Electronic Colloquium on Computational Complexity, Report No. 67 (2017)

Garbled Circuits as Randomized Encodings of Functions: a Primer[*]

Benny Applebaum[†]

Abstract

Yao's garbled circuit (GC) construction is a central cryptographic tool with numerous applications. In this tutorial, we study garbled circuits from a foundational point of view under the framework of *randomized encoding* (RE) of Functions. We review old and new constructions of REs, present some lower-bounds, and describe some applications. We will also discuss new directions and open problems in the foundations of REs.

1 Introduction

Garbled circuits were introduced by Yao (in oral presentations of [86]) as a two-party protocol for secure function evaluation. At the heart of Yao's protocol stands a non interactive *garbling* technique that became a central tool in the design of constant-round secure computation protocols [86, 21, 43, 80, 66, 75]. Over the years, the garbled circuit technique has found a diverse range of other applications to problems such as computing on encrypted data [84, 35], parallel cryptography [10, 11], verifiable computation [46, 14], software protection [55, 58, 25], functional encryption [83, 56], key-dependent message security [19, 2], code obfuscation [5, 38], and others. Correspondingly, it is currently evident that the *garbling technique* should be treated as a standalone abstract object.

The first such abstraction was introduced by Ishai and Kushilevitz [66, 67] under the algebraic framework of *randomizing polynomials*, and was later extended and abstracted under the terminology of *randomized encoding of functions* [10, 11]. In this framework we view garbled circuits as an encoding (or encryption) of a computation. Roughly speaking, we would like to represent a function $f(x)$ by a randomized function $\hat{f}(x;r)$ such that the following hold:

- (Correctness) For every fixed input x and a uniformly random choice of r, the output distribution $\hat{f}(x;r)$ forms a "randomized encoding" of $f(x)$, from which $f(x)$ can be efficiently decoded. Namely, there exists an efficient decoding algorithm Dec, referred to as a *decoder*, such that for every x and r, it holds that $\mathsf{Dec}(\hat{f}(x;r)) = f(x)$. In particular, this means that, if $f(x) \neq f(x')$, then the random variables $\hat{f}(x;r)$ and $\hat{f}(x';r')$, induced by a uniform choice of r and r', should have disjoint supports. (We will later discuss relaxations in which the supports are "almost disjoint" and decoding succeeds for most r's.)

[*] This paper appears in a book of surveys in honor of Oded Goldreich's 60th birthday, and subsumes the short survey [3].

[†] School of Electrical Engineering, Tel-Aviv University, bennyap@post.tau.ac.il. Supported by the European Union's Horizon 2020 Programme (ERC-StG-2014-2020) under grant agreement no. 639813 ERC-CLC, ISF grant 1155/11, and the Check Point Institute for Information Security..

ISSN 1433-8092

- (Privacy) The distribution of this randomized encoding depends only on the encoded value $f(x)$ and essentially does not reveal further information on x. Namely, there exists an efficient randomized algorithm Sim, referred to as a *simulator*, which, given $f(x)$, samples the distribution $\hat{f}(x;r)$ induced by a random choice of r. Ideally, the simulator's output should be identically distributed to the encoding, but we will also consider variants in which the two distributions are only statistically or computationally close. Note that the privacy requirement implies that, if $f(x) = f(x')$, then the random variables $\hat{f}(x;r)$ and $\hat{f}(x';r')$ are close to each other, or even identical when the simulator is perfect.
- (Efficiency/simplicity) To be useful, the encoding $\hat{f}$ should be "simpler" or more "efficient" than the original function with respect to some measure of complexity.

Observe that privacy can be trivially satisfied by the constant function $\hat{f}(x) = 0$ whereas correctness can be trivially satisfied by the identity function $\hat{f}(x) = x$. However, the combination of privacy and correctness forms a natural relaxation of the usual notion of computing. The last, "efficiency" requirement ensures that this relaxation is nontrivial. As we will later see, the use of randomness is necessary for achieving all three goals. For the sake of concreteness consider the following examples:

Example 1.1 (Encoding modular squaring)**.** *Let N be a (public) integer and consider the modular squaring function $f(x) = x^2 \bmod N$. Then, f can be encoded by the function $\hat{f}(x;r) = x^2 + r \cdot N$, where computation is over the integers and r is a random integer in $[0, B]$ for some large integer $B \gg N$. To see that the encoding is correct consider the decoder Dec which, given an integer $\hat{y}$ (supposedly in the image of $\hat{f}$), outputs $\hat{y} \bmod N$. Clearly, $\mathsf{Dec}(\hat{f}(x;r)) = f(x)$ for every x and r. Privacy holds since any pair of inputs x and x' for which $f(x) = f(x') = y$ are mapped to a pair of distributions $\hat{f}(x;r)$ and $\hat{f}(x';r)$ which are (N/B)-statistically close. In particular, given $y = f(x)$, the simulator Sim outputs $y + rN$ for a uniformly chosen $r \overset{R}{\leftarrow} [0, B]$. It is not hard to show that the resulting distribution D_y is (N/B)-statistically close to $\hat{f}(x;r)$. Finally, the encoder $\hat{f}$ avoids modular reduction, which in some cases, may be considered an expensive operation.*

Let us briefly explain how Yao's garbled circuit fits into the randomized encoding framework. (Readers who are not familiar with Yao's construction may safely skip the following example; A full and modular description of this construction appears later in Section 3.)

Example 1.2 (Yao's construction as randomized encoding)**.** *Given a Boolean circuit $C : \{0,1\}^n \to \{0,1\}^m$ and secret randomness r, Yao's construction generates a "garbled circuit" $\hat{C}$ (which consists of several ciphertexts per logical gate of C), together with n pairs of "short" keys (K_i^0, K_i^1), such that for any (a priori unknown) input x, the garbled circuit $\hat{C}$ 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. Therefore, the mapping $g : (x;r) \mapsto (\hat{C}, K_x)$ forms a randomized encoding of the computation $C(x)$. The decoder Dec takes $(\hat{C}, K_x)$ as an input and recovers the value $C(x)$. (This is done by gradually decrypting, for each gate of C, a ciphertext of $\hat{C}$, starting with the input layer for which the keys K_x are used for decryption, and ending up in the output layer from which the actual output $C(x)$ is recovered.) The construction also admits a simulator which, given $y = C(x)$, samples a distribution $\mathsf{Sim}(y)$ which is computationally indistinguishable from the encoder's distribution $(\hat{C}, K_x)$. The encoding satisfies several notions of simplicity. In particular, each output of g depends on at most a single bit of x. Moreover, most of the output entries (the ones that correspond to the "garbled circuit" part $\hat{C}$) depend only on the randomness r, and so they can be computed in an "offline phase" before the* *actual input x is known. As we will later see, these efficiency properties are extremely useful for several applications.*

Note that, in the above examples, the saving in efficiency of the encoder has some cost: the decoder itself performs expensive operations (i.e., modular reduction in Example 1.1, and, in Example 1.2 an evaluation of a garbled version of the circuit C). This is inherent, as the computation $f(x)$ can be written as $\mathsf{Dec}(\hat{f}(x;r))$ and so we do not expect to simultaneously reduce the complexity of the encoder and decoder, since this would allow us to "speed up" the computation of f. Nevertheless, the ability to decompose the computation into an easy part ($\hat{f}$) and complicated part (Dec), while preserving privacy, can be useful in many cases, as demonstrated by the following example:

Example 1.3 (Simple usage scenario)**.** *Imagine a scenario of sending a weak device U into the field to perform some expensive computation f on sensitive data x. The computation is too complex for U to quickly perform on its own, and since the input x is sensitive, U cannot just send the entire input out. Randomized encoding provides a noninteractive solution to this problem: U simply sends out a single (randomized) message $\hat{f}(x;r)$. The rest of the world can, at this point, recover $f(x)$ (by applying the decoder) and nothing else (due to the privacy property). In fact, one could* define *RE as a noninteractive solution to the above problem.*

Remark 1.4 (FHE vs. RE)**.** *It is instructive to compare the use of RE in the above scenario (Example 1.3) with a solution based on fully homomorphic encryption (FHE) [47]. Using FHE, the client can send an encryption $\mathsf{FHE}(x)$ to an outside server, which can generate, in turn, the value $\mathsf{FHE}(f(x))$. However, in order to publish $f(x)$, another round of interaction is needed, since U has to decrypt the output. More generally, the power of RE stems from the ability to* reveal *$f(x)$ to anyone (while hiding x).*

This tutorial. The abstract framework of randomized encoding (RE) is quite general and can be instantiated in several ways, including computational and information-theoretic variants and different forms of efficiency/simplicity requirements. In this tutorial, we use this framework to study garbled circuits from a foundational point of view. As we will see, the use of a general and abstract framework is beneficial, even if one is interested in concrete forms of REs (e.g., ones that correspond to "standard garbled circuits"). Our presentation emphasizes the properties of REs and the way that basic REs can be combined and manipulated to produce better ones. Naturally, this leaves several important aspects uncovered. Most notably, many useful direct information-theoretic randomization techniques are omitted. Such techniques are thoroughly covered in the survey of Ishai [64]. We also focus on asymptotic analysis and leave out the concrete efficiency of REs.

Organization. The rest of this paper is organized as follows: In Section 2, we formally define REs and present their basic properties. A general template for constructing randomized encodings appears in Section 3, together with basic feasibility results. Recent constructions of REs which provide better efficiency or stronger security are presented in Section 4. Some basic applications of REs are sketched in Section 5. We conclude in Section 6 with a summary and suggestions for further reading. A brief comparison of REs with the "garbling scheme" framework of Bellare et al. [26] appears in Appendix A.

Acknowledgements. This tutorial is written in honor of the 60th birthday of Oded Goldreich. As a student, I enthusiastically read Oded's introductory texts and was excited to encounter a combination of the technical with the conceptual (and sometimes even the philosophical). Later, I was fortunate to interact with Oded as an editor, mentor, and friend. In many aspects, my scientific view is rooted in Oded's insightful oral and written discussions. I also thank Yehuda Lindell for initiating this tutorial, and for his useful comments on this manuscript. Finally, I am grateful to Yuval Ishai and Eyal Kushilevitz for introducing me to the notion of randomized encoding and for fruitful and enjoyable collaborations.

2 Definitions and Basic Properties

Notation. For a positive integer $n \in \mathbb{N}$, let $[n]$ denote the set $\{1, \dots, n\}$. A function $\varepsilon : \mathbb{N} \to [0, 1]$ is negligible if it tends to zero faster than $1/n^c$ for every constant $c > 0$. The term "efficient" refers to probabilistic polynomial time. For a finite set (resp., probability distribution) X, we let $x \overset{R}{\leftarrow} X$ denote an element that is sampled uniformly at random from X (resp., according to the distribution X). We let U_n denote the uniform distribution over n-bit strings. The statistical distance between a pair of random variables X and Y over a finite range R is defined by $\frac{1}{2}\sum_{r\in R} |\Pr[X = r] - \Pr[Y = r]|$.

2.1 Syntax and Basic Definition

We begin with a formal definition of randomized encoding of functions. In the following let X, Y, Z, and R be finite sets (typically taken to be sets of fixed-length strings).

Definition 2.1 (Randomized encoding [10, 11]). *Let $f : X \to Y$ be a function. We say that a function $\hat{f} : X \times R \to Z$ is a δ-correct, (t, ε)-private* randomized encoding *of f if there exist a pair of randomized algorithms, decoder Dec and simulator Sim, for which the following hold:*

- *(δ-correctness) For any input $x \in X$,*

$$\Pr_{r \overset{R}{\leftarrow} R}[\mathsf{Dec}(\hat{f}(x; r)) \neq f(x)] \leq \delta.$$

- *((t, ε)-privacy) For any $x \in X$ and any circuit[1] $\mathcal{A}$ of size t*

$$\left|\Pr[\mathcal{A}(\mathsf{Sim}(f(x))) = 1] - \Pr_{r \overset{R}{\leftarrow} R}[\mathcal{A}(\hat{f}(x; r)) = 1]\right| \leq \varepsilon.$$

We refer to the second input of $\hat{f}$ as its random input*, and a use semicolon $(;)$ to separate deterministic inputs from random inputs. We will sometimes write $\hat{f}(x)$ to denote the random variable $\hat{f}(x; r)$ induced by sampling $r \overset{R}{\leftarrow} R$. We refer to* $\log |R|$ *and* $\log |Z|$ *as the* randomness complexity *and the* communication complexity *of $\hat{f}$, respectively. By default, we measure the* computational complexity *of $\hat{f}$ by its circuit size.*

[1] For simplicity, throughout the paper we model adversaries as non-uniform circuits. However, all the results presented here also hold in a uniform model where adversaries are modeled as probabilistic polynomial-time Turing machines. A uniform treatment of REs can be found in [11, 15].

Infinite functions and collections. Definition 2.1 naturally extends to infinite functions $f : \{0,1\}^* \to \{0,1\}^*$, and, more generally, to *collections of functions.* Let $\mathcal{F}$ be a collection of functions with an associated representation (by default, a boolean or arithmetic circuit). We say that a class of randomized functions $\hat{\mathcal{F}}$ is a $\delta(n)$-correct, $(t(n), \varepsilon(n))$-private RE of $\mathcal{F}$ if there exists an efficient algorithm (compiler) which gets as an input a function $f : X \to Y$ from $\mathcal{F}$ and outputs (in time polynomial in the representation length $|f|$) three circuits $(\hat{f} \in \hat{\mathcal{F}}, \mathsf{Dec}, \mathsf{Sim})$ which form a $(t(n), \varepsilon(n)$-RE of f where n denotes the input length of f, i.e., $n = \log |X|$.

Remark 2.2. *We use n both as an input length parameter and as a cryptographic "security parameter" quantifying computational privacy. When describing some of our constructions, it will be convenient to use a separate parameter k for the latter, where computational privacy will be guaranteed as long as* $k = n^{\epsilon}$ *for some constant* $\epsilon > 0$.

Variants. Several variants of randomized encodings have been considered in the literature. Correctness is said to be *perfect* when $\delta = 0$ and *statistical* if $\delta(n)$ is negligible (i.e., $\delta(n) = n^{-\omega(1)}$). We say that the encoding is ε-private if it satisfies (t, ε)-privacy for every t, namely, the simulator output $\mathsf{Sim}(f(x))$ is ε-close in statistical distance to the encoding $\hat{f}(x, U_m)$. Under this convention, *perfect privacy* corresponds to 0-privacy and *statistical privacy* corresponds to $\varepsilon(n)$-privacy for some negligible ε. An encoding is *computationally private* if it is $(t, 1/t)$-private for every polynomial $t(n)$. Throughout this paper, we mainly focus on *computational encodings* which are both computationally private and perfectly correct. However, we will also mention basic feasibility results regarding *perfect encodings* which achieve perfect correctness and perfect privacy.

2.2 Efficiency and Simplicity

So far, the notion of RE can be trivially satisfied by taking $\hat{f} = f$ and letting the simulator and decoder be the identity functions.[2] To make the definition nontrivial, we should impose some efficiency constraint. The most obvious efficiency measure is sequential time; i.e., we would like to encode functions computable by large circuits (or by Turing machines with large time complexity) via relatively small circuits (or fast Turing machines). In addition to sequential time, several other notions of efficiency/simplicity have been considered in the literature. Here we review only a few of them.

Online/offline complexity. We would like to measure separately the complexity of the outputs of $\hat{f}$ which depend solely on r (*offline* part) from the ones which depend on both x and r (*online* part).[3] Without loss of generality, we assume that $\hat{f}$ can be written as $\hat{f}(x; r) = (\hat{f}_{\mathsf{off}}(r), \hat{f}_{\mathsf{on}}(x; r))$, where $\hat{f}_{\mathsf{off}}(r)$ does not depend on x at all. We can therefore split the communication complexity (resp., computational complexity) of $\hat{f}$ into the *online communication complexity* (resp., *online computational complexity*), which corresponds to the online part $\hat{f}_{\mathsf{on}}(x; r)$, and the *offline communication complexity* (resp., *offline computational complexity*), which corresponds to the offline part $\hat{f}_{\mathsf{off}}(r)$.

[2] The omission of efficiency requirements from Definition 2.1 is not accidental. We believe that, at a definitional level, it is best to leave such requirements unspecified and adopt appropriate efficiency requirements in an application-dependent context. Moreover, the ability to treat inefficient encodings as "legal REs" turns out to be useful. Indeed, as we will later see, some constructions start with a trivial RE and gradually convert it into a more efficient one.

[3] The online/offline terminology hints towards applications in which f is known ahead of time in an "offline phase", whereas x becomes available only later in an "online phase".

Efficient online encodings. Let $\hat{\mathcal{F}}$ be an encoding of the collection $\mathcal{F}$. We say that $\hat{\mathcal{F}}$ is *online efficient* if, for every function $f \in \mathcal{F}$, the online computational complexity of the encoding $\hat{f}$ is *independent* of the computational complexity (i.e., circuit size) of the encoded function f (but grows with the bit length of the input of f). We may further require *online universality*, which means that the computation of the online part $\hat{f}_{\mathsf{on}}$ corresponds to some universal computation which is independent of the encoded function f. In such a case, we may think of the encoding of $\mathcal{F}$ as being composed of two mappings: One that maps the function $f \in \mathcal{F}$ and the randomness r to $\hat{f}_{\mathsf{off}}(r)$, and one that maps the input x and the randomness r to $\hat{f}_{\mathsf{on}}(x; r)$. (Indeed, this view is taken in [26].) In the context of garbled circuits, $\hat{f}_{\mathsf{off}}(r)$ is typically referred to as the *garbled circuit* part, and $\hat{f}_{\mathsf{on}}(x; r)$ is typically referred to as the garbled input. It is sometimes useful to think of $\hat{f}_{\mathsf{off}}(r)$ as a ciphertext and $\hat{f}_{\mathsf{on}}(x; r)$ as a key K_x that "opens" the ciphertext to the value $f(x)$.

Remark 2.3. *The distinction between the online part of the encoding and its offline part naturally gives rise to a stronger form of* adaptive privacy *in which the adversary may choose the input x based on the offline part of the encoding. This form of security is discussed later, in Section 4.5. For now, we use the online/offline terminology only as an efficiency requirement without imposing any additional requirement on privacy; That is, all we require is standard privacy as per Definition 2.1.*

Affinity and decomposability. Some of the applications of REs further require some form of algebraic simplicity. Assume that the function f is an arithmetic function whose input $x = (x_1, \ldots, x_n)$ is a vector of elements of some ring R. We say that an RE $\hat{f} : \mathsf{R}^n \times \{0,1\}^m \to \mathsf{R}^s$ of f is an *affine randomized encoding* (ARE) if, for every fixing of the randomness r, the online part of the encoding $\hat{f}_{\mathsf{on}}(x; r)$ becomes an affine function over the ring R, i.e., $\hat{f}_{\mathsf{on}}(x; r) = M_r \cdot x + v_r$, where M_r (resp., v_r) is a matrix (resp., vector) that depends on the randomness r. We say that $\hat{f}$ is a *decomposable* randomized encoding (DRE) if each output of the online part of $\hat{f}$ depends on at most a single input x_i. Namely, $\hat{f}_{\mathsf{on}}(x; r)$ decomposes to $(\hat{f}_1(x_1; r), \ldots, \hat{f}_n(x_n; r))$, where $\hat{f}_i$ may output several ring elements. An RE which is both decomposable and affine is called a DARE. Observe that, in the binary case where $\mathsf{R} = \mathbb{Z}_2$, decomposability implies affinity and so any DRE is also a DARE, although this is not the case for general rings.

Algebraic degree and output locality. Affinity and decomposability essentially require simple dependency on x. One can strengthen these notions by also placing restrictions on the way $\hat{f}$ depends on the randomness r. Specifically, one can require that the *algebraic degree* of each output of $\hat{f}(x; r)$, viewed as a polynomial in r and x, will be small (e.g., constant). Similarly, one may require that each output of $\hat{f}(x; r)$ will depend on a constant number of inputs (including the r's), namely that $\hat{f}$ should have constant *output locality.* Over the binary ring $\mathbb{Z}_2$, constant locality implies constant degree (since over the binary ring, polynomials may be assumed to be multilinear).

The necessity of randomness. For any of the above notions of simplicity, randomness is necessary in order to obtain a "simple" encoding for a "nonsimple" boolean function. To see this, assume that $f : \{0,1\}^n \to \{0,1\}$ has a deterministic encoding $\hat{f} : \{0,1\}^n \to \{0,1\}^s$. The privacy of the encoding promises that there exists a pair of strings $y_0, y_1 \in \{0,1\}^s$ such that, for every $x \in \{0,1\}^n$, we have $\hat{f}(x) = y_{f(x)}$. Also, by correctness, y_0 and y_1 disagree in some position, say the first. Hence, we can compute $f(x)$ by computing the first bit of $\hat{f}(x)$ or its negation. We conclude that if $\hat{f}$ is simple then so is f, assuming that "simplicity" is closed under projection (which is the case for affinity, decomposability, locality, and degree).[4]

2.3 Useful Properties

REs satisfy several natural properties which hold regardless of their efficiency. First, just like in the case of string encodings, if we take an encoding $\hat{f}$ of f, and re-encode it by $\hat{\hat{f}}$, then the resulting encoding also encodes the original function f. Second, an encoding of a tuple $(f_1(x), f_2(x))$ can be obtained by encoding each element of the pair separately and concatenating the result. Finally, given an encoding $\hat{f}(y; r)$ of $f(y)$, we can encode a function of the form $f(g(x))$ by encoding the outer function and substituting y with $g(x)$, i.e., $\hat{f}(g(x); r)$. We summarize these properties via the following lemmas [10]. For simplicity, we restrict our attention to perfectly correct encodings and refer to perfectly correct (t, ε)-private encodings as (t, ε)-encodings.

Lemma 2.4 (Composition)**.** *Suppose that $g(x; r_g)$ is a (t_1, ε_1)-encoding of $f(x)$ with decoder Dec_g and simulator Sim_g, and that $h((x, r_g); r_h)$ is a (t_2, ε_2)-encoding of the function $g(x, r_g)$, viewed as a single-argument function, with decoder Dec_h and simulator Sim_h. Then, the function $\hat{f}(x; (r_g, r_h)) = h((x, r_g); r_h)$ together with the decoder $\mathsf{Dec}(\hat{y}) = \mathsf{Dec}_g(\mathsf{Dec}_h(\hat{y}))$ and the simulator $\mathsf{Sim}(y) = \mathsf{Sim}_h(\mathsf{Sim}_g(y))$ forms a $(\min(t_1 - s, t_2), \varepsilon_1 + \varepsilon_2)$-encoding of $f(x)$ where s upper-bounds the circuit complexity of h and its simulator Sim_h.*

Proof. Perfect correctness follows by noting that $\Pr_{r_g, r_h}[\mathsf{Dec}(\hat{f}(x; r_g, r_h)) \neq f(x)]$ is upper-bounded by

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

To prove privacy, consider a t-size adversary $\mathcal{A}$ which, for some x, distinguishes the distributions $\mathsf{Sim}(f(x))$ from $\hat{f}(x)$ with advantage ε, where $\varepsilon > \varepsilon_1 + \varepsilon_2$ and $t < \min(t_1 - s, t_2)$. We show that we can either violate the privacy of the encoding g or the privacy of the encoding h. Indeed, by considering the "hybrid" distribution $\mathsf{Sim}_h(g(x))$, we can write

$$\begin{aligned} \varepsilon_1 + \varepsilon_2 &< \left| \Pr[\mathcal{A}(\mathsf{Sim}_h(\mathsf{Sim}_g(f(x)))) = 1] - \Pr_{r_g, r_h}[\mathcal{A}(h((x, r_g); r_h)) = 1] \right| & (1) \\ &= \left| \Pr[\mathcal{A}(\mathsf{Sim}_h(\mathsf{Sim}_g(f(x)))) = 1] - \Pr_{r_g}[\mathcal{A}(\mathsf{Sim}_h(g(x; r_g))) = 1] \right| & (2) \\ &\quad + \left| \Pr_{r_g}[\mathcal{A}(\mathsf{Sim}_h(g(x; r_g))) = 1] - \Pr_{r_g, r_h}[\mathcal{A}(h((x, r_g); r_h)) = 1] \right|. & (3) \end{aligned}$$

It follows that either (2) is larger than ε_1 or (3) is larger than ε_2. In the first case, we get, for some fixing of the coins of Sim_h, an adversary $\mathcal{A}(\mathsf{Sim}_h(\cdot))$ of complexity $t + s < t_1$ which violates the privacy of the encoding g. In the second case, there exists an input (x, r_g) for which $\mathcal{A}$ violates the privacy of h. □

[4] This argument relies heavily on the fact that f is a boolean function. Indeed, the claim does not hold in the case of non-boolean functions. Suppose, for example, that $f : \{0,1\}^n \to \{0,1\}^n$ is a permutation. Then it can be trivially encoded by the identity function. Moreover, if f can be computed and inverted in polynomial time, then the encoding allows efficient decoding and simulation.

Lemma 2.5 (Concatenation). *Suppose that $\hat{f}_i(x; r_i)$ is a (t, ε)-encoding of the function $f_i : \{0,1\}^n \to \{0,1\}^{\ell_i}$ with decoder Dec_i and simulator Sim_i for every $i \in [c]$. Then the function $\hat{f}(x; (r_1, \dots, r_c)) = (\hat{f}_i(x; r_i))_{i=1}^c$ together with the decoder $\mathsf{Dec}(\hat{y}) = (\mathsf{Dec}_i(\hat{y}_i))_{i=1}^c$ and simulator $\mathsf{Sim}(y) = (\mathsf{Sim}_i(y_i))_{i=1}^c$ forms a $(t - cs, c\varepsilon)$-encoding of $f(x) = (f_1(x), \dots, f_c(x))$ where s is an upper-bound on the complexity of the encodings $\hat{f}_i$.*

Proof. Perfect correctness follows from

$$\Pr_r[\mathsf{Dec}(\hat{f}(x;r)) \neq f(x)] \leq \sum_{i=1}^{c} \Pr_r[\mathsf{Dec}(\hat{f}_i(x;r_i)) \neq f_i(x)] = 0.$$

Privacy is proved via a standard hybrid argument. Specifically, suppose towards a contradiction, that $\mathcal{A}$ is a $(t - cs)$-size adversary that distinguishes $\hat{f}(x; r)$ from $\mathsf{Sim}(f(x); \rho)$ 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

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

and the tuple

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

By an averaging argument, $\mathcal{A}$ distinguishes with advantage at least ε between $(w, \mathsf{Sim}_j(f_j(x)), z)$ and $(w, \hat{f}_j(x; r_j), z)$ for some fixing $w = (w_1, \dots, w_{j-1})$ and $z = (z_{j+1} \dots z_c)$. Let $\mathcal{B}$ be an adversary that, given a challenge $\hat{y}_j$, calls $\mathcal{A}$ on the input $(w, \hat{y}_j, z)$ and outputs the result. The adversary $\mathcal{B}$ can be implemented by a t-size circuit and it distinguishes $\hat{f}_j(x; r_j)$ from $\mathsf{Sim}_j(f_j(x))$ with advantage ε, in contradiction to our hypothesis. □

Lemma 2.6 (Substitution). *Suppose that the function $\hat{f}(x; r)$ is a (t, ε)-encoding of $f(x)$ with decoder Dec and simulator Sim. 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 $\hat{h}(z; r) = \hat{f}(g(z); r)$ 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,

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

and for privacy we have that, for all z, the distribution $\mathsf{Sim}(h(z)) = \mathsf{Sim}(f(g(z)))$ cannot be distinguished from the distribution $\hat{f}(g(z)) = \hat{h}(z)$ with advantage better than ε by any t-size adversary. □

Remark 2.7 (Composition, concatenation, and substitution with constructive simulators/decoders). *Observe that, in all the above cases, the simulator and the decoder of the new encoding can be derived in a straightforward way based on the original simulators and decoders.*

Remark 2.8 (The efficiency of simulation). *In the composition lemma, the loss in security depends, in part, on the complexity of the simulator. Typically, the latter is roughly the same as the complexity of the encoding itself. Indeed, a natural way to define a simulator $\mathsf{Sim}(y)$ is to sample an encoding $\hat{f}(x_y)$ where x_y is some fixed canonical preimage of y under f. The complexity of such a simulator is the same as the complexity of the encoding plus the cost of finding x_y given y. In fact, it is* *not hard to see that, if $\hat{f}$ is a (t,ε)-private encoding with some simulator Sim', then the canonical simulator defined above is $(t,2\varepsilon)$-private. Of course, in some cases, the canonical simulator may not be efficiently computable, since it may be hard to find a canonical input x_y for some y's (e.g., when f is a one-way permutation).*

3 Feasibility Results

In this section we present basic feasibility results regarding the existence of DARE for several rich function classes. We present these constructions via the unified framework of [15].

3.1 Perfect DARE for Finite Functions

As a warmup, we consider several examples of DARE for finite functions. (The examples apply to any finite ring.)

Example 3.1 (Addition)**.** *The addition function $f(x_1,x_2) = x_1 + x_2$ over some finite ring R is perfectly encoded by the DARE*

$$\hat{f}(x_1,x_2;r) = (x_1 + r, x_2 - r).$$

Indeed, decoding is performed by summing up the two components of the encoding, and simulation is done by sampling a random pair whose sum equals to $y = f(x_1,x_2)$.

Example 3.2 (Multiplication)**.** *The product function $f(x_1,x_2) = x_1 \cdot x_2$ over a ring R is perfectly encoded by the ARE*

$$g(x_1,x_2;r_1,r_2) = (x_1 + r_1, x_2 + r_2, r_2x_1 + r_1x_2 + r_1r_2).$$

Indeed, given an encoding (c_1,c_2,c_3), we can recover $f(x)$ by computing $c_1 \cdot c_2 - c_3$. Also, perfect privacy holds, since the triple (c_1,c_2,c_3) is uniform subject to the correctness constraint. Hence, the simulator $\mathsf{Sim}(y;c_1,c_2) := (c_1,c_2,c_1c_2 - y)$ perfectly simulates g. Observe that the above encoding is affine but not decomposable (the last entry depends on both x_1 and x_2). We can derive a DARE by viewing the last entry $g_3(x,r) = (r_2x_1) + (r_1x_2 + r_1r_2)$ as a deterministic function in x and r and re-encoding it via the DARE for addition (Example 3.1):

$$\hat{g}_3(x,r;s) = (r_2x_1 + s, r_1x_2 + r_1r_2 - s).$$

By the concatenation lemma, the function $\hat{g}(x,r;s) = (g_1(x,r), g_2(x,r), \hat{g}_3(x;r))$ perfectly encodes $g(x,r)$, and therefore, by the composition lemma, $\hat{g}(x;r,s)$ perfectly encodes $f(x)$.

Using similar ideas, it can be shown that any polynomial f over a ring R can be encoded by a perfect DARE (whose complexity may be large). In the next section, we show how to achieve complexity which is polynomial in the formula size of f. For this, it will be useful to record the following concrete DARE for the function $x_1x_2 + x_3$ (MUL-ADD).

Fact 3.3 (DARE for MUL-ADD [15])**.** *The function $f(x_1,x_2,x_3) = x_1x_2 + x_3$ over a finite ring R is perfectly encoded by the DARE $\hat{f}(x_1,x_2,x_3;r_1,r_2,r_3,r_4)$ defined by*

$$\left(x_1 \begin{bmatrix} 1 \\ r_2 \end{bmatrix} + \begin{bmatrix} -r_1 \\ -r_1r_2 + r_3 \end{bmatrix}, \quad x_2 \begin{bmatrix} 1 \\ r_1 \end{bmatrix} + \begin{bmatrix} -r_2 \\ r_4 \end{bmatrix}, \quad x_3 - r_3 - r_4 \right),$$

where r_1,r_2,r_3,r_4 are random and independent ring elements.

3.2 Perfect DARE for Formulas

Our goal in this section is to construct perfect DARE for arithmetic circuits with logarithmic depth (i.e., formulas). 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. We prove the following theorem:

Theorem 3.4. *There exists a perfect DARE for the class of arithmetic circuits with logarithmic depth over an arbitrary ring.*[5]

We mention that a stronger version of the theorem (cf. [67, 40, 10]) provides perfect DAREs for arithmetic branching programs over an arbitrary ring. The latter class is believed to be computationally richer than the class of logarithmic-depth arithmetic circuits, and therefore, as a feasibility result, the branching program-based encoding subsumes the one from Theorem 3.4.[6] The existence of efficient perfect or statistical DARE for general polynomial-size circuits, or even for circuits with, say, $\log^2 n$ depth, is wide open.

Proof of Theorem 3.4. Let C be an arithmetic circuit over a ring R. Instead of individually considering each wire and gate of C (as in Yao's classical garbled circuit construction), we 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. We further assume that each gate has a bounded fan-out, say of 2. (For circuits of logarithmic depth, such a restriction can be forced with overhead polynomial in the size while keeping the depth logarithmic.) 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.

We build the encoding Enc in an iterative fashion, processing the layers of C from top (outputs) to bottom (inputs). We start with a trivial encoding of the identity function C^0. In iteration i, $i = 1, 2, \ldots, h$, we transform 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}, and then re-encoding the resulting function to bring it into a *decomposable affine form*. The affinization step is performed by adding new random inputs and has the effect of increasing the size of the online part.

Formally, the DARE compiler is described in Figure 1. As explained in Figure 1, Lemmas 2.4, 2.5, and 2.6 guarantee that, in the i-th iteration, the compiler generates a decomposable affine encoding $\mathsf{Enc}^i(y^i)$ of $C^i(y^i)$, and so, at the final iteration, we derive a DARE for $C^h = C$. Observe that the computational complexity of Enc^i (and, in particular, the length of its online part) is larger by a constant factor than the complexity of Enc^{i-1}. More precisely, the online part grows by a factor of at most twice the fan-out of B_i. Hence, the encoding can be generated in polynomial time as long as the encoded circuit has logarithmic depth. Circuits for the simulator and decoder can be generated with similar complexity due to the constructive nature of the substitution, composition, and concatenation lemmas (see Remark 2.7). This completes the proof of the theorem. □

[5] Recall that, by default, a statement like this always means that the encoding is efficiently constructible as defined in Section 2.

[6] Arithmetic branching programs can emulate arithmetic circuits of logarithmic depth with only polynomial overhead, whereas the converse is believed to be false (this is equivalent to separating log-space computation from $\mathbf{NC}^1$).

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 two steps:

   (a) **Substitution.** Let $F(y^i) = \mathsf{Enc}^{i-1}(B_i(y^i))$.
   The substitution lemma (2.6) guarantees that, if Enc^{i-1} encodes C^{i-1}, then F encodes C^i. Assuming that Enc^{i-1} is a DARE, each output of F can be written as

   $$Q_* = a * (y^i_\ell * y^i_r) + b \quad \text{or as} \quad Q_+ = a * (y^i_\ell + y^i_r) + b,$$

   where a and b depend on the randomness and ℓ and r are some indices of gates in the i-th layer. This means that F is not decomposable anymore, since each of the resulting outputs depends on a pair of input variables of C^i. Moreover, if the i-th layer contains a multiplication gate, then F is also not affine (since Q_* contains a product of two input variables of C^i).

   (b) **Affinization.** Turn F into a decomposable affine encoder Enc^i of the same function by applying to each output that depends on two inputs y^i_j a perfect DARE. Here we rely on the fact that the above expressions (Q_+ and Q_*) are finite and thus have a constant-size DARE. Concretely, the term Q_* can be handled by applying the basic gadget of Fact 3.3 to $Q_* = z * y^i_r + b$ and substituting ay^i_ℓ into z. The resulting encoding of Q_* can be written in the form $\hat{Q}_* = (a'_\ell y^i_\ell + b'_\ell, a'_r y^i_r + b'_r, w)$, where a'_i, b'_i, w are vectors in R^2 that depend only on random inputs. Similarly, we can handle Q_+ by rewriting it as $Q_+ = z_1 + z_2$, where $z_1 = a * y^i_\ell$ and $z_2 = a * y^i_r + b$, and then applying the encoding for addition (Example 3.1), leading to a DARE of the form $(a * y^i_\ell + b', a * y^i_r + b'')$, where $a, b, b'' \in \mathsf{R}$. Applying this transformation to every term Q in the output of F and concatenating different affine functions of the same input, we get, by the concatenation and composition lemmas (2.5, 2.4), decomposable affine encoding Enc^i of C^i.

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

Figure 1: DARE compiler for arithmetic circuits of logarithmic depth. For simplicity, we treat encoders as probabilistic circuits and omit their random inputs from the notation.

Example 3.5. *Let us apply the DARE compiler (Figure 1) to the formula $ab+cd$ depicted in Figure 2. The initial trivial encoding is simply y^0. Then, substitution yields $y^1_1+y^1_2$, which after affinization becomes $(y^1_1+r_0, y^1_2-r_0)$. Another substitution results in the encoding $(x_1x_2+r_0, x_3x_4-r_0)$. After an additional affinization, we get the final encoding $\hat{f}(x_1, x_2, x_3, x_4; (r_0, r_1, r_2, r_3, r_4, r'_1, r'_2, r'_3, r'_4))$ defined by*

$$x_1 \begin{bmatrix} 1 \\ r_2 \end{bmatrix} + \begin{bmatrix} -r_1 \\ -r_1r_2+r_3 \end{bmatrix}, x_2 \begin{bmatrix} 1 \\ r_1 \end{bmatrix} + \begin{bmatrix} -r_2 \\ r_4 \end{bmatrix}, x_3 \begin{bmatrix} 1 \\ r'_2 \end{bmatrix} + \begin{bmatrix} -r'_1 \\ -r'_1r'_2+r'_3 \end{bmatrix}, x_4 \begin{bmatrix} 1 \\ r'_1 \end{bmatrix} + \begin{bmatrix} -r'_2 \\ r'_4 \end{bmatrix}, \begin{bmatrix} r_0-r_3-r_4 \\ -r_0-r'_3-r'_4 \end{bmatrix}.$$

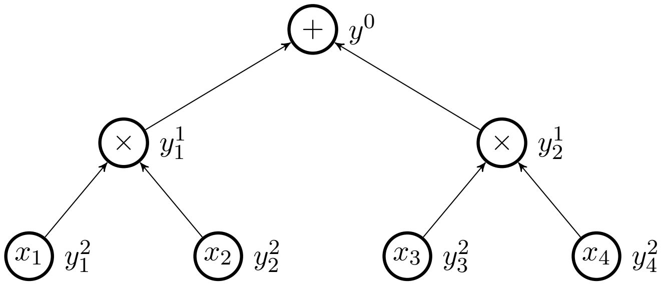

Figure 2: An arithmetic formula computing the function $x_1x_2+x_3x_4$.

3.3 Generalization: "Simple" Encodings for Formulas

The proof of Theorem 3.4 provides a general template for constructing "simple" encodings. Formally, consider a class Simple of single-output (possibly randomized) functions defined over some ring R. Call an encoding $\hat{f}$ *simple* if each of its outputs is computed by a function in Simple.

Theorem 3.6 (Simple encodings for formulas (meta-theorem))**.** *Suppose that the class* Simple *satisfies the following conditions:*

- Simple *contains the identity function $f(x)=x$.*
- *There exists a simple DARE $\hat{g}$ for every function g obtained by taking some function $f(x;r) \in$* Simple *and substituting a deterministic input x_i by the expression $y \diamond z$, where y and z are deterministic inputs and $\diamond \in \{+,*\}$ is a ring operation. Moreover, the circuit complexity of $\hat{g}$ is at most c times larger than the complexity of f for some universal constant c.*

Then, any depth-d arithmetic circuit $C : \mathsf{R}^n \to \mathsf{R}^\ell$ has a simple perfect encoding of complexity at most ℓc^d.

Theorem 3.4 can be derived from Theorem 3.6. Indeed, the notion of DARE corresponds to the case where Simple is the class of randomized functions with a single deterministic input x and (possibly many) random inputs r of the form $f(x;r) = x * g(r) + h(r)$, where g and h are arbitrary.

The proof of Theorem 3.6 follows the outline of Theorem 3.4 except that the affinization gadget is replaced with a simple encoding of $f(x \diamond y)$ (for appropriate f and $\diamond$). The details are left for the reader. We further mention that one can state a more general result by considering circuits over an arbitrary basis $G = \{g\}$, assuming that $f(g(x_1, \ldots))$ admits a simple encoding for every $f \in \mathsf{Simple}$ and every $g \in G$. In some cases, this version leads to better concrete efficiency then the current "gate-by-gate" approach.

3.4 Computational DARE for Boolean Circuits

Falling short of providing perfect DARE for general circuits, we aim for computationally private DARE. Let us reconsider the previous encoding (Figure 1) under the terminology of *keys* (online part) and *ciphertexts* (offline part). In each iteration, the affinization step increases the length of the keys by a constant factor, and as a result the size of the keys becomes exponential in the depth. We fix this problem by using a *key-shrinking gadget.*

Definition 3.7 (Key-shrinking gadget)**.** *Consider the affine function with $4k$-long keys over the ring* R

$$f(y,c,d) = yc + d, \qquad y \in \mathsf{R}, c, d \in \mathsf{R}^{4k}.$$

A key-shrinking gadget *is a computational encoding $\hat{f}$ for f with shorter keys of length k of the form*

$$\hat{f}(y,c,d;r) = (ya + b, W) \qquad y \in \mathsf{R}, a, b \in \mathsf{R}^{k},$$

where a, b, and W may depend on c, d and on the internal randomness of $\hat{f}$ but not on y.

Remark 3.8. *The concrete shrinkage factor ($4k$ to k) is somewhat arbitrary. Indeed, it is possible to turn a key-shrinking gadget with minimal shrinkage ($k+1$ to k) into a key-shrinking gadget with arbitrary polynomial shrinkage (k^c to k for any constant $c > 0$) by re-encoding the gadget polynomially many times and applying the composition lemma (2.4).*

Given such a gadget, we can fix the key-blowup problem of the previous encoding. First, rearrange the affine encoding obtained from the affinization step terms into blocks (grouping together outputs that depend on the same variable), and then apply 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 offline part (as a ciphertext). See Figure 3 for a formal description. Again, correctness and privacy follow easily (and modularly) from the substitution, concatenation, and composition properties of randomized encodings (see Section 2.3).

Lemma 3.9. *Assume that the key-shrinking gadget is (t, ε)-private and can be computed and simulated by a circuit of size s. Then, the compiler from Figure 3 constructs a perfectly correct $(t - |C|s, |C|\varepsilon)$-private DARE for the circuit C with online complexity kn and offline complexity of at most $s|C|$, where n is the input length of C.*

Proof. For brevity, we refer to a perfectly correct (t, ε)-private encoding with online complexity of k and offline complexity of m as a (t, ε, k, m)-encoding. Keeping the notation from Theorem 3.4, we prove that, in the i-th iteration, the compiler generates a $(t_i, \varepsilon_i, k\ell_i, m_i)$-DARE $\mathsf{Enc}^i(y^i)$ of $C^i(y^i)$, where $t_i = t - s|C^i|$, $\varepsilon_i = \varepsilon|C^i|$, $m_i = s|C^i|$, and ℓ_i denotes the input length of C^i (which is the number of gates in the i-th layer of the circuit C).

The proof is by induction on i. Observe that the claim is trivial for $i = 0$. Let us assume that the claim holds for the $i-1$ iteration. As in the proof of Theorem 3.4, Lemmas 2.4, 2.5, and 2.6 guarantee that the function G^i is a $(t_{i-1}, \varepsilon_{i-1}, 4k\ell_i, m_{i-1})$-DARE of $C^i(y^i)$. Indeed, recall that affinization increases the online complexity by a factor of two (and does not affect the offline complexity). Since the fan-out is assumed to be two, this leads to a total factor of 4 in the online complexity. The key-shrinking gadget provides a (t, ε, k, s) affine encoding for G^i_j (for every $j \in [\ell_i]$). Therefore, by Lemma 2.5, the concatenation of these encodings $\mathsf{Enc}^i(y^i)$ is a $(t - \ell_i s, \ell_i\varepsilon, k\ell_i, s\ell_i)$-encoding of G^i.

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 $F(y^i) = \mathsf{Enc}^{i-1}(B_i(y^i))$.

   (b) **Affinization.** Turn F into a decomposable affine encoder G^i of the same function by applying to each output that depends on two inputs y^i_j a perfect DARE. (See affinization step in Figure 1.)

   (c) **Key shrinkage.** To avoid the exponential blowup of keys, the compiler applies the key-shrinking gadget. Partition G^i to $(G^i_1, \ldots, G^i_\ell)$, where $G^i_j = c_j y^i_j + d_j$ is an affine function in the variable y^i_j and ℓ is the number of gates in the i-th level. For every G^i_j whose length is larger than k, the key-shrinking gadget is applied to bring it to the form $(W, a_j y^i_j + b_j)$, where $a_j \in \mathsf{R}^k$, $b_j \in \mathsf{R}^k$, and a_j, b_j, W depend only on random inputs. The W entries are aggregated in the offline part of the encoding. Let $\mathsf{Enc}^i(y^i)$ be the decomposable affine encoding resulting from this step.

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

Figure 3: Encoding arithmetic circuits via key-shrinking gadget. Think of k as a security parameter which is set it to some polynomial in the input length n. We assume, without loss of generality, that the circuit has fan-out of two.

Finally, by the composition lemma (2.4), $\mathsf{Enc}^i(y^i)$ is a $(t_i - \ell_i s, \varepsilon_i + \ell_i \varepsilon, k\ell_i, m_{i-1} + s\ell_i)$-DARE of $C^i(y^i)$, as required. □

As in Theorem 3.4, the compiler implicitly defines a simulator and decoder of complexity $\text{poly}(|C|, s)$ using the constructive version of the substitution, composition, and concatenation lemmas mentioned in Remark 2.7.

Shrinking the keys in the binary case. Lemma 3.9 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. First, observe that, in the binary setting, an affine function $ya + b$ in y operates as a "selection" function which outputs b if $y = 0$, and $a + b$ if $y = 1$. Hence, in order to implement the key-shrinking gadget, we need to encode the selection function

$$\text{Sel}(y, M_0, M_1) := M_y, \qquad \text{where } y \in \{0,1\}, M_0, M_1 \in \{0,1\}^n$$

with n-long vectors (M_0, M_1) by a selection function with shorter vectors (K_0, K_1) (and, possibly, an offline part). Let us start with a slightly easier task and try to encode a variant of the selection function which leaks the "selection bit" y, i.e.,

$$\text{Sel}'(y, M_0, M_1) := (y, M_y).$$

A simple way to implement such an encoding is to employ symmetric encryption, where M_0 is encrypted under the random key K_0 and M_1 under the random key K_1. Formally, we prove the following claim:

Claim 3.10. *Let* (E, D) *be a perfectly correct one-time semantically secure symmetric encryption with key length* k *and message length* n. *Then, the function*

$$g'(y, M_0, M_1; (K_0, K_1)) = (\text{Sel}'(y, K_0, K_1), \mathsf{E}_{K_0}(M_0), \mathsf{E}_{K_1}(M_1))$$

is a computational DARE for $\text{Sel}'(y, M_0, M_1)$.

Sketch. Given the encoding (y, K_y, C_0, C_1), we can decode M_y by applying the decryption algorithm to C_y. The simulator takes (y, M_y) as input, samples a pair of random keys K_0 and K_1, and outputs (y, K_y, C_0, C_1), where $C_y = \mathsf{E}_{K_y}(M_y)$ and $C_{1-y} = \mathsf{E}_{K_y}(0^n)$. It is not hard to show that the simulator's output is computationally indistinguishable from the distribution of the real encoding based on the semantic security of the encryption. □

We move on to encode the selection function Sel. For this we have to hide the value of y. One way to achieve this is to remove y from the output of the previous encoding and to randomly permute the order of the ciphertexts C_0 and C_1. The resulting function privately encodes Sel, however, to guarantee correctness we have to assume that the encryption is verifiable (i.e., decryption with an incorrect key can be identified). This variant of the encoding typically leads to a statistically small decoding error (as in the garbled circuit variant of [76]). A perfectly correct solution can be achieved by releasing a "pointer" to the correct ciphertext. (This version corresponds to the garbled circuit variant in [21, 80, 11].) More abstractly, observe that the input to $\text{Sel}(y, M_0, M_1)$ can be randomized to

$$y' = y + r, \quad M_0' = (1 - r) \cdot M_0 + r \cdot M_1, \quad \text{and} \quad M_1' = r \cdot M_0 + (1 - r) \cdot M_1,$$

where the random bit r is treated as a scalar over $\mathbb{Z}_2$, and M_0 and M_1 as vectors over $\mathbb{Z}_2^n$. This means that $\text{Sel}(y, M_0, M_1)$ can be encoded by $\text{Sel}'(y', M_0', M_1')$ (since y' can be released). We can now re-encode $\text{Sel}'(y', M_0', M_1')$ via Claim 3.10, and derive an encoding $g(y, M_0, M_1; (r, K_0, K_1))$ for the Sel function whose polynomial representation is

$$(y + r, (1 - (y + r))K_0 + (y + r)K_1, \mathsf{E}_{K_0}((1 - r)M_0 + rM_1), \mathsf{E}_{K_1}(rM_0 + (1 - r)M_1)).$$

It is not hard to see that g satisfies the syntactic requirements of Definition 3.7. We therefore obtain a key-shrinking gadget based on a semantically secure encryption scheme. Since the latter can be based on any one-way function, we derive the following corollary (Yao's theorem):

Theorem 3.11. *Assuming the existence of one-way functions, there exists an online-efficient DARE for the class of polynomial-size boolean circuits.*

The above encoding provides a modular and, arguably, conceptually simpler derivation of Yao's original result for boolean circuits.

3.5 Computational DARE for Arithmetic Circuits

We would like to obtain online-efficient DARE for *arithmetic* circuits. Since arithmetic computations arise in many real-life scenarios, this question has a natural motivation in the context of most of the applications of garbled circuits (to be discussed in Section 5). Clearly, one can always encode an arithmetic function $f : \mathsf{R}^n \to \mathsf{R}^\ell$ by implementing an equivalent boolean circuit f' (replacing each R-operation by a corresponding boolean subcircuit) and applying a boolean encoding to the result. This approach suffers from an overhead which depends on the boolean complexity of ring operations (which may be too high), and requires access to the *bits* of the inputs, which may not be accessible in some scenarios.[7] Instead, we would like to obtain an arithmetic encoding which treats the elements as *atomic* values and manipulates them only via *ring operations* just like the arithmetic encoding of Theorem 3.4 for the case of logarithmic-depth arithmetic circuits (i.e., formulas).

Observe that the encoding presented in Figure 3 reduces the problem of constructing arithmetic ARE to the construction of an arithmetic key-shrinking gadget, since all other components "arithmetize". Over a large ring, the key-shrinking gadget essentially implements a symmetric encryption scheme with special homomorphic properties that applies both to the keys and to data. Indeed, we can view the offline part of the key-shrinking gadget as an encryption of the long keys $c, d \in \mathsf{R}^n$ under the short keys $a, b \in \mathsf{R}^k$ with the property that the linear combination of the keys $ay + b$ allows one to decrypt the linear combination of the plaintexts $cy + d$ (and hides any other information). Such a gadget was (approximately) implemented over bounded-size integers by [15], leading to the following theorem:

Theorem 3.12 (Informal)**.** *Assuming the hardness of the* Learning with Errors *problem, there exists an efficient compiler that, given an arithmetic circuit $f : \mathbb{Z}^n \to \mathbb{Z}^\ell$ over the integers and a positive integer U bounding the value of circuit wires, outputs (in time polynomial in the size of f and in the binary representation of U) an arithmetic circuit $\hat{f}$ (over the integers) which forms an online-efficient DARE of f.*

We mention that [15] also provide an alternative (less efficient) construction of arithmetic garbled circuits over the integers based on one-way functions.

[7] E.g., when the input ring elements are encrypted under some cryptographic scheme that supports some limited form of ring operations.

Mod-p arithmetic encodings. The task of encoding circuits over $\mathbb{Z}_p$ can be reduced to encoding circuits over the integers (e.g., by applying the encoding in Example 1.1). The reduction is efficient but "nonarithmetic" since it treats the elements of $\mathbb{Z}_p$ as integers (as opposed to abstract ring elements of $\mathbb{Z}_p$). It would be interesting to obtain a *fully arithmetic encoder* which treats the ring $\mathbb{Z}_p$ in a fully black-box way.[8] We ask:

> Is there an efficient compiler that takes an arithmetic circuit $f : \mathsf{R}^n \to \mathsf{R}^\ell$ which treats the ring in an abstract way and generates an arithmetic circuit $\hat{f}$ over R such that, for any concrete (efficiently implementable) ring R, the function $\hat{f}$ forms an online-efficient ARE of f?

This question is open even for the special case of prime-order fields $\mathbb{Z}_p$. It was recently shown in [8] that, if the decoder is also required to be fully arithmetic (as achieved by Theorem 3.4 for formulas), then the online complexity must grow with the output length of f and so it is impossible to achieve *online efficiency.* This limitation applies even to the simpler case of prime-order fields $\mathbb{Z}_p$. Extending this impossibility result to single-output functions remains an interesting open problem.

4 Advanced Constructions

Having established the feasibility of online-efficient DARE, we move on to study their complexity. In Section 4.1, we minimize the parallel-time complexity of the encoding, and in Section 4.2, we minimize the online communication complexity. The computational complexity of REs is amortized in Section 4.3 by reusing the offline part of REs over many invocations. Then, in Section 4.4, we study the possibility of reducing the total complexity even in a single invocation. We end with a discussion on adaptively secure REs (Section 4.5).

4.1 Minimizing the Parallel Complexity

In this section, we show that any efficiently computable function admits an encoding with constant *locality*; i.e., each of its outputs depends on a constant number of inputs (counting both deterministic and random inputs). Such an encoding can be computed by a constant-depth circuit with bounded fan-in gates (also known as an $\mathbf{NC^0}$ circuit), and so it captures a strong form of constant-parallel-time computation.

We begin with the following theorem from [10]:

Theorem 4.1 ($\mathbf{NC^0}$ RE for branching programs)**.** *There exists a perfect degree-3 DARE in which each output depends on four inputs for the class of arithmetic branching programs over an arbitrary ring.*

The theorem is based on a direct randomization technique for branching programs from [67, 40]. Below, we sketch a proof for a weaker version of the theorem that applies to arithmetic circuits of logarithmic depth ($\mathbf{NC}^1$ circuits).

[8] Formally, one should define a set of legal operations (gates) which are available for the encoder. A natural choice is to allow field operations (addition, subtraction, multiplication, and possibly division and zero-checking), access to the constants $\{0, 1\}$, and some form of randomization gate, say allowing to sample random field elements and a random $\{0, 1\}$ elements.

Proof of Theorem 4.1 restricted to $\mathbf{NC}^1$ *functions.* We rely on the meta-theorem for encoding logarithmic-depth circuits (Theorem 3.6). Let Simple denote the class of randomized functions of the form $\{abc + d, b + c + d\}$ where a can be a deterministic or a random input, and b, c, d are either random inputs (possibly with a minus sign) or ring constants. Recall that an encoding $\hat{f}$ is simple if, for every output i, the function that computes the i-th output of $\hat{f}$ is a function in Simple. By Theorem 3.6, it suffices to present a perfect simple DARE of finite complexity for the function $Q_+ = (x + y)bc + d$ and for the function $Q_* = (x * y)bc + d$ where x and y are deterministic inputs. (Note that only a is allowed to be a deterministic input, and so we do not have to consider the case where one of the other variables is substituted.)

Indeed, for the case of addition, the function Q_+ can be written as $X + Y$ where $X = xbc + d$ and $Y = ybc$. By applying the encoding for addition (Example 3.1), we derive the encoding $(xbc + d + r, ybc - r)$. The first item can be viewed as $X + Y$ where $X = xbc$ and $Y = d + r$, and so, by re-encoding it again, we get the encoding $(xbc + r', d + r - r', ybc - r)$, which is simple.

We move on to the case of multiplication. For simplicity, assume that the ring is commutative. (The more general case can also be treated with slightly more work.) The function Q_* can then be written as $XY + d$ where $X = xbc$ and $Y = ybc$. By applying the MUL-ADD encoding (Fact 3.3), we derive an encoding which is simple except for one output of the form $r_0xb + r_1r_2 + r_3$. The latter can be brought to simple form via double use of the addition encoding (Example 3.1). □

In [11] it was shown how to extend the above theorem to the case of polynomial-size circuits. Formally,

Theorem 4.2 ($\mathbf{NC^0}$-RE for circuits)**.** *Assuming the existence of one-way functions computable by circuits of logarithmic depth (*$\mathbf{NC}^1$*), the class of polynomial-size boolean circuits admits an online-efficient DARE of degree 3 and locality 4.*

Proof idea. First observe that it suffices to encode a polynomial-size boolean circuit f by an $\mathbf{NC}^1$-computable encoding g. Indeed, given such an encoding, we can apply Theorem 4.1 to re-encode the encoding g by an $\mathbf{NC^0}$ encoding $\hat{g}$ and derive (by the composition lemma) an $\mathbf{NC^0}$ encoding for f.

The second observation is that the DARE from Theorem 3.11 (Yao's theorem) can be computed by an $\mathbf{NC}^1$ circuit, assuming that the key-shrinking gadget is computable in $\mathbf{NC}^1$. Hence, it suffices to implement the key-shrinking gadget in $\mathbf{NC}^1$. Such an implementation is given in [11] based on the existence of $\mathbf{NC}^1$-computable pseudorandom generators (PRGs) with minimal stretch (i.e., ones that stretch a k-bit seed into a $k + 1$ pseudorandom string). It was shown by [59] that such PRGs follow from the existence of $\mathbf{NC}^1$-computable one-way functions.[9] □

The existence of one-way functions in $\mathbf{NC}^1$ follows from most standard cryptographic assumptions including ones that are related to hardness of factoring, discrete logarithm, and lattice problems. Hence, the underlying assumption is very conservative. Still, the question of proving Theorem 4.2 based on a weaker assumption (e.g., the existence of arbitrary one-way functions) remains an interesting open question.

We also mention that the concrete degree and locality bounds (3 and 4) are known to be (at least) almost optimal, since most functions do not admit an encoding of degree 1 or locality 2. It is also known that perfectly private encodings require degree 3 and locality 4 ([66], see also [6,

[9] Indeed, the original statement in [11], which precedes [59], relied on the existence of minimal-stretch pseudorandom generators (PRGs) in $\mathbf{NC}^1$. See discussion in [6, Chapter 5].

Chapter 3]). The existence of statistical (or computational) encodings of locality 3 and degree 2 remains open.

Finally, we mention that stronger notions of constant-parallel-time encodings were studied in [12, 13].

4.2 Reducing the Online Complexity

As we saw in Theorem 3.11, every function f can be encoded by an online-efficient DARE $\hat{f}(x;r) = (\hat{f}_{\mathsf{off}}(r), \hat{f}_{\mathsf{on}}(x;r))$. In particular, the online communication (and computational complexity) is $O(|x| \cdot k)$, where k is the security parameter. Let us define the *online rate* of DARE to be the ratio

$$\frac{|\hat{f}_{\mathsf{on}}(x;r)|}{|x|}.$$

Using this terminology, the online rate achieved by Theorem 3.11 is proportional to the security parameter. In [16], it was shown that this can be significantly improved.

Theorem 4.3 (Online-succinct RE). *Under the decisional Diffie-Hellman assumption (DDH), the RSA assumption, or the Learning-with-Errors assumption (LWE), the class of polynomial-size circuits admits an RE with online rate* $1+o(1)$ *and with* $O(n^{1+\varepsilon})$ *online computation, for any* $\varepsilon > 0$.

Theorem 4.3 shows that, instead of communicating a cryptographic key per input bit (as in Theorem 3.11), it suffices to communicate n bits together with a single cryptographic key (i.e., $n+k$ bits instead of nk bits). A similar result holds for arithmetic mod-p formulas under the LWE assumption. (See [16].)

Proof idea. We briefly sketch the proof of Theorem 4.3. The starting point is a standard decomposable affine RE, such as the one from Theorem 3.11. Since this DARE is defined over the binary field, its online part can be viewed as a sequence of selection operations: For each i, we use the i-th bit of x to select a single "key" $K_i^{x_i}$ out of two k-bit vectors (K_i^0, K_i^1) that depend only on the randomness of the encoding. By the composition theorem, it suffices to encode this online procedure by an encoding such that most of its bits depend on the keys $(K_i^0, K_i^1)_{i\in[n]}$ while few of its bits (say $n+k$) depend on the input x. Put differently, we would like to have a compact way to reveal the selected keys.

Let us consider the following "riddle", which is a slightly simpler version of this problem. In the offline phase, Alice has n vectors $M_1, \ldots, M_n \in \{0,1\}^k$. She is allowed to send Bob a long encrypted version of these vectors. Later, in the online phase, she receives a bit vector $x \in \{0,1\}^n$. Her goal is to let Bob learn only the vectors which are indexed by x, i.e., $\{M_i\}_{i:x_i=1}$, while sending only a single message of length $O(n)$ bits (or even $n+k$ bits).[10]

Before solving the riddle, let us further reduce it to an algebraic version in which Alice wants to reveal a 0–1 linear combination of the vectors which are indexed by x. Observe that, if we can solve the new riddle with respect to nk-bit vectors $T = (T_1, \ldots, T_n)$, then we can solve the original riddle with k-bit vectors $(M_1, \ldots, M_n)$. This is done by placing the M_i's in the diagonal of T; i.e., T_i is partitioned to k-size blocks with M_i in the i-th block and zero elsewhere. In this case, Tx simply "packs" the vectors $\{M_i\}_{i:x_i=1}$.

[10] The main difference between the riddle and our actual encoding problem is that, in the latter case, the vector x itself should remain hidden; this gap can be bridged by permuting the pairs and randomizing the vector x; see [16] for details.

It turns out that the linear version of the riddle can be efficiently solved via the use of a symmetric-key encryption scheme with some (additive) homomorphic properties. Specifically, let (E, D) be a symmetric encryption scheme with both key homomorphism and message homomorphism as follows: A pair of ciphertexts $\mathsf{E}_k(x)$ and $\mathsf{E}_{k'}(x')$ can be mapped (without any knowledge of the secret keys) to a new ciphertext of the form $\mathsf{E}_{k+k'}(x+x')$. Given such a primitive, the answer to the riddle is easy: Alice encrypts each vector under a fresh key K_i and publishes the ciphertexts C_i. At the online phase, Alice sends the sum of keys $K_x = \sum K_i x_i$ together with the indicator vector x. Now Bob can easily construct $C = \mathsf{E}_{K_x}(Mx)$ by combining the ciphertexts indexed by x, and since K_x is known, Bob can decrypt the result. Intuitively, Bob learns nothing about a column M_j which is not indexed by x, as the online key K_x is independent of the j-th key. Indeed, the DDH- and LWE-based solutions provide (approximate) implementations of this primitive. (A somewhat different approach is used in the RSA-based construction.) □

A few comments regarding Theorem 4.3 are in order.

Reducing the online computational complexity. The online *computational* overhead of the encoding from Theorem 4.3 still has multiplicative dependence on the security parameter. It is possible to achieve linear computational complexity (e.g., $O(n + \text{poly}(k))$) based on very strong cryptographic assumptions (such as general-purpose obfuscation). Achieving a linear computational overhead based on weaker assumptions is an interesting open question.

Impossibility of DARE with constant rate. The encoding constructed in Theorem 4.3 is not a DARE. Specifically, while the theorem yields affine encodings (e.g., under DDH or LWE), it does not provide decomposable encodings. It turns out that this is inherent: constant-rate DAREs are impossible to achieve even for very simple functions [16].

Rate 1 is optimal. Theorem 4.3 provides an asymptotic online rate of 1. It is clear that this is optimal for functions with a long output (such as the identity function). However, the case of boolean functions is slightly more subtle. It is not hard to show that the online rate must be at least 1, if the online part is independent of the encoded function f. (As in the case of affine REs or in the model of [26].) Indeed, for every $i \in [n]$, consider the function $f_i(x) = x_i$, and assume that all these functions admit encodings $\hat{f}_i(x; r)$ whose online parts are all identical to $g(x; r)$. Using the appropriate decoders and the offline parts (which are independent of x), one can recover the value x_i from $g(x; r)$ for every i. The length of $g(x; r)$ must therefore be at least n.

It is not trivial to extend the lower bound to the more general case where the online computation $\hat{f}_{\mathsf{on}}(x; r)$ may fully depend on the code of f. This is especially true in the uniform model, where f has a succinct description (say as a Turing machine). Interestingly, it was observed in [16] that an encoding with online rate smaller than 1 would allow us to *compress* $f(x)$ in the following sense. Given x we can compute $\hat{f}_{\mathsf{on}}(x; r)$ for some fixed r and derive a short string $\hat{x}$ of length smaller than $|x|$ from which the value of $f(x)$ can be recovered. Moreover, the online efficiency of the encoding implies that $\hat{x}$ can be computed much faster than the time complexity of f. Following [60, 41], it was conjectured in [16] that some efficiently computable functions cannot be compressed, and so online rate smaller than 1 cannot be achieved. Later in [7], it was shown that the existence of incompressible functions can be based on relatively standard complexity-theoretic assumptions. A combination of these results yields the following theorem:

Theorem 4.4 ([16, 7])**.** *The class of polynomial-time computable functions does not admit an online-efficient randomized encoding with online rate smaller than* 1*, unless every function which is computable by a deterministic Turing machine in time* $2^{O(n)}$ *can be computed by subexponential-size nondeterministic circuits.*[11]

4.3 Reusable RE

Having minimized the online communication complexity of the encoding, we move on to a more ambitious goal: Improving the total computational complexity of $\hat{f}$. Recall that Theorem 3.11 yields an encoding whose offline complexity is proportional to the circuit size of f. A natural way to reduce this expensive offline cost is to amortize it over many independent inputs. For this, we need a *reusable randomized encoding.*

Definition 4.5 (ρ-Reusable randomized encoding)**.** *Let* $f : \{0,1\}^n \to \{0,1\}^\ell$ *be a function. We say that* $\hat{f} = (\hat{f}_{\mathsf{off}}, \hat{f}_{\mathsf{on}})$ *is a perfectly correct* (t, ε)*-private* ρ*-reusable randomized encoding of* f *if, for any* $i \leq \rho$*, the function*

$$\hat{f}^i(x^1, \ldots, x^i; r) := (\hat{f}_{\mathsf{off}}(r), \hat{f}_{\mathsf{on}}(x^1; r), \ldots, \hat{f}_{\mathsf{on}}(x^i; r))$$

is a perfectly correct, (t, ε)*-private encoding of the* i*-wise direct-product function*

$$f^i(x^1, \ldots, x^i) = (f(x^1), \ldots, f(x^i)).$$

Remark 4.6. *Observe that the definition is monotone: a* ρ*-reusable RE is always* $(\rho-1)$*-reusable, and a* 1*-reusable RE is simply an RE. Also note that a decoder of* $\hat{f}^i$ *can always be defined by applying the basic decoder of* $\hat{f}$ *to the offline part of the encoding and to each coordinate of* $\hat{f}^i$ *separately. Hence, reusability is essentially a strengthening of the* privacy *requirement. Finally, note that the above definition is static; i.e., the* i*-th input* x^i *is chosen independently of the encodings of the previous inputs. We will discuss stronger (adaptive) versions of security later in Section 4.5.*

As before, the definition naturally extends to infinite functions $f : \{0,1\}^* \to \{0,1\}^*$, and, more generally, to collections of functions via the use of an efficient compiler. Given a description of a function f with input length of n, the compiler should output (the descriptions of) an encoder $\hat{f}$, a decoder, and a $(t(n), \varepsilon(n))$-private $\rho(n)$-reusable simulator. The latter should be uniform over the number of encoded instances. Specifically, we assume that the simulator is given as a Turing machine that takes as an input a sequence of f-outputs and outputs the corresponding simulated encodings. By default, $\rho(n)$ and $t(n)$ should be larger than any polynomial and $\varepsilon(n)$ should be smaller than any polynomial.

Reusable REs with low online complexity. Reusable REs are nontrivial if their online complexity is smaller than the complexity of f. None of the encodings that we have seen so far satisfy this requirement, even for the case of 2-reusability. Recently, Goldwasser et al. [54] (building on the work of Gorbunov et al. [57]) showed that it is possible to construct nontrivial reusable REs whose online complexity depends only on the depth of the encoded function.[12]

[11] Similar assumptions are widely used in the complexity-theoretic literature. Such assumptions are considered to be strong, and yet plausible—their failure will force us to change our current view of the interplay between time, non-uniformity, and nondeterminism.

[12] In fact, [57] and [54] construct so-called *predicate encryption*, and *functional encryption* which can be viewed as a multi-user variant of randomized encoding.

Theorem 4.7 (Reusable REs)**.** *Assuming the intractability of Learning with Errors, there exists a reusable RE for efficiently computable functions whose online complexity depends only on the input length, output length, and circuit depth of the encoded function f.*

It follows that the amortized complexity of encoding f grows only with its circuit depth and input/output length.

4.3.1 Proof Idea of Theorem 4.7

The proof of Theorem 4.7 consists of two high-level steps. First, we prove the theorem for functions of a certain type, denoted as "conditional disclosure of secrets" (CDS) functions, and then we reduce the general case to the first, via the use of fully homomorphic encryption.

CDS functions. We begin with the definition of CDS functions. For a predicate $P : \{0,1\}^n \to \{0,1\}$, let g_P denote the non-boolean function which maps an n-bit input τ ("tag") and a k-bit string s ("secret") to the value (τ, c), where

$$c = \begin{cases} s & \text{if } P(\tau) = 1 \\ \perp & \text{otherwise.} \end{cases}$$

That is, g_P always reveals the value of the tag, but reveals the secret s only if the tag satisfies the predicate. In this sense, g_P conditionally discloses the secret.[13]

Rekeyable encryption. Our reusable REs for CDS are based on a special type of public-key encryption scheme with the following rekeying mechanism: For every pair of public keys (A, B) and a target public key C, there exists a special re-encryption key $T_{A,B\to C}$ that allows one to transform a pair of ciphertexts $(\mathsf{E}_A(s), \mathsf{E}_B(s))$ into a ciphertext $\mathsf{E}_C(s)$. Intuitively, the re-encryption key $T_{A,B\to C}$ should be "useless" given only one of the two encryptions $(\mathsf{E}_A(s), \mathsf{E}_B(s))$; For example, given $\mathsf{E}_A(s)$ and $T_{A,B\to C}$, it should be impossible to generate $\mathsf{E}_C(s)$. In fact, this ciphertext should be pseudorandom, and so even if the secret key that corresponds to C is known, s should be hidden. It is shown in [57] that such a rekeyable encryption scheme can be based on the Learning with Errors assumption, where the length of the ciphertext grows linearly with the number of iterated re-encryptions.[14]

From rekeyable encryption to reusable RE for CDS [57]. In order to encode the CDS function we would like to encrypt s in a way that is decryptable only if the (public) tag τ satisfies the predicate P. This should be done with a reusable offline part (whose complexity may grow with the complexity of P) while keeping the online part (which depends on s and τ) short, i.e., independent of the circuit size of P. The basic idea is as follows. For each wire i of the circuit that computes P, we will have a pair of public keys $(\mathsf{pk}_{i,0}, \mathsf{pk}_{i,1})$, labeled by zero and one. In the online part, we release the tag τ together with n ciphertexts, one for each input wire, where the i-th ciphertext is $\mathsf{E}_{\mathsf{pk}_{i,\tau_i}}(s)$, i.e., an encryption of s under the key of the i-th wire which is labeled by

[13] The term "conditional disclosure of secretes" originates from the work of [50] who considered an information-theoretic variant of this primitive. In a reusable setting, the resulting primitive is closely related to the notion of attribute-based encryption, cf. [68, 45].

[14] In the original terminology of [57], this is called two-to-one recoding (TOR).

τ_i. The offline part of the encoding consists of the secret key sk of the output wire which is labeled by 1, together with several re-encryption keys. Specifically, for each internal gate g, we include all four re-encryption keys that correspond to the semantic of the gate. Namely, if the keys of the left incoming wire are (A_0, A_1), the keys of the right incoming wire are (B_0, B_1), and the keys of the outgoing wire are (C_0, C_1), then we include the transformation key $T_{A_a, B_b \to C_{g(a,b)}}$, for every pair of bits $a, b \in \{0, 1\}$.

Given such an encoding, the decoder can propagate the ciphertexts from the inputs to the outputs, and compute for each wire i the ciphertext $\mathsf{E}_{\mathsf{pk}_{i,\sigma_i}}(s)$, where σ_i is the value that the public tag τ induces on the i-th wire. If the predicate is satisfied, the decoder learns an encryption of s under the 1-key of the output wire, and since the corresponding private key appears as part of the encoding, s is revealed. The security properties of the underlying encryption guarantee that s remains hidden when $P(\tau)$ is not satisfied. Moreover, the offline part is reusable and the online part grows linearly with the depth of the circuit that computes P, and is independent of the circuit size.

From CDS to general functions [54]. We move on to handle a general (non-CDS) function f. For this step, we employ a fully homomorphic encryption (FHE) which, by [34], can also be based on the intractability of Learning with Errors. In such an encryption, there exists a special Eval algorithm that maps a public key pk, a ciphertext $\mathsf{FHE}_{\mathsf{pk}}(x)$, and a circuit g to a new ciphertext $\mathsf{FHE}_{\mathsf{pk}}(g(x))$.

The encoding of f is based on the following basic idea. Encrypt the input x using FHE and include the ciphertext ct in the online part. In addition, provide (ideally in the offline part) some sort of conditional decryption mechanism that decrypts an FHE ciphertext ct' only if ct' is "legal" in the sense that it is the result of homomorphically applying f to $\mathsf{ct} = \mathsf{FHE}_{\mathsf{pk}}(x)$. Such an encoding would allow the decoder to transform $\mathsf{FHE}_{\mathsf{pk}}(x)$ to $\mathsf{FHE}_{\mathsf{pk}}(f(x))$ and then decrypt the value $f(x)$, without learning any other information on x.

To implement this approach, we should find a way to push most of the complexity of the conditional decryption mechanism to the offline phase. This is somewhat tricky since the condition depends on the online ciphertext ct. We solve the problem in two steps. First, consider the following naive (and inefficient) encoding:

$$(\mathsf{pk}, \mathsf{ct} = \mathsf{FHE}_{\mathsf{pk}}(x), \hat{D}(\mathsf{sk}, \mathsf{ct}'; r)), \tag{4}$$

where $\hat{D}$ is a DARE of the decryption algorithm D, and $\mathsf{ct}' = \mathsf{Eval}(\mathsf{pk}, f, \mathsf{ct})$. It is not hard to verify that this is indeed an encoding of $f(x)$. However, the encoding is not even online efficient. Indeed, ct' is computed in the online phase (based on ct) at a computational cost which depends on the circuit size of f.

In order to improve the efficiency, we take a closer look at $\hat{D}(\mathsf{sk}, \mathsf{ct}'; r))$. Since $\hat{D}$ is a DARE, we can decompose it to $\hat{D}_0(\mathsf{sk}; r)$ and to $\hat{D}_i(\mathsf{ct}'_i; r)$ for $i \in [\ell]$, where ℓ is the bit length of ct'. Using the "keys" terminology, for each bit of ct', there exists a pair of keys $(K_{i,0}, K_{i,1})$ where $K_{i,b} = \hat{D}_i(b; r)$, and the encoding $\hat{D}$ reveals, for each i, the key that corresponds to ct'_i. Namely, $K_{i,b}$ is exposed only if the i-th bit of $\mathsf{Eval}_f(\mathsf{ct})$ equals to b. This means that we can re-encode (4) by

$$(\mathsf{pk}, \mathsf{ct} = \mathsf{FHE}_{\mathsf{pk}}(x), \hat{D}_0(\mathsf{sk}; r), [g_{i,b}((\mathsf{pk}, \mathsf{ct}), K_{i,b})]_{i \in [\ell], b \in \{0,1\}}),$$

where $g_{i,b}((\mathsf{pk}, \mathsf{ct}), s)$ is the CDS function that releases the secret s only if the pair $(\mathsf{pk}, \mathsf{ct})$ satisfies the predicate "the i-th bit of $\mathsf{Eval}(\mathsf{pk}, f, \mathsf{ct})$ equals to b". Now, we can re-encode $g_{i,b}$ by a re-usable

CDS encoding $\hat{g}_{i,b}$, and, by the composition and concatenation lemmas, get a reusable encoding for f. In the online part of the resulting encoding we sample a public/private key pair for the FHE $(\mathsf{pk}, \mathsf{sk})$ and compute $\mathsf{ct} = \mathsf{FHE}_{\mathsf{pk}}(x), \hat{D}_0(\mathsf{sk}; r)$ together with the online parts of $\hat{g}_{i,b}$. Overall, the online complexity is independent of the circuit size of f. (The dependence on the circuit depth is inherited from that of the CDS encoding.) This completes the proof of Theorem 4.7. □

4.4 Reducing the Computational Complexity of REs

Theorem 4.7 implies that the amortized complexity of encoding f grows only with its parallel complexity (i.e., circuit depth) and input/output length. This result does not leave much room for improvements. Still, one can ask whether it is possible to obtain an encoding whose complexity is independent of the complexity of the encoded function f in a non-amortized setting, i.e., even for a single use of $\hat{f}$. This question makes sense only if the encoded function f has a succinct representation whose length is independent of its time complexity. Indeed, let us move to a uniform setting and assume that f can be computed by a Turing machine (whose description length is independent of its time complexity). We will show that, under sufficiently strong cryptographic assumptions, one can obtain an encoding whose complexity is completely independent of the time complexity of f and depends only on the input/output length of f. We refer to such an encoding as a *fast RE*. The construction will employ a powerful cryptographic tool: the general-purpose program obfuscator.

Obfuscators. General-purpose program obfuscation allows us to transform an arbitrary computer program into an "unintelligible" form while preserving its functionality. Syntactically, an obfuscator for a function family $\mathcal{F} = \{f_K\}$ is a randomized algorithm that maps a function $f_K \in \mathcal{C}$ (represented by an identifier K) into a "program" $[f] \in \{0,1\}^*$. The obfuscated program should preserve the same functionality as f_K while hiding all other information about f_K. The first property is formalized via the existence of an efficient universal evaluation algorithm Eval which, given an input x and an obfuscated program $[f]$, outputs $f_K(x)$. The second property has several different formulations. We will rely on *indistinguishability obfuscation* (iO), which asserts that, given a pair of functionally equivalent functions f and f', it is hard to distinguish between their obfuscated versions $[f]$ and $[f']$.

Obfuscators Versus REs. As a cryptographic primitive, obfuscators are stronger than REs. Intuitively, an obfuscator provides the adversary the ability to compute the function by herself, while REs provide only access to encodings that were computed by the encoder (which is out of the adversary's control). Indeed, given an iO for a function class $\mathcal{F}$, we can easily get an RE for the same class by obfuscating the constant function $g = f(x)$ (x is hardwired) and letting the simulator $\mathsf{Sim}(y)$ be an obfuscated version of the constant function $h = y$. We can therefore trivially get fast REs based on "fast" obfuscators whose complexity is independent of the time complexity of the obfuscated function (and depends only on its description length). The following theorem provides a stronger result: Fast REs (for Turing machines) can be based on *standard* obfuscators whose complexity is proportional to the circuit complexity of the obfuscated function.

Theorem 4.8 (Fast RE from circuit obfuscators [28, 37, 72]). *Assuming the existence of iO for circuits and the existence of one-way functions, the class of polynomial-time computable functions*

P *admits an RE with computational complexity of* $\mathrm{poly}(n,\ell)$*, where* n *and* ℓ *denote the input and output length of the encoded function.*

Fast REs were first constructed by Goldwasser et al. [53] based on a stronger assumption. Bitansky et al. [28] and Canetti et al. [37] concurrently relaxed the assumption to iO, but their results yield REs whose complexity grows with the space complexity of f. The dependence on the space complexity was later removed by Koppula et al. [72]. We mention that Theorem 4.8 was further used to obtain fast obfuscators. (For more on the relation between REs and obfuscators see [74].) The proof of Theorem 4.8 is beyond the scope of this paper. The interested reader is referred to the original papers.

Optimality of Theorem 4.8. Theorem 4.8 provides REs whose complexity depends on the input length and on the output length of the encoded function. We already saw in Theorem 4.4 that one cannot go below the input length. The following theorem (from [16]) shows that one cannot go below the output length either:

Theorem 4.9 (Communication of RE is larger than the output length)**.** *Assuming that one-way functions exist, for every constant* c *there exists a function* $f : \{0,1\}^n \to \{0,1\}^{n^c}$ *such that every* $(n^{\omega(1)}, 1/3)$*-private RE of* f *has communication complexity of at least* n^c *bits. Furthermore, there are at least* n^c *bits in the output of the simulator* $\mathsf{Sim}(y)$ *that depend on the input* y *(as opposed to the randomness).*

Note that some complexity-theoretic assumption is necessary for Theorem 4.9. In particular, if, say $\mathbf{P} = \mathbf{NP}$, then one can obtain an encoding with total complexity n. Indeed, this can be done by letting $\hat{f}(x;r) = x'$, where x' is a random sibling of x under f, and taking the decoder to be $\mathsf{Dec}(x') = f(x')$, and the simulator to be $\mathsf{Sim}(y) = x'$, where x' is a random preimage of y. If $\mathbf{P} = \mathbf{NP}$, then this encoding can be implemented efficiently.[15]

Proof. Fix some constant c, and let $f : \{0,1\}^n \to \{0,1\}^\ell$ be a pseudorandom generator with output length $\ell = n^c$. (The existence of such a pseudorandom generator follows from the existence of one-way functions [61].) It suffices to prove the "furthermore" part, as the online complexity of the simulator lower-bounds the communication complexity of the encoding. Let $\hat{f}(x;r)$ be an RE of f with decoder Dec and simulator Sim such that the number of bits of $\mathsf{Sim}(y)$ that depend on y is smaller than ℓ. Then, we distinguish the output of f from a truly random string via the following test: Given a string $y \in \{0,1\}^\ell$, we accept if and only if the outcome of $\mathsf{Dec}(\mathsf{Sim}(y))$ is equal to y.

First we claim that, when y is random, the test accepts with probability at most $\frac{1}{2}$. Indeed, fix some value r for the randomness of the simulator and some value d for the randomness of the decoder. Then the image of $\mathsf{Sim}(y;r) = (z_r, \mathsf{Sim}_{\mathsf{on}}(y;r))$ can take at most $2^{\ell-1}$ values, and therefore the decoder $\mathsf{Dec}(\cdot;s)$ recovers y successfully for at most half of all y's in $\{0,1\}^\ell$.

On the other hand, if y is in the image of f, the test accepts with probability at least $2/3-\mathrm{neg}(n)$. Indeed, let x be a preimage of y, then by definition $\mathsf{Dec}(\hat{f}(x;r))$ outputs $y = f(x)$ with probability 1. Since $\hat{f}(x;r)$ is $(t, 1/3)$-indistinguishable from $\mathsf{Sim}(f(x))$, it follows that $\mathsf{Dec}(\mathsf{Sim}(y)) = y$ with probability at least $2/3 - \mathrm{neg}(n)$. □

[15] In fact, if one-way functions do not exist, then it can be shown that the aforementioned encoding achieves a relaxed version of privacy (against uniform adversaries).

Remark 4.10 (Inefficient simulation)**.** *Theorem 4.9 exploits the efficiency of the simulator to "compress" an "incompressible" source. This argument does not hold if we allow* inefficient *simulation. The resulting notion corresponds to the following indistinguishability-based definition of privacy. The encodings $\hat{f}(x)$ and $\hat{f}(x')$ should be indistinguishable for any pair of inputs x and x' which are mapped by f to the same output, i.e., $f(x) = f(x')$. Indeed, based on iO, one can get fast REs with inefficient simulation whose complexity grows only with the input length of the encoded function (see [28, 37, 72]).*

4.5 On Adaptive Security

The standard security definition of REs can be captured by the following game: (1) The challenger secretly tosses a random coin $b \overset{R}{\leftarrow} \{0,1\}$; (2) the adversary chooses an input x, submits it to the challenger, and gets as a result the string $\hat{y}$ which, based on the secret bit b, is either sampled from the encoding $\hat{f}(x;r)$ or from the simulator $\mathsf{Sim}(f(x))$. At the end, the adversary outputs his guess b' for the bit b. The security of REs says that t-bounded adversaries cannot win the game (guess b) with probability better than $\frac{1}{2} + \varepsilon/2$.

In some scenarios (e.g., the online/offline setting or the reusable setting) it is natural to consider an *adaptive* version of this game in which the adversary chooses its input x based on the previous part of the encodings. Let us focus on the simpler online/offline setting. Syntactically, this requires an online/offline simulator $\mathsf{Sim}(y;r) = (\mathsf{Sim}_{\mathsf{off}}(r); \mathsf{Sim}_{\mathsf{on}}(x;r))$ whose offline part does not depend on its input $f(x)$, and has the same length as the offline part of the encoding. Adaptive security can be defined as follows:

Definition 4.11 (Adaptively secure RE [25])**.** *Let f be a function and $\hat{f}(x;r) = (\hat{f}_{\mathsf{off}}(r), \hat{f}_{\mathsf{on}}(x;r))$ be a perfectly correct RE with decoder* Dec *and online/offline simulator* $\mathsf{Sim}(y;r) = (\mathsf{Sim}_{\mathsf{off}}(r), \mathsf{Sim}_{\mathsf{on}}(y;r))$. *We say that $\hat{f}$ is (t,ε)* adaptively private *if every t-bounded adversary $\mathcal{A}$ wins the following game with probability at most $\frac{1}{2} + \varepsilon$:*

1. *The challenger secretly tosses a random coin $b \overset{R}{\leftarrow} \{0,1\}$, chooses randomness r, and outputs*
$$\hat{y}_{\mathsf{off}} = \begin{cases} \hat{f}_{\mathsf{off}}(r) & \text{if } b = 1, \\ \mathsf{Sim}_{\mathsf{off}}(r) & \text{if } b = 0. \end{cases}$$

2. *Based on $\hat{y}_{\mathsf{off}}$, the adversary $\mathcal{A}$ chooses an input x, submits it to the challenger, and gets as a result the string*
$$\hat{y}_{\mathsf{on}} = \begin{cases} \hat{f}_{\mathsf{on}}(x;r) & \text{if } b = 1, \\ \mathsf{Sim}_{\mathsf{on}}(f(x);r) & \text{if } b = 0. \end{cases}$$

3. *At the end, the adversary outputs his guess b' and wins if $b' = b$.*

It turns out that the online complexity of adaptively secure REs must grow with the output length of the encoded function and thus cannot be online efficient. Indeed, this follows directly from Theorem 4.9.

Corollary 4.12 (Adaptive security requires long online communication [16])**.** *Assuming one-way functions exist, for every constant c there exists a function $f : \{0,1\}^n \to \{0,1\}^{n^c}$ such that every RE of f has online communication complexity of at least n^c bits.*

Proof. By Theorem 4.9, there exists a function f for which the online part of the simulator must be longer than n^c. Privacy ensures that the online communication complexity of $\hat{f}$ satisfies the same bound. □

Remark 4.13 (Adaptive security with inefficient simulation)**.** *The output length limitation does not seem to apply to inefficient simulators (just like in Remark 4.10), and so, in this case, one may hope to achieve online efficiency. In fact, such a construction of adaptive REs with inefficient simulation easily yields efficiently simulatable adaptive REs: To encode $f : \{0,1\}^n \to \{0,1\}^s$ encode the related function*

$$g(x,b,y) = \begin{cases} f(x) & \text{if } b = 0 \\ y & \text{otherwise} \end{cases},$$

via an adaptive encoding $\hat{g}(x,y,b;r)$ with inefficient simulation. Now, the function $\hat{g}(x,y,b;r)$ with b fixed to zero and y fixed to, say, the all zero string, forms an adaptive encoding for f with the (efficient) simulator $\mathsf{Sim}(y;r) := \hat{g}_{\mathsf{off}}(r), \hat{g}_{\mathsf{on}}(x,y,b;r))$ with $b = 1$ and $x = 0^n$. Note that the input length of g equals the sum of the input and output lengths of f. Hence, the above transformation incurs an overhead which is (at least) as large as the output length, as expected due to Corollary 4.12.

Constructions. Any standard RE can be viewed as an adaptive one by including all the encoding in the online part. Yao's construction (Theorem 3.11) therefore provides an adaptive RE based on one-way functions whose online part depends on the circuit of the encoded function f. An encoding whose online part depends only on the size of the circuit was constructed by Bellare et al. [25]. This is done by taking Yao's encoding $\hat{f} = (\hat{f}_{\mathsf{off}}, \hat{f}_{\mathsf{on}})$, encrypting its offline part via a one-time pad, and releasing the key in the online part of the encoding. This yields an encoding whose online complexity is proportional to the circuit size of f based on one-way functions.

Using a (programmable) random oracle, it is possible to "compress" the key of the one-time pad and get an online-efficient adaptive encoding. Alternatively, such an encoding can be obtained via "complexity leveraging". Indeed, any (t, ε) adaptive attack against an encoding $\hat{f}$ with offline communication complexity of s immediately translates into a $(t, \varepsilon \cdot 2^{-s})$ static attack against $\hat{f}$ by guessing the offline part and calling the adaptive adversary. Hence, a sufficiently strong static encoding (e.g., Yao's construction instantiated with subexponential secure encryption) yields an adaptively secure RE.

In the standard model, Theorem 4.8 provides an adaptively-secure encoding with an optimal total complexity (proportional to the input and output length) based on the existence of general-purpose indistinguishability obfuscation. Based on general one-way functions, Hemenway et al. [62] constructed an adaptive encoding whose online complexity is proportional to the width of the encoded circuit (or exponential in its depth). Achieving an online complexity which depends only on the input and output lengths of f (based on one-way functions) remains an interesting open problem.

5 Applications

The ability to encode complex functions by simple ones is extremely useful. In the next subsections we will demonstrate several interesting ways in which this tool can be employed. We consider the archetypal cryptographic setting where Alice and Bob wish to accomplish some computational goal (e.g., a functionality f) in the presence of an adversary. We will see that REs can be beneficial when they are applied to each component of this system: to the functionality, to the honest parties, and even to the adversary.

5.1 Encoding the Functionality

Delegating computations. Suppose that Bob is a computationally weak device (client) who wishes to compute a complex function f on an input x. Bob is too weak to compute f on his own, and so he delegates the computation to a computationally strong server Alice. Since Bob does not trust Alice, he wishes to guarantee the following: (1) *secrecy*: Alice should learn nothing on the input x; and (2) *verifiability*: Bob should be able to verify the correctness of the output (i.e., a cheating Alice should be caught w.h.p.). Similar problems have been extensively studied in various settings, originating from the early works on interactive proofs, program checking, and instance-hiding schemes.

Let us start with secrecy and consider a variant where both parties should learn the output $f(x)$ but x should remain private. As we saw in the introduction (Example 1.3), a randomized encoding $\hat{f}$ immediately solves this problem via the following single-round protocol: Bob selects private randomness r, computes $\hat{f}(x;r)$, and sends the result to Alice, who applies the recovery algorithm and outputs the result. The privacy of the RE guarantees that Alice learns nothing beyond $f(x)$. We refer to this protocol as the *basic RE protocol*. Jumping ahead, we note that the protocol has a nontrivial correctness guarantee: even if the server Alice deviates from the protocol and violates correctness, she cannot force an erroneous output which violates privacy; that is, it is possible to simulate erroneous outputs solely based on the correct outputs.

It is not hard to modify the basic protocol and obtain full secrecy: instead of encoding f, encode an encrypted version of f. Namely, define a function $g(x,s) = f(x) \oplus s$, where s plays the role of a one-time pad (OTP), and apply the previous protocol as follows: Bob uniformly chooses the pad s and the randomness r, and sends the encoding $\hat{g}(x,s;r)$ of g to Alice, who recovers the result $y = g(x,s) = f(x) \oplus s$, and sends it back to Bob. Finally, Bob removes the pad s and terminates with $f(x)$. (See [11] for more details.)

Achieving verifiability is slightly more tricky. The idea, due to [14], is to combine an RE with a private-key signature scheme (also known as message authentication code or MAC) and ask the server to sign the output of the computation under the client's private key. Here the privacy property of the RE will be used to hide the secret key. Specifically, given an input x, Bob asks Alice to compute $y = f(x)$ (via the previous protocol) and, in addition, to generate a signature on $f(x)$ under a private key k which is chosen randomly by the client. The latter request is computed via the basic RE protocol that hides the private key from Alice. More precisely, Bob, who holds both x and k, invokes an RE protocol in which both parties learn the function $g(x,k) = \mathtt{MAC}_k(f(x))$. Bob then accepts the answer y if and only if the result of the protocol is a valid signature on y under the key k. (The latter computation is typically cheap.) The soundness of the protocol follows by showing that a cheating Alice, which fools Bob to accept an erroneous $y^* \neq f(x)$, can be used to either break the privacy of the RE or to forge a valid signature on a new message. For this argument to hold, we crucially rely on the ability to simulate erroneous outputs based on the correct outputs.

The main advantage of this approach over alternative solutions is the ability to achieve good soundness with low computational overhead; For example, a soundness error of $2^{-\tau}$ increases the communication by an additive overhead of τ, whereas the overhead in competing approaches is multiplicative in τ. (See [14] for a more detailed comparison.) Instantiating these approaches with known constructions of REs leads to protocols with highly efficient clients; For example, Theorems 4.1 and 4.2 yield an $\mathbf{NC^0}$ client for either log-space functions or poly-time functions depending on the level of security needed (information-theoretic or computational). In the computational setting, we can use online-efficient REs (Theorem 3.11) to get a client whose online computational complexity does not grow with the complexity of f, at the expense of investing a lot of computational resources in a preprocessing phase before seeing the actual input x.[16] Moreover, we can achieve optimal online communication via the use of online-succinct REs (Theorem 4.3), amortize the cost of the offline phase by employing reusable REs (Theorem 4.7), or even avoid the offline cost at all via the use of fast REs (Theorem 4.8).

We also mention that REs can achieve other related properties such as *correctability* [14]: i.e., Bob is able to *correct* Alice's errors as long as Alice is somewhat correct with respect to a predefined distribution over the inputs. In the latter case, REs yield $\mathbf{NC^0}$ correctors for log-space computations, strengthening the results of [52].

Secure computation [66]. Let us move on to a more general setting where the roles of Alice and Bob are symmetric and neither of them is computationally weak. The main observation is that, instead of securely computing f, it suffices to securely compute the randomized encoding $\hat{f}(x;r)$. Indeed, if Alice and Bob learn a sample from $\hat{f}(x;r)$ then they can locally recover the value of $f(x)$ and nothing else. In other words, the task of securely computing f *reduces* to the task of securely computing a simpler randomized functionality $\hat{f}(x;r)$. As protocol designers, we get a powerful tool which allows us to construct a complex interactive object (protocol) by arguing about a simpler *noninteractive* object (RE).

This paradigm, which was introduced in [66] (and motivated the original definition of REs), yields several new results in the domain of secure computation. As an example, if the algebraic degree of $\hat{f}$ is constant, then it can be computed in a constant number of rounds [27, 39]. By instantiating this approach with known perfect-RE constructions (Theorem 4.1), we derive constant-round protocols for boolean or arithmetic log-space functions with information-theoretic security. In the computational setting, constant-degree REs (Theorem 4.2) yield a new constant-round protocol for poly-time functions, providing an alternative construction to the classical protocol of [21].[17]

The above paradigm was implicitly used by Yao to prove the feasibility of general secure computation based on oblivious transfer (OT) [85]. Indeed, consider a two-party functionality $f(x,y)$, where x is given to Alice and y is given to Bob. A DARE $\hat{f}$ for f can be written as $\hat{f}(x,y;r) = (\hat{f}_1(x_1,y;r),\ldots,\hat{f}_n(x_n,y;r))$, and therefore it can be privately computed (with semi-honest security) by using n calls to an OT functionality. For each $i \in [n]$, Bob prepares a pair of "keys": $K_{i,0} = \hat{f}_i(0,y;r)$ and $K_{i,1} = \hat{f}_i(0,y;r)$, and lets Alice choose the key K_{i,x_i} that corresponds to her input by using the OT. After collecting all the n keys, Alice can recover the output by applying the decoder. The existence of DARE for any efficiently computable function f (Theorem 3.11) therefore gives a direct noninteractive reduction from securely computing f to OT. Other classical completeness results (e.g., for the multiparty case [51] and for the malicious case [69]) can also be proved using the above paradigm.

[16] The standard privacy of REs guarantees security only if the input x is chosen independently of the offline part. Adaptively secure REs can be used to deal with the case where the input is (adversarially) chosen based on the offline part.

[17] The RE-based solution requires a slightly stronger assumption—one-way function computable in log-space rather in poly-time—but can also lead to efficiency improvements, cf. [41].

5.2 Encoding the Primitive: Parallel Cryptography

Suppose now that we already have an implementation of some cryptographic protocol. A key observation made in [10] is that we can "simplify" some of the computations in the protocol by replacing them with their encodings. Consider, for example, the case of public-key encryption: Alice publishes a public/private key pair $(\mathsf{pk}, \mathsf{sk})$; Bob uses the public key pk and a sequence of random coins s to "garble" a message m into a ciphertext $c = \mathsf{E}(\mathsf{pk}, m, s)$; Finally, Alice recovers m by applying the decryption algorithm to the ciphertext $\mathsf{D}(\mathsf{sk}, c)$. Suppose that Bob sends an encoding of his ciphertext $\widehat{\mathsf{E}}(\mathsf{pk}, m, s; r)$ instead of sending c. This does not violate semantic security as all the information available to an adversary in the modified protocol can be emulated by an adversary who attacks the original protocol (thanks to the simulator of the RE). On the other hand, Alice can still decrypt the message: first she recovers the original ciphertext (via the recovery algorithm) and then she applies the original decryption algorithm. As a result, we "pushed" the complexity of the sender (encryption algorithm) to the receiver (decryption algorithm).

By employing REs with some additional properties, it is possible to prove similar results for many other cryptographic protocols (e.g., one-way functions, pseudorandom generators, collision-resistance hash functions, signatures, commitments, and zero-knowledge proofs) and even information-theoretic primitives (e.g., ε-biased generators and randomness extractors). In the case of stand-alone primitives (e.g., one-way functions and pseudorandom generators) there is no receiver and so the gain in efficiency comes for "free".

Being security preserving, REs give rise to the following paradigm. In order to construct some cryptographic primitive $\mathcal{P}$ in some low-complexity class $\mathcal{WEAK}$, first encode functions from a higher-complexity class $\mathcal{STRONG}$ by functions from $\mathcal{WEAK}$; then, show that $\mathcal{P}$ has an implementation f in $\mathcal{STRONG}$, and finally replace f by its encoding $\hat{f} \in \mathcal{WEAK}$ and obtain a low-complexity implementation of $\mathcal{P}$. This approach was used in [10, 11, 13] to obtain cryptographic primitives in $\mathbf{NC^0}$ and even in weaker complexity classes. The fact that REs preserve cryptographic hardness has also been used to reduce the complexity of cryptographic *reductions* [10, 11] and to reduce the complexity of complete problems for subclasses of statistical zero-knowledge [42].

5.3 Encoding the Adversary: Key-Dependent Security

Key-dependent message (KDM) secure encryption schemes [36, 29] provide secrecy even when the attacker sees encryptions of messages related to the secret key sk. Namely, we say that an encryption is KDM secure with respect to a function class $\mathcal{F}$ if semantic security holds even when the adversary can ask for an encryption of the message $f(\mathsf{sk})$ where f is an arbitrary function in $\mathcal{F}$. Several constructions are known to achieve KDM security for simple linear (or affine) function families [31, 9, 32]. To improve this situation, we would like to have an *amplification* procedure which starts with an $\hat{\mathcal{F}}$-KDM secure encryption scheme and boosts it into an $\mathcal{F}$-KDM secure scheme, where the function class $\mathcal{F}$ should be richer than $\hat{\mathcal{F}}$. It was shown in [33, 19] that a strong form of amplification is possible, provided that the underlying encryption scheme satisfies some special *additional* properties. Below we show how to use REs in order to achieve a *generic* KDM amplification theorem [2].

Let $f(x)$ be a function and let us view the encoding $\hat{f}(x; r)$ as a *collection* of functions $\hat{\mathcal{F}} = \left\{\hat{f}_r(x)\right\}_r$, where each member of the collection corresponds to some possible fixing of the randomness r, i.e., $\hat{f}_r(x) = \hat{f}(x; r)$. Now suppose that our scheme is KDM secure with respect to the family $\hat{\mathcal{F}}$, and we would like to immunize it against the (more complicated) function f. This can be easily achieved by modifying the encryption scheme as follows: To encrypt a message m we first translate it into the $\hat{f}$-encoding by applying the RE simulator $\mathsf{Sim}(m)$, and then encrypt the result under the original encryption scheme. Decryption is done by applying the original decryption algorithm, and then applying the decoding algorithm Dec to translate the result back to its original form. Observe that an encryption of $f(\mathsf{sk})$ in the new scheme is the same as an encryption of $S(f(\mathsf{sk})) = \hat{f}(\mathsf{sk}; r)$ under the original scheme. Hence, a KDM query for f in the new scheme is emulated by an old KDM query for a *randomly chosen* function $\hat{f}_r$. It follows that the KDM security of the new scheme with respect to f reduces to the KDM security of the original scheme with respect to $\hat{\mathcal{F}}$.

This idea easily generalizes to the case where, instead of a single function f, we have a class of functions $\mathcal{F}$ which are all encoded by functions in $\hat{\mathcal{F}}$. Moreover, the simple structure of the reduction (i.e., a single KDM query of the new scheme translates to a single KDM query of the original scheme) allows one to obtain a strong amplification theorem which is insensitive to the exact setting of KDM security, including the symmetric-key/public-key setting, the cases of chosen-plaintext/chosen-ciphertext attacks, and the case of multiple keys. Using known constructions of REs (Theorem 3.11), we can amplify KDM security with respect to linear functions (or even bit projections) into functions computable by circuits of arbitrary fixed polynomial size (e.g., n^2).

Interestingly, here we essentially encoded the adversary and used the simulator in the construction rather than in the proof. A similar approach ("encoding the adversary") was used by [20] to obtain strong impossibility results for concurrent non-malleable secure computation.

6 Summary and Suggestions for Further Reading

Randomized encodings form a powerful tool with a wide range of applications. Since Yao's original construction in the early 1980s, the complexity of REs has been significantly improved, especially in the last decade. Many of these improvements are based on encryption schemes which suggest various forms of *homomorphism*. (This includes Theorems 3.12, 4.3, 4.7, and 4.8 as well as works which are not covered here, cf. [49, 4, 30].) It is interesting to find out whether this use of homomorphism is necessary. In particular, we ask:

> Is it possible to achieve highly efficient REs (such as the ones constructed in Section 4) based on weaker assumptions, e.g., one-way functions?

An even more basic question is to understand the power of information-theoretic REs:

> Are there efficiently computable perfectly secure DAREs for polynomial-size circuits?

The question is interesting also for other notions of simplicity such as constant algebraic degree or constant output locality. Coming up with a positive result, or developing tools for proving nontrivial lower bounds, is an important problem in the context of information-theoretic cryptography.

Another interesting direction for future research is to explore stronger forms of security for REs. This direction is being extensively pursued by the study of primitives such as functional encryption (and its variants) and even program obfuscators. Indeed, these primitives can be viewed as REs which offer richer functionalities and stronger security guarantees (cf. [83, 56]).

6.1 Suggestions for Further Reading

We end by briefly mentioning some of the aspects of REs that were omitted from this paper.

Encoding RAM computation. A relatively recent line of work, initiated by Lu and Ostrovsky [78], studies the possibility of encoding stateful computations modeled as RAM programs. Roughly speaking, such a computation is defined by a (large) memory D, a (typically small) program P, and an input x. The computation $P^D(x)$ generates an output y and updates the memory D (typically only at a small number of locations). The goal is to obtain a *stateful RE* that initializes the memory $\hat{D}$ based on D, and then, given a program/input pair (P, x), generates an encoding $(\hat{P}, \hat{x})$ that together with $\hat{D}$ forms an encoding of the original computation. The complexity of $(\hat{P}, \hat{x})$ is allowed to depend on the time complexity of P and the input length x but should have only a minor (say poly-logarithmic) dependence on the size of the memory. The security definition allows one to sequentially execute many programs, and so the initialization cost of the memory is amortized. Following [78], several constructions of garbled RAM have been presented, including ones which are based solely on one-way functions [48, 44].

Concrete efficiency and practical implementations. Starting with the work of Malkhi et al. [79], computational DAREs (garbled circuits) were implemented on various platforms, and their concrete efficiency was extensively studied. These constructions are typically formalized under the garbling-schemes framework of Bellare, Hoang, and Rogaway [26], and offer a wide range of concrete tradeoffs between computational complexity, communication complexity, and intractability assumptions (cf. [80, 77, 71, 81, 63, 73, 24, 70, 87]). A partial summary of these results can be found in [82].

Communication complexity treatment. Feige, Kilian, and Naor [43] and Ishai and Kushilevitz [65] studied REs in a communication complexity framework. In this model, the input $x \in \{0,1\}^n$ is partitioned between several parties (typically two or n) who also have access to shared randomness r. Each party should send a single message $M_i(x_i; r)$, based on her input x_i and on the randomness r, to a referee, who should be able to recover the value of $f(x)$ and nothing else. All parties are computationally unbounded, and the goal is to minimize the communication. In the two-party setting, such a *private simultaneous message* (PSM) protocol naturally defines a *two-decomposable* RE $\hat{f}(x; r) = (M_1(x_1; r), M_2(x_2; r))$, and in the multiparty setting it corresponds to a DARE $\hat{f}(x; r) = (M_1(x_1; r), \ldots, M_n(x_n; r))$. The best known protocols (for an arbitrary function) appear in [23]. This model and its variants (e.g., for conditional disclosure of secrets) are further studied in [68, 45, 18, 22].

Complexity-theoretic treatment. From a complexity-theoretic perspective, the notion of "simple" computation may be associated with probabilistic polynomial-time computation. Hence, the question of encoding a complex function by a "simpler" one naturally corresponds to polynomial-time computable encodings. We can now ask which functions can be statistically encoded by efficiently computable encodings, and define a corresponding complexity class **SRE**. To avoid trivialities, we allow the decoder to be inefficient. Going back to Example 1.3, this corresponds to the setting where a weak (polynomial-time) client who holds an input x wishes to release the value $f(x)$ to a computationally unbounded server without leaking the value of x. It is not hard to show that the resulting complexity class **SRE** contains **BPP** and is contained in the class of languages that admit statistical zero-knowledge protocols (**SZK**) [6]. The exact relation between these three classes is further studied in [1] and [17].

References

[1] S. Agrawal, Y. Ishai, D. Khurana, and A. Paskin-Cherniavsky. Statistical randomized encodings: A complexity theoretic view. In M. M. Halldórsson, K. Iwama, N. Kobayashi, and B. Speckmann, editors, *ICALP 2015, Part I*, volume 9134 of *LNCS*, pages 1–13. Springer, Heidelberg, July 2015.

[2] B. Applebaum. Key-dependent message security: Generic amplification and completeness. In K. G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 527–546. Springer, Heidelberg, May 2011.

[3] B. Applebaum. Randomly encoding functions: A new cryptographic paradigm - (invited talk). In S. Fehr, editor, *ICITS 11*, volume 6673 of *LNCS*, pages 25–31. Springer, Heidelberg, May 2011.

[4] B. Applebaum. Garbling XOR gates "for free" in the standard model. In A. Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 162–181. Springer, Heidelberg, Mar. 2013.

[5] B. Applebaum. Bootstrapping obfuscators via fast pseudorandom functions. In P. Sarkar and T. Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 162–172. Springer, Heidelberg, Dec. 2014.

[6] B. Applebaum. *Cryptography in Constant Parallel Time.* Information Security and Cryptography. Springer, 2014.

[7] B. Applebaum, S. Artemenko, R. Shaltiel, and G. Yang. Incompressible functions, relative-error extractors, and the power of nondeterministic reductions (extended abstract). In *30th Conference on Computational Complexity, CCC 2015, June 17-19, 2015, Portland, Oregon, USA*, pages 582–600, 2015.

[8] B. Applebaum, J. Avron, and C. Brzuska. Arithmetic cryptography: Extended abstract. In T. Roughgarden, editor, *ITCS 2015*, pages 143–151. ACM, Jan. 2015.

[9] B. Applebaum, D. Cash, C. Peikert, and A. Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In S. Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 595–618. Springer, Heidelberg, Aug. 2009.

[10] B. Applebaum, Y. Ishai, and E. Kushilevitz. Cryptography in NC^0. In *45th FOCS*, pages 166–175. IEEE Computer Society Press, Oct. 2004.

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

[12] B. Applebaum, Y. Ishai, and E. Kushilevitz. Cryptography with constant input locality. In A. Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 92–110. Springer, Heidelberg, Aug. 2007.

[13] B. Applebaum, Y. Ishai, and E. Kushilevitz. Cryptography by cellular automata or how fast can complexity emerge in nature? In A. C.-C. Yao, editor, *ICS 2010*, pages 1–19. Tsinghua University Press, Jan. 2010.

[14] B. Applebaum, Y. Ishai, and E. Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In S. Abramsky, C. Gavoille, C. Kirchner, F. Meyer auf der Heide, and P. G. Spirakis, editors, *ICALP 2010, Part I*, volume 6198 of *LNCS*, pages 152–163. Springer, Heidelberg, July 2010.

[15] B. Applebaum, Y. Ishai, and E. Kushilevitz. How to garble arithmetic circuits. In R. Ostrovsky, editor, *52nd FOCS*, pages 120–129. IEEE Computer Society Press, Oct. 2011.

[16] B. Applebaum, Y. Ishai, E. Kushilevitz, and B. Waters. Encoding functions with constant online rate or how to compress garbled circuits keys. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 166–184. Springer, Heidelberg, Aug. 2013.

[17] B. Applebaum and P. Raykov. On the relationship between statistical zero-knowledge and statistical randomized encodings. *Electronic Colloquium on Computational Complexity (ECCC)*, 15, 2015. To appear in CRYPTO 2016.

[18] B. Applebaum and P. Raykov. From private simultaneous messages to zero-information Arthur-Merlin protocols and back. LNCS, pages 65–82. Springer, Heidelberg, 2016.

[19] B. Barak, I. Haitner, D. Hofheinz, and Y. Ishai. Bounded key-dependent message security. In H. Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 423–444. Springer, Heidelberg, May 2010.

[20] B. Barak, M. Prabhakaran, and A. Sahai. Concurrent non-malleable zero knowledge. In *47th FOCS*, pages 345–354. IEEE Computer Society Press, Oct. 2006.

[21] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513. ACM Press, May 1990.

[22] A. Beimel, A. Gabizon, Y. Ishai, and E. Kushilevitz. Distribution design. In M. Sudan, editor, *ITCS 2016*, pages 81–92. ACM, Jan. 2016.

[23] A. Beimel, Y. Ishai, R. Kumaresan, and E. Kushilevitz. On the cryptographic complexity of the worst functions. In Y. Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 317–342. Springer, Heidelberg, Feb. 2014.

[24] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway. Efficient garbling from a fixed-key blockcipher. In *2013 IEEE Symposium on Security and Privacy*, pages 478–492. IEEE Computer Society Press, May 2013.

[25] M. Bellare, V. T. Hoang, and P. Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In X. Wang and K. Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 134–153. Springer, Heidelberg, Dec. 2012.

[26] M. Bellare, V. T. Hoang, and P. Rogaway. Foundations of garbled circuits. In T. Yu, G. Danezis, and V. D. Gligor, editors, *ACM CCS 12*, pages 784–796. ACM Press, Oct. 2012.

[27] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.

[28] N. Bitansky, S. Garg, H. Lin, R. Pass, and S. Telang. Succinct randomized encodings and their applications. In R. A. Servedio and R. Rubinfeld, editors, *47th ACM STOC*, pages 439–448. ACM Press, June 2015.

[29] J. Black, P. Rogaway, and T. Shrimpton. Encryption-scheme security in the presence of key-dependent messages. In K. Nyberg and H. M. Heys, editors, *SAC 2002*, volume 2595 of *LNCS*, pages 62–75. Springer, Heidelberg, Aug. 2003.

[30] D. Boneh, C. Gentry, S. Gorbunov, S. Halevi, V. Nikolaenko, G. Segev, V. Vaikuntanathan, and D. Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In P. Q. Nguyen and E. Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 533–556. Springer, Heidelberg, May 2014.

[31] D. Boneh, S. Halevi, M. Hamburg, and R. Ostrovsky. Circular-secure encryption from decision Diffie-Hellman. In D. Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 108–125. Springer, Heidelberg, Aug. 2008.

[32] Z. Brakerski and S. Goldwasser. Circular and leakage resilient public-key encryption under subgroup indistinguishability - (or: Quadratic residuosity strikes back). In T. Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 1–20. Springer, Heidelberg, Aug. 2010.

[33] Z. Brakerski, S. Goldwasser, and Y. T. Kalai. Black-box circular-secure encryption beyond affine functions. In Y. Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 201–218. Springer, Heidelberg, Mar. 2011.

[34] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In R. Ostrovsky, editor, *52nd FOCS*, pages 97–106. IEEE Computer Society Press, Oct. 2011.

[35] C. Cachin, J. Camenisch, J. Kilian, and J. Muller. One-round secure computation and secure autonomous mobile agents. In U. Montanari, J. D. P. Rolim, and E. Welzl, editors, *ICALP 2000*, volume 1853 of *LNCS*, pages 512–523. Springer, Heidelberg, July 2000.

[36] J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In B. Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118. Springer, Heidelberg, May 2001.

[37] R. Canetti, J. Holmgren, A. Jain, and V. Vaikuntanathan. Succinct garbling and indistinguishability obfuscation for RAM programs. In R. A. Servedio and R. Rubinfeld, editors, *47th ACM STOC*, pages 429–437. ACM Press, June 2015.

[38] R. Canetti, H. Lin, S. Tessaro, and V. Vaikuntanathan. Obfuscation of probabilistic circuits and applications. In Y. Dodis and J. B. Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 468–497. Springer, Heidelberg, Mar. 2015.

[39] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th ACM STOC*, pages 11–19. ACM Press, May 1988.

[40] R. Cramer, S. Fehr, Y. Ishai, and E. Kushilevitz. Efficient multi-party computation over rings. In E. Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 596–613. Springer, Heidelberg, May 2003.

[41] I. Damgård and Y. Ishai. Scalable secure multiparty computation. In C. Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 501–520. Springer, Heidelberg, Aug. 2006.

[42] Z. Dvir, D. Gutfreund, G. N. Rothblum, and S. P. Vadhan. On approximating the entropy of polynomial mappings. In B. Chazelle, editor, *ICS 2011*, pages 460–475. Tsinghua University Press, Jan. 2011.

[43] U. Feige, J. Kilian, and M. Naor. A minimal model for secure computation (extended abstract). In *26th ACM STOC*, pages 554–563. ACM Press, May 1994.

[44] S. Garg, S. Lu, R. Ostrovsky, and A. Scafuro. Garbled RAM from one-way functions. In R. A. Servedio and R. Rubinfeld, editors, *47th ACM STOC*, pages 449–458. ACM Press, June 2015.

[45] R. Gay, I. Kerenidis, and H. Wee. Communication complexity of conditional disclosure of secrets and attribute-based encryption. In R. Gennaro and M. J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 485–502. Springer, Heidelberg, Aug. 2015.

[46] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In T. Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 465–482. Springer, Heidelberg, Aug. 2010.

[47] C. Gentry. Fully homomorphic encryption using ideal lattices. In M. Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

[48] C. Gentry, S. Halevi, S. Lu, R. Ostrovsky, M. Raykova, and D. Wichs. Garbled RAM revisited. In P. Q. Nguyen and E. Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 405–422. Springer, Heidelberg, May 2014.

[49] C. Gentry, S. Halevi, and V. Vaikuntanathan. i-Hop homomorphic encryption and rerandomizable Yao circuits. In T. Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 155–172. Springer, Heidelberg, Aug. 2010.

[50] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting data privacy in private information retrieval schemes. *JCSS: Journal of Computer and System Sciences*, 60, 2000.

[51] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In A. Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[52] S. Goldwasser, D. Gutfreund, A. Healy, T. Kaufman, and G. N. Rothblum. A (de)constructive approach to program checking. In R. E. Ladner and C. Dwork, editors, *40th ACM STOC*, pages 143–152. ACM Press, May 2008.

[53] S. Goldwasser, Y. T. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich. How to run turing machines on encrypted data. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 536–553. Springer, Heidelberg, Aug. 2013.

[54] S. Goldwasser, Y. T. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich. Reusable garbled circuits and succinct functional encryption. In D. Boneh, T. Roughgarden, and J. Feigenbaum, editors, *45th ACM STOC*, pages 555–564. ACM Press, June 2013.

[55] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. One-time programs. In D. Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 39–56. Springer, Heidelberg, Aug. 2008.

[56] S. Gorbunov, V. Vaikuntanathan, and H. Wee. Functional encryption with bounded collusions via multi-party computation. In R. Safavi-Naini and R. Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 162–179. Springer, Heidelberg, Aug. 2012.

[57] S. Gorbunov, V. Vaikuntanathan, and H. Wee. Attribute-based encryption for circuits. In D. Boneh, T. Roughgarden, and J. Feigenbaum, editors, *45th ACM STOC*, pages 545–554. ACM Press, June 2013.

[58] V. Goyal, Y. Ishai, A. Sahai, R. Venkatesan, and A. Wadia. Founding cryptography on tamper-proof hardware tokens. In D. Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 308–326. Springer, Heidelberg, Feb. 2010.

[59] I. Haitner, O. Reingold, and S. P. Vadhan. Efficiency improvements in constructing pseudorandom generators from one-way functions. In L. J. Schulman, editor, *42nd ACM STOC*, pages 437–446. ACM Press, June 2010.

[60] D. Harnik and M. Naor. On the compressibility of NP instances and cryptographic applications. In *47th FOCS*, pages 719–728. IEEE Computer Society Press, Oct. 2006.

[61] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[62] B. Hemenway, Z. Jafargholi, R. Ostrovsky, A. Scafuro, and D. Wichs. Adaptively secure garbled circuits from one-way functions. Cryptology ePrint Archive, Report 2015/1250, 2015. `http://eprint.iacr.org/2015/1250`.

[63] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *20th USENIX Security Symposium*, 2011.

[64] Y. Ishai. Randomization techniques for secure computation. In M. Prabhakaran and A. Sahai, editors, *Secure Multi-Party Computation*, volume 10 of *Cryptology and Information Security Series*, pages 222–248. IOS Press, 2013.

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

[66] Y. Ishai and E. Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *41st FOCS*, pages 294–304. IEEE Computer Society Press, Nov. 2000.

[67] Y. Ishai and E. Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In P. Widmayer, F. T. Ruiz, R. M. Bueno, M. Hennessy, S. Eidenbenz, and R. Conejo, editors, *ICALP 2002*, volume 2380 of *LNCS*, pages 244–256. Springer, Heidelberg, July 2002.

[68] Y. Ishai and H. Wee. Partial garbling schemes and their applications. In J. Esparza, P. Fraigniaud, T. Husfeldt, and E. Koutsoupias, editors, *ICALP 2014, Part I*, volume 8572 of *LNCS*, pages 650–662. Springer, Heidelberg, July 2014.

[69] J. Kilian. Founding cryptography on oblivious transfer. In *20th ACM STOC*, pages 20–31. ACM Press, May 1988.

[70] V. Kolesnikov, P. Mohassel, and M. Rosulek. FleXOR: Flexible garbling for XOR gates that beats free-XOR. In J. A. Garay and R. Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 440–457. Springer, Heidelberg, Aug. 2014.

[71] V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, Heidelberg, July 2008.

[72] V. Koppula, A. B. Lewko, and B. Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In R. A. Servedio and R. Rubinfeld, editors, *47th ACM STOC*, pages 419–428. ACM Press, June 2015.

[73] B. Kreuter, a. shelat, and C. hao Shen. Billion-gate secure computation with malicious adversaries. Cryptology ePrint Archive, Report 2012/179, 2012. `http://eprint.iacr.org/2012/179`.

[74] H. Lin, R. Pass, K. Seth, and S. Telang. Output-compressing randomized encodings and applications. LNCS, pages 96–124. Springer, Heidelberg, 2016.

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

[76] Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. *Journal of Cryptology*, 28(2):312–350, Apr. 2015.

[77] Y. Lindell, B. Pinkas, and N. P. Smart. Implementing two-party computation efficiently with security against malicious adversaries. In R. Ostrovsky, R. D. Prisco, and I. Visconti, editors, *SCN 08*, volume 5229 of *LNCS*, pages 2–20. Springer, Heidelberg, Sept. 2008.

[78] S. Lu and R. Ostrovsky. How to garble RAM programs. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 719–734. Springer, Heidelberg, May 2013.

[79] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay - secure two-party computation system. In *13th USENIX Security Symposium*, pages 287–302, 2004.

[80] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *1st EC*, pages 129–139. ACM Press, Nov. 1999.

[81] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams. Secure two-party computation is practical. In M. Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 250–267. Springer, Heidelberg, Dec. 2009.

[82] M. Rosulek. A brief history of practical garbled circuit optimizations. Securing Computation workshop, Simons Institute, June 2015. Available at `http://web.engr.oregonstate.edu/~rosulekm/`.

[83] A. Sahai and H. Seyalioglu. Worry-free encryption: functional encryption with public keys. In E. Al-Shaer, A. D. Keromytis, and V. Shmatikov, editors, *ACM CCS 10*, pages 463–472. ACM Press, Oct. 2010.

[84] T. Sander, A. Young, and M. Yung. Non-interactive cryptocomputing for NC1. In *40th FOCS*, pages 554–567. IEEE Computer Society Press, Oct. 1999.

[85] A. C.-C. Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, Nov. 1982.

[86] A. C.-C. Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, Oct. 1986.

[87] S. Zahur, M. Rosulek, and D. Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, Heidelberg, Apr. 2015.

A Randomized Encodings vs. Garbling Schemes [26]

In this section we briefly compare the RE framework to the notion of *garbling schemes* introduced by Bellare, Hoang and Rogaway (BHR) in [26].

Roughly speaking, garbling schemes can be viewed as a concrete variant of RE which is required to satisfy some additional syntactic properties. In particular, the encoding $\hat{f}(x; r)$ is partitioned into three parts (F, d, X), where X is the online part, and (F, d) essentially corresponds to the offline part of the encoding. (The d part is viewed as a "decoding key" and it allows to consider a designated decoding variant as opposed to public decoding). The BHR framework decomposes the encoder into two algorithms $(\mathsf{Gb}, \mathsf{En})$, and decomposes the decoder into two algorithms $(\mathsf{De}, \mathsf{Ev})$ as follows.

Syntax. The probabilistic garbling algorithm Gb is given a security parameter 1^k and a description of a function f (under some fixed representation scheme) and returns a triple of strings $(F, e, d) \overset{R}{\leftarrow} \mathsf{Gb}(1^k, f)$. Think of F as an offline encoding, and on e and d as an encoding and decoding keys, respectively. The strings e, d and the length of F are allowed to depend on the syntactic properties of f (i.e., its input length, output length and description length), but, otherwise, should be independent of f. The (deterministic) *input encoding* algorithm, $\mathsf{En}(\cdot, \cdot)$, takes an encoding key e and an input $x \in \{0, 1\}^n$ and outputs a garbled input $X = \mathsf{En}(e, x)$. The decoding proceeds in two steps: first the algorithm $\mathsf{Ev}(F, X)$ maps F and a garbled input X to a "garbled output" $Y = \mathsf{Ev}(F, X)$, and second the decoding algorithm $\mathsf{De}(d, Y)$ maps a decoding key d and a garbled output Y to a final output $y = \mathsf{De}(d, Y)$. Correctness asserts that $\mathsf{De}(d, \mathsf{Ev}(F, \mathsf{En}(e, x))) = \mathsf{ev}(f, x)$, where ev is a universal evaluation algorithm that maps a description f of a function, and an input x to the value of the corresponding function on the input x. Overall, BHR define a *garbling scheme* as the five-tuple $(\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$.

Security. Garbling schemes can satisfy different variants of security. The most central one is a parameterized version of privacy which, in addition to standard privacy (as defined for REs), may support some notion of "function hiding" for the encoded function f (the exact level of hiding depends on a leakage parameter). Other notions of security include *obliviousness* and *authenticity*. Obliviousness means that decoding can be performed only given a secret key d (i.e., (F, X) do not reveal $y = f(x)$). Authenticity essentially means that, given (F, X), it is hard to generate Y' which leads to an undetected decoding error (i.e., $\mathsf{De}(d, Y')) \notin \{f(x), \perp\}$).

Comparison. Syntactically, garbling schemes and REs are quite different. These differences reflect two conflicting goals. The garbling scheme framework offers a concrete and highly detailed treatment of garbled circuits which can be almost given to a programmer as an API. As a result, efficiency requirements (e.g., online efficiency) are hard-wired into the syntax itself. Moreover, in order to keep the framework wide enough (while keeping it concrete), the syntax has to take into account several different usage scenarios (e.g., the possibility of designated decoding) which, inevitably, make the definition more complicated (e.g., the decomposition of the decoder into an evaluator and decoder). In contrast, the RE framework strives for minimalism. It deals only with basic correctness and privacy, and deliberately leave efficiency issues to be determined in an application-dependent context.[18] As a result, the RE framework offers a high-level view which misses some practical aspects but highlights the general properties of REs which are independent of efficiency concerns (cf. Section 2.3).

[18] This is also the case with more refined security variants. Indeed, function-hiding, obliviousness, and authenticity can all be reduced to basic privacy via simple transformations with a polynomial overhead (e.g., by encoding a universal circuit $g(\langle f\rangle, x) = f(x)$, encoding an encrypted circuit $g(s, x) = f(x) \oplus s$ or encoding a MAC-ed circuit $g(s, x) = \mathtt{MAC}_s(f(x))$; see Section 5.1). Of course, a direct study of these notions may lead to solutions with better efficiency.