}_0(r), \hat{f}_1(x_1,r), \ldots, \hat{f}_n(x_n,r))$$

for some functions $\hat{f}_0 : \mathbb{F}^m \to \mathbb{F}^{s_0}$ and $\hat{f}_i : \mathbb{F} \times \mathbb{F}^m \to \mathbb{F}^{s_i}$ such that $\hat{f}_i$ has degree 1 in x_i. That is, $\hat{f}_i(x_i,r)$ can be written as $a_i(r)\cdot x_i + b_i(r)$ for arbitrary functions $a_i, b_i : \mathbb{F}^m \to \mathbb{F}^{s_i}$.

Note that a DARE over the binary field coincides with the previous notion of decomposable randomized encoding; however, over larger rings this notion is strictly stronger.

An *arithmetic circuit* over a ring $\mathbb{F}$ is defined similarly to a standard boolean circuit, except that the inputs x_i can take arbitrary values from $\mathbb{F}$ and each gate can add or multiply over $\mathbb{F}$ the values of its two inputs. (An input gate can contain either an input variable x_i or a constant value; this makes it possible to implement $a-b$ by $a+(-1)\cdot b$.) Note that arithmetic circuits generalize the standard model of boolean circuits in the sense that any boolean circuit can be simulated by an arithmetic circuit over the binary field with a small overhead.

We will use the composition-based approach to construct a DARE for every arithmetic circuit over $\mathbb{F}$, where the encoding size is exponential in the circuit depth. Since any arithmetic formula can be simulated by a circuit whose depth is logarithmic in the formula size [Bre74], this yields polynomial-size encodings for arithmetic formulas.

The construction we present next relies on a concrete DARE referred to as the *affinization gadget*. Here and in the following, we will make free use of concatenation and composition of randomized encodings (Lemmas 3.3 and 3.4).

Claim 4.10 (Affinization gadget) *Let $\mathbb{F}$ be a finite ring and let $f_{\mathrm{AG}} : \mathbb{F}^3 \to \mathbb{F}$ be the function defined by $f_{\mathrm{AG}}(x_1, x_2, x_3) = x_1x_2 + x_3$. Then f_{AG} is encoded by the DARE $\hat{f}_{\mathrm{AG}} : \mathbb{F}^3 \times \mathbb{F}^4 \to \mathbb{F}^5$ defined by $\hat{f}_{\mathrm{AG}}((x_1, x_2, x_3), (r_1, r_2, r_3, r_4)) = (a_1x_1 + b_1, a_2x_2 + b_2, a_3x_3 + b_3)$, where $a_1 = (1, r_2)$, $b_1 = (-r_1, -r_1r_2 + r_3)$, $a_2 = (1, r_1)$, $b_2 = (-r_2, r_4)$, $a_3 = 1$, and $b_3 = -r_3 - r_4$.*

Proof: First note that the function $f_{\mathrm{sum}}(x, x') = x+x'$ is encoded by $\hat{f}_{\mathrm{sum}}((x, x'), r) = (x + r, x' - r)$. (This is a special case of Claim 4.2.) To prove the correctness and privacy of $\hat{f}_{\mathrm{AG}}$, we use the following intermediate encoding of f_{AG} (which is affine but not decomposable):

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

It is not hard to see that $\hat{f}'_{\mathrm{AG}}$ encodes f_{AG}: the decoder and simulator can be defined by $\mathrm{Dec}(\hat{y}_1, \hat{y}_2, \hat{y}_3) = \hat{y}_1\hat{y}_2 + \hat{y}_3$ and $\mathrm{Sim}(y; s_1, s_2) = (s_1, s_2, y - s_1s_2)$. Their validity follows from the fact that the first two outputs $\hat{y}_1, \hat{y}_2$ of $\hat{f}'_{\mathrm{AG}}$ are uniformly random and independent, and the third output is $\hat{y}_3 = f_{\mathrm{AG}}(x_1, x_2, x_3) - \hat{y}_1\hat{y}_2$. The encoding $\hat{f}_{\mathrm{AG}}$ is obtained from $\hat{f}'_{\mathrm{AG}}$ by applying the encoding $\hat{f}_{\mathrm{sum}}$ (twice, using the additional randomness r_3, r_4) to break the terms in the last entry of $\hat{f}'_{\mathrm{AG}}$ and then arranging the outputs in the canonical format of Definition 4.9. ■

We now use the affinization gadget for implementing the basic composition step.

Lemma 4.11 (Composition step) *Let $\mathbb{F}$ be a finite ring and let $f : \mathbb{F}^n \to \mathbb{F}^l$. Let $g : \mathbb{F}^{n'} \to \mathbb{F}^n$ be a depth-1 arithmetic circuit, i.e., each output of g is either the sum or the product of two inputs. Suppose f admits a DARE $\hat{f}$ with parameters $s_0, \ldots, s_n$ as in Definition 4.9. Then, the function $f' : \mathbb{F}^{n'} \to \mathbb{F}^l$ defined by $f'(x') = f(g(x'))$ admits a DARE $\hat{f}'$ with parameters $s'_0, \ldots, s'_n$ such that $s'_0 = s_0$ and $\sum_{i=1}^{n'} s'_i \leq 5\sum_{i=1}^{n} s_i$.*

Proof: Let $\hat{f}(x, r) = (b_0(r), a_1(r) \cdot x_1 + b_1(r), \ldots, a_n(r) \cdot x_n + b_n(r))$ be the given DARE for f. Substituting $g(x')$ for x, the function $\hat{f}''(x', r) \stackrel{\text{def}}{=} \hat{f}(g(x'), r)$ encodes $f'(x') = f(g(x'))$. However, the encoding $\hat{f}''$ is no longer decomposable. We further encode it to make it decomposable by applying the summation and affinization gadgets. Concretely, for $i = 1, \ldots, n$:

- If $x_i = x'_j + x'_k$ (i.e., the i-th output of g is $x'_j + x'_k$), we bring $a_i(r) \cdot (x'_j + x'_k) + b_i(r)$ to a DARE format using the summation gadget $\hat{f}_{\mathrm{sum}}$. That is, we encode this expression by $(a_i(r) \cdot x'_j + r', a_i(r) \cdot x'_k + (b_i(r) - r'))$ where r' is a fresh vector of random inputs of length s_i. The total output length of this DARE is $2s_i$.
- If $x_i = x'_j \cdot x'_k$, we bring $a_i(r) \cdot (x'_j \cdot x'_k) + b_i(r)$ to a DARE format by first parsing it as $(a_i(r) \cdot x'_j) \cdot x'_k + b_i(r)$ and then applying the affinization gadget to each of the s_i entries of the output. The total output length of the resulting DARE is $5s_i$.

Overall, we left $b_0(r)$ unchanged (hence $s'_0 = s_0$) and we generated at most $5 \cdot \sum_{i=1}^{n} s_i$ additional outputs, which can be rearranged in the format of Definition 4.9. ■

By iterating the composition step, we can compile an arbitrary arithmetic circuit into a corresponding DARE whose output length is exponential in the circuit depth.

Theorem 4.12 *Let C be an arithmetic circuit of depth d over a finite ring $\mathbb{F}$ computing a function $f : \mathbb{F}^n \to \mathbb{F}^l$. Then f admits a DARE over $\mathbb{F}$ with output length $l \cdot 2^{O(d)}$.*

Proof: We can assume without loss of generality that C is layered, namely $f(x) = C_1 \circ C_2 \circ \cdots \circ C_d(x)$ where each C_i is computed by a depth-1 arithmetic circuit. (Any circuit can be converted to this form without increasing the depth.) Let $f_0 : \mathbb{F}^l \to \mathbb{F}^l$ be the identity function on the outputs and $f_i = f_{i-1} \circ C_i$ for $i = 1, \ldots, d$. We obtain a DARE $\hat{f}_d$ for $f_d = f$ via the following iterative process:

- Let $\hat{f}_0 = f_0$.
- For $i = 1, \ldots, d$, obtain a DARE $\hat{f}_i$ for f_i by applying the composition step (Lemma 4.11) with $f = f_{i-1}$, $\hat{f} = \hat{f}_{i-1}$, and $g = C_i$.

In each step, the output length of the encoding grows by at most a constant factor. ■

Letting $\mathbb{F}$ be the binary field, Theorem 4.12 yields a polynomial-size decomposable encoding for functions f in the complexity class NC^1 (equivalently, functions f with polynomial-size formulas). More efficient constructions for the binary and arithmetic case are given in [Kol05] and [CFIK03], respectively.

Relaxing privacy. We now explain how the exponential blowup in the depth can be avoided by settling for computational privacy. We focus here on the binary case; an extension to arithmetic circuits over the integers is given in [AIK11]. Consider the encoding $\hat{f}'$ obtained via composition step of Lemma 4.11. In the binary case, the parameter s'_i captures the input locality of the bit x'_i in $\hat{f}'$. The effect of x'_i on the output of $\hat{f}'$ is equivalent to a *selection* between two binary strings (z_0, z_1) of length s'_i each, where each z_i depends only on the randomness of $\hat{f}'$ and not on the input x'. The key observation is that when the input locality s'_i grows beyond a computational security parameter k, we can use symmetric encryption to reduce it back to k at the expense of slightly increasing the length s'_0 of $\hat{y}'_0$. (The latter corresponds to the "garbled circuit" part in Yao's construction.) Since $\hat{y}'_0$ is not further encoded, this eliminates the exponential blowup.

Concretely, we would like to encode the selection function $f_{\mathrm{sel}}(b, z_0, z_1) = z_b$, where $b \in \{0,1\}$ and $z_0, z_1 \in \{0,1\}^K$, by a function $\hat{f}_{\mathrm{sel}}((b, z_0, z_1), r)$ for which at most k bits of the output depend on b. A simple way to implement $\hat{f}_{\mathrm{sel}}$ using symmetric encryption is to encrypt z_0 and z_1 under k-bit random keys κ_0 and κ_1, respectively, and output the pair of ciphertexts in a random order along with the key κ_b. In order to allow the correct decoding of z_b, the encryption scheme should be verifiable

(i.e., decryption with incorrect key can be identified), which typically leads to a statistically small decoding error (as in the garbled circuit variant used in [LP09]). A perfectly correct version which can rely on an arbitrary symmetric encryption scheme (alternatively, a pseudorandom generator) can be obtained by appending to the encoding the value $\pi \oplus b$, where π is the random bit which determines whether the two ciphertexts are swapped. This bit points to the ciphertext which the decoder should decrypt without revealing information about b. (This version corresponds to the garbled circuit variant used in [BMR90,NPS99,AIK06a].)

With the above mechanism for containing the input locality in place, the composition step increases the length of the encoding by at most an *additive* term of $O(k(n+n'))$. The overall encoding length in the iterative construction becomes $O(k \cdot |C|)$ where $|C|$ is the number of gates in C and k is a security parameter (size of an encryption key which suffices for the desired level of computational privacy).

4.1.3. Decomposable encodings from group products

We now present a third approach for constructing decomposable encodings, relying on the computational power of non-abelian groups. We start with a simple construction of decomposable encodings for an iterated group product due to Kilian [Kil88]. This construction naturally extends the construction for abelian groups given in Claim 4.2.

Claim 4.13 (Decomposable encoding for group product) *Let G be a finite group and let $f_{\mathrm{prod}} : G^n \to G$ be the group product function $f_{\mathrm{prod}}(x_1, \ldots, x_n) = \prod_{i=1}^{n} x_i$. Then the function $\hat{f}_{\mathrm{prod}} : G^n \times G^{n-1} \to G^n$ defined by*

$$\hat{f}_{\mathrm{prod}}((x_1, \ldots, x_n), r) = (x_1 r_1,\ r_1^{-1} x_2 r_2,\ r_2^{-1} x_3 r_3, \ldots, r_{n-2} x_{n-1} r_{n-1},\ r_{n-1}^{-1} x_n)$$

is a decomposable encoding of f_{prod}.

Proof: As in Claim 4.2, it suffices to argue that for any $x \in G^n$, the output $\hat{y}$ of $\hat{f}_{\mathrm{prod}}(x, r)$ is uniformly distributed over the set of $x' \in G^n$ such that $f_{\mathrm{prod}}(x') = f_{\mathrm{prod}}(x)$. It is easy to verify that $f_{\mathrm{prod}}(\hat{y}) = f_{\mathrm{prod}}(x)$ for every choice of r. Also note that each entry of $\hat{y}$ except the last involves a new random input r_i which does not appear in previous entries. This ensures that the first $n-1$ entries of $\hat{y}$ are uniformly distributed over G^{n-1}. Since the last entry of $\hat{y}$ is determined so that $f_{\mathrm{prod}}(\hat{y}) = f_{\mathrm{prod}}(x)$, the distribution of $\hat{y}$ must indeed be uniform subject to $f_{\mathrm{prod}}(\hat{y}) = f_{\mathrm{prod}}(x)$. ■

A celebrated theorem of Barrington [Bar86] shows that evaluating a boolean circuit of depth d reduces to computing an iterated product of $m = 2^{O(d)}$ elements $g_1, \ldots, g_m$ from the symmetric group S_5, where each g_i is determined by a single bit of the input. Moreover, the output of the group product is in one-to-one correspondence to the output of the circuit (that is, there are two fixed group elements that correspond to the two possible outputs). Thus, applying Claim 4.13 to decompose this iterated group product, we get a very different alternative proof for Theorem 4.12 in the binary case.

The results of [Cle90,BC92] give more efficient variants of Barrington's theorem and extend it to arithmetic circuits. In particular, it is shown that for any $\varepsilon > 0$ there is a finite group G_ε (whose size grows as ε tends to 0) such that by using an iterated product over G_ε the parameter m can be as small as $2^{(1+\varepsilon)d}$. These results can be combined with Claim 4.13 to give corresponding improvements in the complexity of the encoding. Ex-

tensions of the iterated group product approach to efficiently encoding non-deterministic *branching programs* appear in [FKN94]. More efficient constructions which apply to a wider class of branching programs are given in [IK97,CFIK03].

4.2. Low-Degree Encodings

In this section we describe several general approaches for encoding functions by low-degree polynomials. Throughout this section we will consider polynomials over a finite field $\mathbb{F}$, where we will mainly be interested in the case of the binary field. However, it will sometimes be useful to work over a field $\mathbb{F}$ which is bigger than the input domain. For instance, we may want to encode a function $f : \{0,1\}^n \to \{0,1\}$ by a low-degree encoding over a non-binary field $\mathbb{F}$. This is captured by the following definition.

Definition 4.14 (Degree-d encoding) *Let $\mathbb{F}$ be a finite field, let $H \subseteq \mathbb{F}$, and let $f : H^n \to Y$. We say that $\hat{f} : \mathbb{F}^n \times \mathbb{F}^m \to \mathbb{F}^s$ is a* degree-d encoding of f *if there exist polynomials $p_1, \ldots, p_s \in \mathbb{F}[x_1, \ldots, x_n, r_1, \ldots, r_m]$, each of total degree at most d, such that*

- *$\hat{f}(x, r) = (p_1(x, r), \ldots, p_s(x, r))$ for all $x \in \mathbb{F}^n$ and $r \in \mathbb{F}^m$, and*
- *$\hat{f}(x, r)$, restricted to inputs x in H^n, is a randomized encoding of f according to Definition 3.1.*

When H is a strict subset of $\mathbb{F}$, the encoding provides no correctness or privacy guarantees for $x \notin H^n$. In this case, some applications of low-degree encodings (such as secure computation in the presence of active corruptions) need to ensure that the inputs are indeed taken from H.

We start with a simple example, showing the existence of a degree-2, statistically correct encoding of the boolean OR function over any field $\mathbb{F}$. This encoding was implicitly used for low-degree approximations of constant-depth circuits in the works of Smolensky and Razborov [Smo87,Raz89].

Example 4.15 (Degree-2 encoding for disjunction) Let $f_{\mathrm{OR}} : \{0,1\}^n \to \{0,1\}$ be the disjunction function $f_{\mathrm{OR}}(x_1, \ldots, x_n) = x_1 \vee x_2 \vee \ldots \vee x_n$. For a finite field $\mathbb{F}$, define $\hat{f}_{\mathrm{OR}} : \mathbb{F}^n \times \mathbb{F}^n \to \mathbb{F}$ by $\hat{f}_{\mathrm{OR}}(x, r) = \sum_{i=1}^{n} r_i x_i$. Then, $\hat{f}_{\mathrm{OR}}$ is a perfectly private, $(1/|\mathbb{F}|)$-correct degree-2 encoding of $\hat{f}_{\mathrm{OR}}$. Indeed, if $f(x) = 0$ then the output $\hat{y} = \hat{f}_{\mathrm{OR}}(x, r)$ is identically 0, and if $f(x) = 1$ then $\hat{y}$ is uniformly distributed over $\mathbb{F}$. Thus, a decoder Dec which outputs 0 if $\hat{y} = 0$ and outputs 1 otherwise errs with at most $1/|\mathbb{F}|$ probability. We note that concatenating s independent outputs of $\hat{f}_{\mathrm{OR}}$ (using ns random field elements) yields a perfectly private, $|\mathbb{F}|^{-s}$-correct degree-2 encoding for f_{OR}.

A corollary of this example is that f_{OR} can be computed by a 2-round protocol with security against $t < n/2$ active corruptions. It is open whether such protocols exist for general functions.

Example 4.15 can be generalized to a degree-2 encoding of any function which tests the membership of x in some affine subspace of $\mathbb{F}^n$. Other functions which (trivially) admit a degree-2 encoding are those that have degree 2 over $\mathbb{F}$. It turns out that no other functions $f : \{0,1\}^n \to \{0,1\}$ admit degree-2 encodings with perfect privacy even if

any correctness error $\delta < 1/2$ is allowed [IK00]. It is open whether the same holds for statistically private encodings, even when perfect correctness is required.

In the following we give several constructions of degree-3 encodings for arbitrary functions f.

4.2.1. Slightly correct degree-3 encodings

We first show that by relaxing the correctness requirement to allow an arbitrary nontrivial correctness error $\delta < 1/2$ (but still insisting on perfect privacy), we can get very efficient degree-3 encodings whose complexity is linear in the circuit size.

The idea is to use a standard reduction from the evaluation of a circuit C on input x to the satisfiability of a system of quadratic equations $q_i(x, y) = 0$ in the inputs x and additional auxiliary inputs y corresponding to gates of the circuit. The system should have the property that if $C(x) \neq 0$ then there is no y such that (x, y) satisfy all equations, and if $C(x) = 0$ then there is exactly *one* such y. The degree-3 encoding of C guesses a solution y at random and "verifies" it by taking a random linear combination of all $q_i(x, y)$, as in Example 4.15. We formulate below an arithmetic version that works over any finite field $\mathbb{F}$. An encoding for standard boolean circuits can be obtained by taking $\mathbb{F}$ to be the binary field.

Construction 4.16 (Slightly correct degree-3 encoding from circuits) *Let $C : \mathbb{F}^n \to \mathbb{F}$ be an arithmetic circuit of size s over a finite field $\mathbb{F}$. Let $f_C : \mathbb{F}^n \to \{0, 1\}$ be the function defined by*

$$f_C(x) = \begin{cases} 0, & C(x) = 0, \\ 1, & C(x) \neq 0. \end{cases}$$

Denote the gates of C by $y_1, \ldots, y_s$, where $y_1, \ldots, y_n$ are the inputs and y_s is the output gate. Define degree-2 polynomials $q_i(x_1, \ldots, x_n, y_1, \ldots, y_s)$, $0 \leq i \leq s$, as follows:

- $q_0 = y_s$,
- *For $i = 1, \ldots, n$, $q_i = y_i - x_i$,*
- *For every addition gate y_j with inputs y_a, y_b, let $q_j = y_j - (y_a + y_b)$,*
- *For every multiplication gate y_j with inputs y_a, y_b, let $q_j = y_j - y_a y_b$.*

Define the randomized encoding $\hat{f}_C : \mathbb{F}^n \times \mathbb{F}^{2s+1} \to \mathbb{F}$ by

$$\hat{f}_C(x, (y_1, \ldots, y_s, r_0, \ldots, r_s)) = \sum_{i=0}^{s} r_i q_i(x_1, \ldots, x_n, y_1, \ldots, y_s).$$

Claim 4.17 *The function $\hat{f}_C$ from Construction 4.16 is a perfectly private, δ-correct degree-3 encoding of f_C for some $\delta < 1/2$.*

Proof: Note that for any input x, if $C(x) = 0$ then there is exactly one y which satisfies all equations $q_i(x, y) = 0$, namely y which contains the outputs of all gates in the evaluation of C on x. If $C(x) \neq 0$, on the other hand, then no such y exists. We can now analyze the output distributions in both cases. If $C(x) \neq 0$ then conditioned on any choice of y, the linear combination $\sum r_i q_i$ will be uniformly random over $\mathbb{F}$ (since at

least one q_i takes a nonzero value). Thus, in this case the output of $\hat{f}_C$ is uniform over $\mathbb{F}$. On the other hand, if $C(x) = 0$ then there is a single choice of y conditioned on which the output of $\hat{f}_C$ will be identically zero, and conditioned on all other choices of y the output will random as before. Thus, in this case the output will be slightly biased towards 0 (where the distribution will be the same for all x such that $C(x) = 0$). Perfect privacy follows from the fact that the output distribution of $\hat{f}_C(x, r)$ depends only on $f_C(x)$. The ability to (probabilistically) decode with a nontrivial correctness error $\delta < 1/2$ follows from the fact that the two output distributions are distinct. ■

As in Example 4.15, one could amplify the correctness of this construction via repetition. However, using this approach to get a small error δ, say $\delta < 1/3$, will make the output size of the encoding exponential in the circuit size.

4.2.2. Degree-3 encodings over huge fields

The previous construction worked over any field, but required exponentially many outputs to achieve a small correctness error. Here we show the existence of a degree-3 encoding with a *single* output and a small correctness error for any function $f : \{0,1\}^n \to \{0,1\}$. The price is that the field size will generally be doubly exponential in n. The idea (adapted from [FKN94]) is to find a large enough prime p such that the sequence of quadratic characters modulo p contains the truth table of f as a subsequence. The encoding $\hat{f}$ is computed by first mapping the bits of x to the corresponding residue a_x, and then encoding the quadratic character of a_x by multiplying it with the square of a random field element.

For a prime p and $0 \leq a < p$ we let $QR_p(a)$ denote the quadratic character of a modulo p, namely $QR_p(a) = a^{(p-1)/2} \mod p$. Note that $QR_p(a) \in \{-1, 1\}$ for $a \neq 0$. We rely on the following fact from number theory.

Fact 4.18 (cf. [Per92]) *For any sequence $(t_0, t_1, \ldots, t_{N-1}) \in \{-1,1\}^N$ there exists a prime $p \in 2^{O(N)}$ and $0 < a_0 \leq p - N$ such that $QR_p(a_0 + i) = t_i$ for $i = 0, \ldots, N-1$.*

Construction 4.19 (Degree-3 encoding over huge fields) *For $f : \{0,1\}^n \to \{0,1\}$ and $x \in \{0,1\}^n$, define $i(x) = \sum_{j=1}^{n} 2^{j-1} x_j$ and $t_{i(x)} = (-1)^{f(x)}$. Let p, a_0 be as promised by Fact 4.18 for the sequence $(t_0, \ldots, t_{2^n-1})$. Define the randomized encoding $\hat{f} : \mathbb{F}_p^n \times \mathbb{F}_p \to \mathbb{F}_p$ by*

$$\hat{f}(x, r) = r^2 \cdot \left(a_0 + \sum_{j=1}^{n} 2^{j-1} x_j \right)$$

Claim 4.20 *The function $\hat{f}$ from Construction 4.19 is a perfectly private, $(1/p)$-correct degree-3 encoding of f over $\mathbb{F}_p$.*

Proof: The output $\hat{y} = \hat{f}(x, r)$ is distributed as $r^2 a$ for some $a = a(x)$ such that $QR_p(a) = (-1)^{f(x)}$. Thus, $\hat{y} = 0$ with probability $1/p$, and otherwise it is uniformly distributed over the $(p-1)/2$ elements $a' \in \mathbb{F}_p$ such that $QR_p(a') = QR_p(a)$. Since this distribution is determined by $f(x)$, we get perfect privacy. The decoder can output 0 if $QR_p(\hat{y}) = 1$ and output 1 if $QR_p(\hat{y}) \in \{-1, 0\}$. This decoder can err only when $\hat{y} = 0$. ■

4.2.3. Degree-3 encodings from circuits

The previous constructions we described are not perfectly correct and are not efficient in the input length even for simple functions. Using suitable variants of Yao's garbled circuit construction [IK02,AIK06a], it is possible to avoid these limitations and efficiently transform any boolean formula (resp., boolean circuit) into a perfectly private and correct (resp., computationally private and perfectly correct) degree-3 encoding over the binary field.

Below we describe a modification of the composition-based construction from Section 4.1.2 which yields perfectly correct degree-3 encodings over an arbitrary field $\mathbb{F}$ with similar efficiency features. Recall that the affinization gadget from Claim 4.10 is an encoding $\hat{f}_{\mathrm{AG}}$ of the function $f_{\mathrm{AG}}(x_1, x_2, x_3) = x_1x_2 + x_3$ which has degree 1 in the inputs x_i and total degree 2 in x_i, r_j. (In fact, for our purposes here we can use the simpler form $\hat{f}'_{\mathrm{AG}}$ of this gadget appearing in the proof of Claim 4.10.)

The composition step of Lemma 4.11 uses this gadget to encode an entry of the form $t = a(r)(x'_i x'_j) + b(r)$. (We consider here the case of multiplication; the case of addition is simpler.) This is done by parsing it as $(a(r)x'_i) \cdot x'_j + b(r)$ and using the affinization gadget with $x_1 = a(r)x'_i$, $x_2 = x'_j$ and $x_3 = b(r)$. The resulting encoding now has entries of the form $t' = r_h \cdot a(r)x'_j + c(r)$, where $c(r)$ has degree 2 in the randomness. The entry t' has degree 1 in the x' variables (this is sufficient for maintaining the DARE property) but its degree in the random inputs r_i may be bigger by 1 compared to t. The next substitution step replaces x'_j by a product of new inputs and repeats the above process, which further increases the degree in the randomness.

To eliminate this growth of degree, it suffices to apply the affinization gadget again to t' before the composition step. That is, t' is parsed as $(r_h a(r)) \cdot x'_j + c(r)$ and then encoded via the affinization gadget. This modification can be used to maintain the invariant that the degree in the x variables is 1 and the total degree in both x and r is at most 3.

The above approach yields a polynomial-size degree-3 encodings for functions in (boolean and arithmetic) NC^1. Settling for computational privacy and assuming the existence of a pseudorandom generator in NC^1 (equivalently, a one-way function in NC^1 [HILL99,HRV10]), this can be extended to all polynomial-time computable functions. The idea is to first construct an NC^1 encoding of f with oracle access to the PRG (this can be done directly via Yao's construction or via the approach of Section 4.1.2). Substituting the NC^1 PRG implementation, we get an encoding for f in NC^1, which can then be encoded again into a degree-3 encoding. We refer the reader to [AIK06a] for more details.

4.2.4. Degree-3 encodings from branching programs

In this section we describe a perfect degree-3 encoding from [IK02] whose complexity is quadratic in the size of a *branching program* computing f. This construction efficiently applies to complexity classes such as $\oplus\mathrm{L}/poly$ (a mod-2 variant of non-deterministic logspace) which are believed to strictly contain NC^1.

The high level idea of the encoding we describe next is to reduce the evaluation of a branching program to computing the *determinant* of a matrix M whose entries are degree-1 polynomials in the inputs. Relying on the special form that M has, we can encode its determinant via a matrix product $R_1 M R_2$ where R_1 and R_2 are random matrices whose entries contain either random inputs or constants.

We start by defining the notion of (arithmetic) branching programs to which the encoding applies.

Definition 4.21 *(**Branching program over** $\mathbb{F}$**)** *A branching program (BP) over $\mathbb{F}$ is defined by a quadruple $BP = (G, \phi, s, t)$, where $G = (V, E)$ is a directed acyclic graph, ϕ is an edge labeling function assigning each edge a degree-1 polynomial in a single input variable x_i, and s, t are two special vertices. The* size *of BP is the number of vertices in G. Each input assignment $x = (x_1, \ldots, x_n) \in \mathbb{F}^n$ induces an assignment G_x of a value from $\mathbb{F}$ to each $e \in E$. The output $BP(x)$ is defined as the sum of the weights of all directed paths from s to t in G_x, where the weight of a path is the product of the values of its edges.*

Branching programs are quite powerful. In particular, any boolean formula (resp., arithmetic formula over $\mathbb{F}$) has a BP over $\mathbb{F}_2$ (resp., over $\mathbb{F}$) of the same size, whereas it is conjectured that BPs cannot be efficiently simulated by formulas. However, as polynomial-size BPs capture different variants of *log-space* computation, it seems unlikely that all polynomial-time computable functions admit polynomial-size BPs.

The construction. The following description is taken almost verbatim from [IK02, AIK06b]. Let $BP = (G, \phi, s, t)$ be a BP of size ℓ over $\mathbb{F}$, computing a function $f : \mathbb{F}^n \to \mathbb{F}$. Fix some topological ordering of the vertices of G, where the source vertex s is labeled 1 and the terminal vertex t is labeled ℓ. For any input x, let A_x be the $\ell \times \ell$ matrix over $\mathbb{F}$ whose (i, j) entry contains the value assigned by ϕ to the edge (i, j) (or 0 if there is no such edge). Define $L(x)$ as the submatrix of $A_x - I$ obtained by deleting column s and row t (i.e., the first column and the last row). Note that each entry of $L(x)$ has degree (at most) 1 in the inputs x; moreover, $L(x)$ contains the constant -1 in each entry of its second diagonal (the one below the main diagonal) and the constant 0 below this diagonal.

Fact 4.22 ([IK02]) $f(x) = \det(L(x))$.

Let $r^{(1)}$ and $r^{(2)}$ be vectors over $\mathbb{F}$ of length $\binom{\ell-1}{2}$ and $\ell - 2$ respectively. Let $R_1(r^{(1)})$ be an $(\ell - 1) \times (\ell - 1)$ matrix with 1's on the main diagonal, 0's below it, and the elements of $r^{(1)}$ in the remaining $\binom{\ell-1}{2}$ entries above the diagonal (a unique element of $r^{(1)}$ is assigned to each matrix entry). Let $R_2(r^{(2)})$ be an $(\ell - 1) \times (\ell - 1)$ matrix with 1's on the main diagonal, $r^{(2)}$'s elements in the rightmost column, and 0's in the remaining entries.

Fact 4.23 ([IK02]) *Let M, M' be $(\ell - 1) \times (\ell - 1)$ matrices that contain the constant -1 in each entry of their second diagonal and the constant 0 below this diagonal. Then, $\det(M) = \det(M')$ if and only if there exist $r^{(1)}$ and $r^{(2)}$ such that $R_1(r^{(1)}) M R_2(r^{(2)}) = M'$.*

Claim 4.24 *Let BP and f be as above. Define a degree-3 function $\hat{f}(x, (r^{(1)}, r^{(2)}))$ whose outputs contain the $\binom{\ell}{2}$ entries on or above the main diagonal of the matrix $R_1(r^{(1)}) L(x) R_2(r^{(2)})$. Then, $\hat{f}$ is a degree-3 encoding of f.*

Proof: We start by describing the simulator and the decoder and then prove their correctness. Given an output $\hat{y}$ of $\hat{f}$, representing a matrix M, the decoder Dec simply

outputs $\det(M)$. The simulator Sim, on input $y \in \mathbb{F}$, outputs the $w = \binom{\ell}{2}$ entries on and above the main diagonal of the matrix $R_1(r^{(1)})H_y R_2(r^{(2)})$, where $r^{(1)}$, $r^{(2)}$ are randomly chosen, and H_y is the $(\ell-1)\times(\ell-1)$ matrix that contains (-1)'s in its second diagonal, y in its top-right entry, and 0's elsewhere.

By Facts 4.22 and 4.23, for every $x \in \mathbb{F}^n$ the support sets of $\hat{f}(x,r)$ and $\mathrm{Sim}(f(x))$ are equal. Specifically, these support sets include all vectors in $\mathbb{F}^w$ representing matrices with determinant $f(x)$. Since the support sets of $\mathrm{Sim}(y)$ for distinct $y \in \mathbb{F}$ form a disjoint partition of the entire output space $\mathbb{F}^w$ (by Fact 4.23) and since Sim uses $m = w - 1$ random field elements, it follows that the support size of $\mathrm{Sim}(y)$ is $|\mathbb{F}|^m$, for each $y \in \mathbb{F}$. Since both the simulator and the encoding use m random field elements, it follows that both of the distributions $\hat{f}(x,r)$ and $\mathrm{Sim}(f(x))$ are uniform over their support and are therefore identical. ■

This yields the following theorem.

Theorem 4.25 *Let BP be a branching program of size ℓ over a finite field $\mathbb{F}$ computing the function $f : \mathbb{F}^n \to \mathbb{F}$. Then f admits a degree-3 encoding over $\mathbb{F}$ with output length $O(\ell^2)$.*

We note that Theorem 4.25 in fact holds over general finite *rings* [CFIK03]. Moreover, the degree-3 encoding has degree 1 in the inputs x and can be made decomposable via the transformation described in the next section.

4.3. Local Encodings

In this section we discuss the existence of randomized encodings with small *output locality* and their relations with previous notions of simplicity. We will focus here on the case of functions $f : \{0,1\}^n \to \{0,1\}^l$ over the binary alphabet. Recall that an encoding $\hat{f}(x,r)$ of f has output locality d (or locality d for short) if each output bit of $\hat{f}$ depends on at most d bits of its input (x,r).

Note that if $\hat{f}$ has locality d then it also has degree (at most) d over the binary field $\mathbb{F}_2$. We show the following converse: if $\hat{f}$ has degree d over $\mathbb{F}_2$, then it can be efficiently converted into an encoding $\hat{f}'$ of f with locality $d+1$ (and degree d). In particular, the degree-3 encodings from the previous section can be efficiently converted into encodings with locality 4.

Reducing the locality. The idea for reducing the locality is to represent a degree-d polynomial over $\mathbb{F}_2$ as a sum of monomials, each having locality d, and randomize this sum by composing it with a variant of the decomposable encoding for group products from Claim 4.13. (A direct use of Claim 4.13 over the binary group gives a $(d+2)$-local encoding instead of the $(d+1)$-local encoding we obtain here.)

Construction 4.26 (Locality construction) *Let $f(x) = T_1(x) + \ldots + T_k(x)$, where summation is over an abelian group. The local encoding $\hat{f}$ is defined by:*
$\hat{f}(x,(r_1,\ldots,r_k,r'_1,\ldots,r'_{k-1})) \stackrel{def}{=} (T_1(x)-r_1, T_2(x)-r_2, \ldots, T_k(x)-r_k, r_1-r'_1, r'_1+r_2-r'_2, \ldots, r'_{k-2}+r_{k-1}-r'_{k-1}, r'_{k-1}+r_k)$.

For example, applying the locality construction to the polynomial $x_1x_2 + x_2x_3 + x_4$ results in the function $(x_1x_2 - r_1, x_2x_3 - r_2, x_4 - r_3, r_1 - r'_1, r'_1 + r_2 - r'_2, r'_2 + r_3)$.

Lemma 4.27 (Locality lemma [AIK06b]) *Let f and $\hat{f}$ be as in Construction 4.26. Then, $\hat{f}$ is a randomized encoding of f. In particular, if f is a degree-d polynomial over $\mathbb{F}_2$ written as the sum of monomials, then $\hat{f}$ is an encoding of f with degree d and locality* $\max(d+1, 3)$.

Note that the encoding $\hat{f}$ obtained via the locality construction is also *decomposable* whenever f has degree 1 in the input x. Moreover, any degree-3 encoding can be efficiently converted into one that has degree 1 in x using the encodings from the previous section. This yields the following strong transformation from low-degree encodings to encodings that simultaneously satisfy all of our efficiency features.

Theorem 4.28 *Suppose $f : \{0,1\}^n \to \{0,1\}^l$ has a degree-3 encoding $\hat{f} : \mathbb{F}_2^n \times \mathbb{F}_2^m \to \mathbb{F}_2^s$. Then f admits a decomposable degree-3 encoding $\hat{f}' : \mathbb{F}_2^n \times \mathbb{F}_2^{m'} \to \mathbb{F}_2^{s'}$ with locality 4, where $m', s' \in \mathrm{poly}(n, m, s)$.*

To summarize the known relations between the different notions of simplicity, any low-degree encoding can be converted into an encoding that has constant output locality with a polynomial overhead, and vice versa. Both can be converted into a decomposable encoding with polynomial overhead.

5. Open Questions

We conclude by collecting some open questions about the existence and complexity of "simple" randomized encodings. Unless noted otherwise, we will assume the underlying alphabet or field to be binary. The existence of a "statistical encoding" refers to an ε-correct, ε-private encoding for an arbitrarily small $\varepsilon > 0$. When the notion of simplicity is unspecified, the question is open for all notions of simplicity considered in this survey.

Degree. Does every finite function (equivalently, the function $x_1x_2x_3+x_4$) admit a statistical *degree-2* encoding? The answer is negative for perfectly private encodings [IK00]. A positive answer would imply 2-round secure k-party protocols which tolerate $t < k/2$ corrupted parties.

Locality. Does every finite function (equivalently, the function $x_1x_2x_3 + x_4$) admit a statistical encoding with *output locality 3*? The question is open even with perfect correctness and privacy. A positive answer is implied by a positive answer to the previous question and would imply cryptography with (optimal) output locality of 3 under a broad set of assumptions.

Complexity of statistical encodings. Does every polynomial-time computable function admit a polynomial-time computable (or even polynomial-size) simple encoding with perfect or statistical privacy? A positive answer would imply efficient constant-round statistically secure protocols for all polynomial-time computable functions, settling a long-standing open question in information-theoretic cryptography [BMR90]. As an alternative to resolving this (longstanding) question, one could try to relate it to open problems from other domains. There is also significant room for improving the asymptotic complexity of statistical encodings for natural representation models such as formulas and branching programs as well as specific functions of interest.

Complexity of computationally private encodings. Yao's construction yields a simple polynomial-size encoding for any polynomial-time computable function. In the case of decomposable encoding, this can be based on the existence of any one-way function whereas local or low-degree encodings rely on a one-way function in NC^1. Can the latter assumption be relaxed to the existence of an arbitrary one-way function? This would suffice for showing that an arbitrary one-way function implies a one-way function in NC^0. Can the encoding size be made smaller than the computation time of the function being encoded? A more modest goal is to encode any boolean circuit of size s by a decomposable encoding of size $O(s) + \mathrm{poly}(k)$ (where k is a security parameter), improving on the $O(ks)$ size of Yao's construction. A final open question is to obtain efficient decomposable encodings for arithmetic circuits over a finite field $\mathbb{F}$ in which both the encoder and the decoder make a black-box use of $\mathbb{F}$. A positive result would give general constant-round secure computation protocols in the arithmetic black-box model.

Acknowledgments. I would like to thank Benny Applebaum and Eyal Kushilevitz for many fruitful discussions and collaborations on which this survey is based, and Manoj Prabhakaran and Amit Sahai for persistent encouragement to write this survey.

References

[AIK06a] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Computationally private randomizing polynomials and their applications. *Computational Complexity*, 15(2):115–162, 2006. Preliminary version in CCC 2005.

[AIK06b] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. *SIAM J. Comput.*, 36(4):845–888, 2006. Preliminary version in FOCS 2004.

[AIK09] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography with constant input locality. *J. Cryptology*, 22(4):429–469, 2009. Preliminary version in Crypo 2007.

[AIK10a] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography by cellular automata or how fast can complexity emerge in nature? In *ICS*, pages 1–19, 2010.

[AIK10b] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *ICALP (1)*, pages 152–163, 2010.

[AIK11] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. In *FOCS*, pages 120–129, 2011. Full version in ECCC 19: 58 (2012).

[AIKW12] Benny Applebaum, Yuval Ishai, Eyal Kushilevitz, and Brent Waters. Encoding functions with constant online rate, or: Garbling circuits with short input keys. Manuscript, 2012.

[AIR01] William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In *EUROCRYPT*, pages 119–135, 2001.

[App11a] Benny Applebaum. Key-dependent message security: Generic amplification and completeness theorems. In *EUROCRYPT*, 2011.

[App11b] Benny Applebaum. Randomly encoding functions: A new cryptographic paradigm (invited talk). In *ICITS*, pages 25–31, 2011.

[Bar86] David A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. In *Proc. 18th STOC*, pages 1–5, 1986.

[BC92] Michael Ben-Or and Richard Cleve. Computing algebraic formulas using a constant number of registers. *SIAM J. Comput.*, 21(1):54–58, 1992.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. of 20th STOC*, pages 1–10, 1988.

[BHHI10] Boaz Barak, Iftach Haitner, Dennis Hofheinz, and Yuval Ishai. Bounded key-dependent message security. In *EUROCRYPT*, pages 423–444, 2010.

[BHR12a] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. *IACR Cryptology ePrint Archive*, 2012:564, 2012. To appear in *Asiacrypt* 2012.

[BHR12b] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. *IACR Cryptology ePrint Archive*, 2012:265, 2012. To appear in *ACM CCS* 2012.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *STOC*, pages 503–513, 1990.

[Bre74] Richard P. Brent. The parallel evaluation of general arithmetic expressions. *J. ACM*, 21(2):201–206, 1974.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Electronic Colloquium on Computational Complexity (ECCC) TR01-016, 2001. Preliminary version in FOCS 2001.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *Proc. of 20th STOC*, pages 11–19, 1988.

[CCKM00] Christian Cachin, Jan Camenisch, Joe Kilian, and Joy Müller. One-round secure computation and secure autonomous mobile agents. In *ICALP*, pages 512–523, 2000.

[CFGN96] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *STOC*, pages 639–648, 1996.

[CFIK03] Ronald Cramer, Serge Fehr, Yuval Ishai, and Eyal Kushilevitz. Efficient multi-party computation over rings. In *EUROCRYPT*, pages 596–613, 2003.

[Cle90] Richard Cleve. Towards optimal simulations of formulas by bounded-width programs. In *STOC*, pages 271–277, 1990.

[DGRV11] Zeev Dvir, Dan Gutfreund, Guy N. Rothblum, and Salil P. Vadhan. On approximating the entropy of polynomial mappings. In *ICS*, pages 460–475, 2011.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, pages 643–662, 2012.

[EGL85] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.

[FKN94] Uri Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation. In *STOC*, 1994.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[GGH+07] Shafi Goldwasser, Dan Gutfreund, Alexander Healy, Tali Kaufman, and Guy N. Rothblum. Verifying and decoding in constant depth. In *STOC*, pages 440–449, 2007.

[GGH+08] Shafi Goldwasser, Dan Gutfreund, Alexander Healy, Tali Kaufman, and Guy N. Rothblum. A (de)constructive approach to program checking. In *STOC*, pages 143–152, 2008.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, 2010.

[GHV10] Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. *i*-hop homomorphic encryption and rerandomizable Yao circuits. In *CRYPTO*, pages 155–172, 2010.

[GIS+10] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In *TCC*, pages 308–326, 2010.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. One-time programs. In *CRYPTO*, pages 39–56, 2008.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play ANY mental game. In *STOC*, pages 218–229, 1987.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications*. Cambridge University Press, 2004.

[GS12] Sanjam Garg and Amit Sahai. Adaptively secure multi-party computation with dishonest majority. In *CRYPTO*, pages 105–123, 2012.

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In *CRYPTO*, pages 162–179, 2012.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.

[HK07] Omer Horvitz and Jonathan Katz. Universally-composable two-party computation in two rounds. In *CRYPTO*, pages 111–129, 2007.

[HRV10] Iftach Haitner, Omer Reingold, and Salil P. Vadhan. Efficiency improvements in constructing pseudorandom generators from one-way functions. In *STOC*, pages 437–446, 2010.

[IK97] Yuval Ishai and Eyal Kushilevitz. Private simultaneous messages protocols with applications. In *ISTCS*, pages 174–184, 1997.

[IK00] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *FOCS*, pages 294–304, 2000.

[IK02] Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *ICALP*, pages 244–256, 2002.

[IKO+11] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Manoj Prabhakaran, and Amit Sahai. Efficient non-interactive secure computation. In *EUROCRYPT*, pages 406–425, 2011.

[IKP10] Yuval Ishai, Eyal Kushilevitz, and Anat Paskin. Secure multiparty computation with minimal interaction. In *CRYPTO*, pages 577–594, 2010.

[IKP12] Yuval Ishai, Eyal Kushilevitz, and Anat Paskin-Cherniavsky. From randomizing polynomials to parallel algorithms. In *ITCS*, pages 76–89, 2012.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In *CRYPTO*, pages 572–591, 2008.

[IPS09] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Secure arithmetic computation with no honest majority. In *TCC*, pages 294–314, 2009.

[Kil88] J. Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31, 1988.

[Kol05] Vladimir Kolesnikov. Gate evaluation secret sharing and secure one-round two-party computation. In *ASIACRYPT*, pages 136–155, 2005.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of Yao's protocol for secure two-party computation. *J. Cryptology*, 22(2):161–188, 2009.

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *SODA*, pages 448–457, 2001.

[NP06] Moni Naor and Benny Pinkas. Oblivious polynomial evaluation. *SIAM J. Comput.*, 35(5):1254–1281, 2006. Preliminary version in STOC 1999.

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *ACM Conference on Electronic Commerce*, pages 129–139, 1999.

[Per92] R. Perlata. On the distribution of quadratic residues and nonresidues modulo a prime number. *Mathematics of Computation*, 58:433–440, 1992.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In *CRYPTO*, pages 554–571, 2008.

[Rab81] M. Rabin. How to exchange secrets by oblivious transfer. Technical Report TR-81, Harvard Aiken Computation Laboratory, 1981.

[RAD78] Ronald L. Rivest, Leonard M. Adleman, and Michael Dertouzos. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[Raz89] Alexander A. Razborov. On the method of approximations. In *STOC*, pages 167–176, 1989.

[Smo87] Roman Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *STOC*, pages 77–82, 1987.

[SS10] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In *ACM Conference on Computer and Communications Security*, pages 463–472, 2010.

[SYY99] Tomas Sander, Adam Young, and Moti Yung. Non-interactive cryptocomputing for NC^1. In *FOCS*, pages 554–567, 1999.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *FOCS*, pages 162–167, 1986.