Obfuscating Compute-and-Compare Programs under LWE

Daniel Wichs[*] Giorgos Zirdelis[†]

August 15, 2017

Abstract

We show how to obfuscate a large and expressive class of programs, which we call *compute-and-compare programs*, under the learning-with-errors (LWE) assumption. Each such program $\mathbf{CC}[f, y]$ is parametrized by an arbitrary polynomial-time computable function f along with a target value y and we define $\mathbf{CC}[f, y](x)$ to output 1 if $f(x) = y$ and 0 otherwise. In other words, the program performs an arbitrary computation f and then compares its output against a target y. Our obfuscator satisfies distributional virtual-black-box security, which guarantees that the obfuscated program does not reveal any partial information about the function f or the target value y, as long as they are chosen from some distribution where y has sufficient pseudo-entropy given f. We also extend our result to multi-bit compute-and-compare programs $\mathbf{MBCC}[f, y, z](x)$ which output a message z if $f(x) = y$.

Compute-and-compare programs are powerful enough to capture many interesting obfuscation tasks as special cases. This includes obfuscating conjunctions, and therefore we improve on the prior work of Brakerski et al. (ITCS '16) which constructed a conjunction obfuscator under a non-standard "entropic" ring-LWE assumption, while here we obfuscate a significantly broader class of programs under standard LWE. We show that our obfuscator has several interesting applications. For example, we can take any encryption scheme and publish an obfuscated *plaintext equality tester* that allows users to check whether a ciphertext decrypts to some target value y; as long as y has sufficient pseudo-entropy this will not harm semantic security. We can also use our obfuscator to generically upgrade attribute-based encryption to predicate encryption with one-sided attribute-hiding security, and to upgrade witness encryption to indistinguishability obfuscation which is secure for all "null circuits". Furthermore, we show that our obfuscator gives new circular-security counter-examples for public-key bit encryption and for unbounded length key cycles.

Our result uses the *graph-induced multi-linear maps* of Gentry, Gorbunov and Halevi (TCC '15), but only in a carefully restricted manner which is provably secure under LWE. Our technique is inspired by ideas introduced in a recent work of Goyal, Koppula and Waters (EUROCRYPT '17) in a seemingly unrelated context.

1 Introduction

The goal of program obfuscation [Had00, BGI$^+$01, GGH$^+$13b] is to encode a program in a way that preserves its functionality while hiding everything else about its code and its internal operation. Barak et al. [BGI$^+$01] proposed a strong security definition for obfuscation, called *virtual black-box (VBB)* security, which (roughly) guarantees that the obfuscated program can be simulated given black-box access to the program's functionality. Unfortunately, they showed that *general purpose* VBB obfuscation is unachievable. This leaves open two possibilities: (1) achieving weaker security notions of obfuscation for general programs, and (2) achieving virtual black box obfuscation for restricted classes of programs.

Along the first direction, Barak et al. proposed a weaker security notion called *indistinguishability obfuscation (iO)* which guarantees that the obfuscations of two functionally equivalent programs are indistinguishable. In a breakthrough result, Garg, Gentry, Halevi, Raykova, Sahai and Waters [GGH$^+$13b]

[*]Northeastern University. E-mail: wichs@ccs.neu.edu.

[†]Northeastern University. E-mail: zirdelis.g@husky.neu.edu.

showed how to iO-obfuscate all polynomial-size circuits using *multi-linear maps* [GGH13a]. Since then, there has been much follow-up work on various constructions and cryptanalysis of multi-linear maps, constructions and cryptanalysis of iO using multi-linear maps, and various applications of iO. At this point, we have heuristic candidate constructions of iO which we do not know how to attack, but we lack a high degree of confidence in their security and don't have a good understanding of the underlying computational problems on which such schemes are based. It remains a major open problem to construct iO under standard well-studied assumptions.

Along the second direction, several interesting but highly restricted classes of programs have been shown to be virtual black-box obfuscatable. This includes constructions of (multi-bit) point function obfuscators [Can97, Wee05, CD08, Zha16] in the random oracle model or under various (semi-)standard assumptions, hyperplane obfuscators assuming strong DDH [CRV10], and very recently conjunction obfuscators, first using multi-linear maps [BR13] and later a variant of Ring LWE called "entropic Ring LWE" [BVWW16]. It remains an open problem to understand which classes of programs can we even hope to VBB obfuscate to avoid the impossibility results of Barak et al.

In summary, prior to this work, we did not know how to achieve any meaningful definition of obfuscation for any expressive class of programs under any standard assumption.

1.1 Our Results

In this work, we show how to obfuscate a large and expressive class of programs which we call *compute-and-compare programs*, achieving a strong notion of security called *distributional virtual black box (D-VBB)*, under the *learning with errors (LWE)* assumption. This is the first such result that allows us obfuscate complex programs under a standard assumption.

A compute-and-compare program $\mathbf{CC}[f, y]$ is parameterized by a function $f : \{0,1\}^{\ell_{in}} \to \{0,1\}^{\ell_{out}}$, represented as a circuit or a Turing Machine, along with a target value $y \in \{0,1\}^{\ell_{out}}$ and we define $\mathbf{CC}[f, y](x) = 1$ if $f(x) = y$ and $\mathbf{CC}[f, y](x) = 0$ otherwise. In other words, the program performs an arbitrary computation f and then compares the output against a target y. The D-VBB definition of security says that an obfuscation of $\mathbf{CC}[f, y]$ hides all partial information about the function f and the target value y as long as they are chosen from some distribution where y has sufficient min-entropy or (HILL) pseudo-entropy given f.[1] We can relax this to only requiring that y is computationally unpredictable given f, but in that case we also need an additional mild assumption that there exist pseudo-random generators for unpredictable sources which holds e.g, in the random oracle model or assuming the existence of extremely lossy functions (ELFs) [Zha16]. All our results hold in the presence of auxiliary input, as long as y remains sufficiently unpredictable even given f and the auxiliary input.

We also extend our result to *multi-bit* compute-and-compare programs $\mathbf{MBCC}[f, y, z](x)$ which output a message z if $f(x) = y$ and otherwise output $\perp$. In this case we ensure that the obfuscated program does not reveal anything about f, y, z as long as they are chosen from some distribution where y has sufficient pseudo-entropy (or is computationally unpredictable) even given f, z.

When the function f is represented as a Turing Machine with some fixed run-time t, our obfuscator is *succinct* meaning that the run-time of our obfuscator and the size of the obfuscated program only have a poly-logarithmic dependence on t. To get this we need to further rely on true (non-leveled) fully homomorphic encryption (FHE) which requires a circular security assumption. Assuming only leveled FHE, which we have under standard LWE, we get a *weakly succinct* scheme where the run-time of the obfuscator depends polynomially on $\log t, d$, where d is the depth of the circuit computing f.

Obfuscating Evasive Programs. We note that compute-and-compare programs $\mathbf{CC}[f, y]$ where y has pseudo-entropy given f are an example of *evasive programs*, meaning that for any input x chosen a-priori, with overwhelming probability the program outputs 0. When obfuscating evasive programs,

[1]The HILL pseudo-entropy must be at least λ^{ε}, where λ is the security parameter and $\varepsilon > 0$ is an arbitrary constant.

D-VBB security ensures that one cannot find an input on which it evaluates to anything other than 0. This may seem strange at first; what is the point of creating the obfuscated program and ensuring that it functions correctly on all inputs if users cannot even find any input on which it does not output 0? However, the point is that there may be some users with additional information about y (for whom it does not have much pseudo-entropy) and who may therefore be able to find inputs on which the program outputs 1. In other words, the correctness of obfuscation is meaningful for users for whom y does not have much pseudo-entropy (but for such users we do not get any meaningful security), while security is meaningful for users for whom y has sufficient pseudo-entropy (but for such users correctness is not very meaningful since they will always get a 0 output). The work of [BBC+14] shows that one cannot have (D-)VBB obfuscation for all evasive functions (with auxiliary input) and our work is the first to identify a large sub-class of evasive functions for which it is possible. We show that this type of obfuscation is already powerful and has several interesting applications.

1.2 Applications

Obfuscation for compute-and-compare programs is already sufficiently powerful and expressive to capture many interesting obfuscation tasks and gives rise to new applications as well as a simple and modular way to recover several prior results.

Conjunctions and Affine Testers. We can think of *conjunctions* as a restricted special case of compute-and-compare programs $\mathbf{CC}[f, y]$ where the function $f(x)$ simply outputs some subset of the bits of x. Therefore our result improves on the work of [BVWW16] which constructed an obfuscator for conjunctions under a non-standard entropic Ring-LWE assumption, whereas here we get a conjunction obfuscator under standard LWE as a special case of our result. Moreover, our obfuscator also achieves a stronger notion of security for a broader class of distributions than the previous constructions.

As another special case which generalizes conjunctions, we can obfuscate arbitrary *affine testers* which are parameterized by a matrix $\mathbf{A}$ and a vector $\mathbf{y}$ and test whether an input $\mathbf{x}$ satisfies $\mathbf{Ax} \overset{?}{=} \mathbf{y}$, where security is guaranteed as long as $\mathbf{y}$ has sufficient pseudo-entropy given $\mathbf{A}$.

Secure Sketches. We also show that our obfuscator allows us to convert any *secure sketch* [DORS08] into a (computational) *private secure sketch* [DS05]. A secure sketch $\mathsf{SS}(y)$ of a string y allows us to recover y given any string x which is close to y (e.g., in hamming distance) without revealing all the entropy in y. However, the sketch may reveal various sensitive partial information about y. We show how to convert any secure sketch into a private one, which does not reveal any partial information, by obfuscating a program that has $\mathsf{SS}(y)$ inside it.

Plaintext Equality Tester. Using our obfuscator, we can take an arbitrary encryption scheme and obfuscate a *plaintext equality tester* $\mathbf{CC}[\mathsf{Dec}_{\mathsf{sk}}, y]$ which has a hard-coded secret key sk and a target plaintext value y and tests whether a given ciphertext ct decrypts to $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}) = y$. Or, more generally, we can evaluate an arbitrary polynomial-time function g on the plaintext and test if $g(\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct})) = y$ by obfuscating $\mathbf{CC}[g \circ \mathsf{Dec}_{\mathsf{sk}}, y]$. As long as the target y has sufficient pseudo-entropy, our obfuscated plaintext equality tester can be simulated without knowing sk and therefore will not harm the semantic security of the encryption scheme. The idea of obfuscating a plaintext-equality tester is implicitly behind several of our other applications, and we envision that more applications should follow.

Attribute Based Encryption to One-Sided Predicate Encryption. We show that our obfuscator allows us to generically upgrade *attribute-based encryption* (ABE) into *predicate encryption* (PE) with one-sided attribute-hiding security, meaning that the attribute is hidden from any user who is not qualified to decrypt. Although the recent work of Gorbunov, Vaikuntanathan and Wee [GVW15] constructed such predicate encryption for all circuits under LWE by cleverly leveraging a prior construction of attribute-based encryption [BGG$^+$14] under LWE, it was a fairly intricate non-generic construction with a subtle analysis, while our transformation is simple and generic. For example, it shows that any future advances in attribute-based encryption (e.g., getting rid of the dependence on circuit depth in encryption efficiency and ciphertext size) will directly translate to predicate encryption as well.

Witness Encryption to Null iO. A witness encryption scheme [GGSW13] allows us to use any NP statement x as a public-key to encrypt a message m. Any user who knows the corresponding witness w for x will be able to decrypt m, but if x is a false statement then m is computationally hidden. We show that our obfuscator for compute-and-compare programs allows us to convert any witness encryption (WE) into an obfuscation scheme that has correctness for all circuits and guarantees that we cannot distinguish the obfuscations of any two *null* circuits C, C' such that $C(x) = C'(x) = 0$ for all inputs x. We call this notion *null iO* or *niO*. We previously knew that iO implies niO which in turn implies WE, but we did not know anything about the reverse directions. Our result shows that under the LWE assumptions, WE implies niO. It remains as a fascinating open problem whether niO implies full iO.

Circular Security Counter-Examples. Finally, we show that our obfuscator gives us new counter-examples to various circular security problems.

Firstly, it gives us a simple construction of a public-key bit-encryption scheme which is semantically secure but is not circular secure: given ciphertexts encrypting the secret key one bit at a time, we can completely recover the secret key. This means that, under the LWE assumption, semantic security does not generically imply circular security for all public-key bit-encryption schemes. Previously, we only had such counter-examples under non-standard assumptions (multi-linear maps or obfuscation) [Rot13, KRW15]. The very recent work of Goyal, Koppula and Waters [GKW17b] provided such a counter-example for *symmetric-key* bit-encryption under LWE. Using our obfuscator, we get a simple and modular counter-example for *public-key* bit-encryption under LWE.

Secondly, it gives us a simple construction of a public-key bit-encryption scheme which is semantically secure but not circular secure for key cycles of any unbounded polynomial length ℓ. That is, we construct a single scheme such that, given a cycle $\mathsf{Enc}_{\mathsf{pk}_1}(\mathsf{sk}_2), \mathsf{Enc}_{\mathsf{pk}_2}(\mathsf{sk}_3), \ldots, \mathsf{Enc}_{\mathsf{pk}_{\ell-1}}(\mathsf{sk}_\ell), \mathsf{Enc}_{\mathsf{pk}_\ell}(\mathsf{sk}_1)$ of any arbitrary polynomial length ℓ, we can completely recover all of the secret keys. Previously, we had such results for bounded-length cycles under LWE [AP16, KW16] or unbounded-length cycles under iO [GKW17a]. Using our obfuscator, we get a result for unbounded-length cycles under LWE. Furthermore, our scheme does not require any common public parameters.

Thirdly, we consider a compiler proposed by Black, Rogaway, and Shrimpton [BRS03] which transforms any semantically secure scheme into a circular secure (and even Key-Dependent Message secure) one in the random-oracle model. We show that this compiler fails in the standard model: under the LWE assumption, there exists a semantically secure scheme such that, when we apply the transformation of [BRS03] and replace the random oracle with *any* hash function, the resulting scheme fails to be circular secure.

1.3 Concurrent and Independent Work of [GKW17c]

The concurrent and independent work of Goyal, Koppula and Waters [GKW17c] achieves essentially the same results as this work modulo small differences in presentation and focus. They define a notion of "lockable obfuscation" which (in our language) is an obfuscation scheme for multi-bit compute and compare programs $\mathbf{MBCC}[f, y, z]$ where y is uniformly random and independent of f, z. While we allow for more general distributions, where y only has pseudo-entropy/unpredictability given f, z, this can be achieved generically from lockable obfuscation using a pseudorandom generator that works with any high pseudo-entropy seed. Indeed, the main constructions in both works are essentially identical. Both works also present applications of this type of obfuscation to one-sided predicate encryption, null iO, and circular security counter-examples. Our work also shows applications to private secure sketches and to other obfuscation tasks such as obfuscating conjunctions and affine spaces while the work of [GKW17c] gives new results showing the uninstatiability of random oracle schemes.

1.4 Our Techniques

Our result relies on the *graph induced multilinear maps* of Gentry, Gorbunov and Halevi [GGH15]. In the original work [GGH15], such maps were used in a heuristic manner to construct iO and various other applications, but there was no attempt to define or prove any stand-alone security properties of such maps. The subsequent work of [CLLT16] came up with attacks on various uses of such multilinear maps, showing that some of the applications in [GGH15] are insecure. However, a series of works also showed that certain highly restricted uses of these multilinear maps are actually provably secure under the LWE assumption. In particular, the works of [BVWW16, KW16, AP16, GKW17b, CC17] all either implicitly or explicitly rely on various provably secure properties of the [GGH15] multilinear map. Following [BVWW16] we refer to a restricted version of the [GGH15] scheme as a *directed encoding.*

Our particular use of directed encodings is inspired by the recent work of Goyal, Koppula and Waters [GKW17b] which studied the seemingly unrelated problem of circular security counterexamples for symmetric-key bit-encryption. As one of the components of their solution, they described a clever way of encoding branching programs. We essentially use this encoding as the core component of our obfuscation construction. The work of [GKW17b] did not explicitly define or analyze any security properties of their encoding and did not draw a connection to obfuscation. Indeed, as we will elaborate later, there are major differences between the security properties of the encoding that they implicitly used in the context of their construction and the ones that we rely on in our work. We show how to use this encoding to get a "basic obfuscation scheme" for compute-and-compare program $\mathbf{CC}[f, y]$ where f is a branching program and y has very high pseudo-entropy. We then come up with generic transformations to go from branching programs to circuits or Turing Machines and to reduce the requirements on the pseudo-entropy of y to get our final result.

Directed Encodings. A directed encoding scheme contains public keys $\mathbf{A}_i \in \mathbb{Z}_q^{n\times m}$. We define an encoding of a "small" secret $\mathbf{S} \in \mathbb{Z}_q^{n\times n}$ along the edge $\mathbf{A}_i \to \mathbf{A}_j$ as a "small" matrix $\mathbf{C} \in \mathbb{Z}_q^{m\times m}$ such that $\mathbf{A}_i\mathbf{C} = \mathbf{S}\mathbf{A}_j + \mathbf{E}$ where $\mathbf{E} \in \mathbb{Z}_q^{n\times m}$ is some "small" noise. For simplicity, we will just write $\mathbf{A}_i\mathbf{C} \approx \mathbf{S}\mathbf{A}_j$ where the $\approx$ hides "small" noise. Creating such an encoding requires knowing a trapdoor for the public key $\mathbf{A}_i$.

Given an encoding $\mathbf{C}_1$ of a secret $\mathbf{S}_1$ along an edge $\mathbf{A}_1 \to \mathbf{A}_2$ and an encoding $\mathbf{C}_2$ of a secret $\mathbf{S}_2$ along an edge $\mathbf{A}_2 \to \mathbf{A}_3$, the value $\mathbf{C}_1 \cdot \mathbf{C}_2$ is an encoding of $\mathbf{S}_1 \cdot \mathbf{S}_2$ along the edge $\mathbf{A}_1 \to \mathbf{A}_3$ with slightly larger noise. More generally, given encodings $\mathbf{C}_i$ of secrets $\mathbf{S}_i$ along edges $\mathbf{A}_i \to \mathbf{A}_{i+1}$, the value $\mathbf{C}^* = \prod_{i=1}^{L} \mathbf{C}_i$ is an encoding of $\mathbf{S}^* = \prod_{i=1}^{L} \mathbf{S}_i$ along the edge $\mathbf{A}_1 \to \mathbf{A}_{L+1}$ meaning that $\mathbf{A}_1\mathbf{C}^* \approx \mathbf{S}^*\mathbf{A}_{L+1}$.

We can also encode a secret $\mathbf{S}$ along multiple edges $\{\mathbf{A}_1 \to \mathbf{A}'_1\ ,\ \dots\ ,\ \mathbf{A}_w \to \mathbf{A}'_w\}$ simultaneously by sampling a matrix $\mathbf{C} \in \mathbb{Z}_q^{m\times m}$ such that

$$\begin{bmatrix} \mathbf{A}_1 \\ \dots \\ \mathbf{A}_w \end{bmatrix} \mathbf{C} = \begin{bmatrix} \mathbf{S}\cdot\mathbf{A}'_1 + \mathbf{E}_1 \\ \dots \\ \mathbf{S}\cdot\mathbf{A}'_w + \mathbf{E}_w \end{bmatrix}$$

This can be done the same way as in the single-edge case given the trapdoor for the matrix $\mathbf{B} = \begin{bmatrix} \mathbf{A}_1 \\ \dots \\ \mathbf{A}_w \end{bmatrix}$ with dimensions $(n\cdot w)\times m$. The resulting encoding $\mathbf{C}$ satisfies $\mathbf{A}_j\mathbf{C} \approx \mathbf{S}\mathbf{A}'_j$ for all $j \in [w]$ and therefore is an encoding of $\mathbf{S}$ along each one of the edges $\mathbf{A}_j \to \mathbf{A}'_j$ separately.

Encoding Branching Programs. As a building block, we define the notion of "encoding" a permutation branching program P. This encoding is not an obfuscation scheme yet, since it does not allow us to evaluate the encoded program and learn the output. However, it's a useful first step toward obfuscation.

We think of a boolean permutation branching program P of input size ℓ_{in}, length L and width w, as a graph containing $(L+1)\cdot w$ vertices that are grouped into $(L+1)$ levels of w vertices each; we denote these vertices by (i,j) for $i \in \{1,\ldots,L+1\}, j \in \{0,\ldots,w-1\}$. Each level $i \le L$ is associated with two permutations $\pi_{i,0}, \pi_{i,1}$ over $\{0,\ldots,w-1\}$. For each vertex (i,j) at level $i \le L$ we use the permutations to define two outgoing edges labeled with 0 and 1 that respectively go to vertices $(i+1, \pi_{i,0}(j))$ and $(i+1, \pi_{i,1}(j))$ at level $i+1$. To evaluate the branching program P on an input $x = (x_1, \ldots, x_{\ell_{in}}) \in \{0,1\}^{\ell_{in}}$ we start at the vertex $(1,0)$ and at each level $i \in [L]$ we follow the edge labeled with the bit $x_{(i \bmod \ell_{in})}$. At the final level $L+1$, we end up at a vertex $(L+1, b)$ where $b \in \{0,1\}$ is the output of the program $P(x)$. See Figure 1.4 for an example. By Barrington's theorem [Bar89], any $\mathbf{NC}^1$ circuit can be converted into a branching program with constant-width $w = 5$ and polynomial-length.[2]

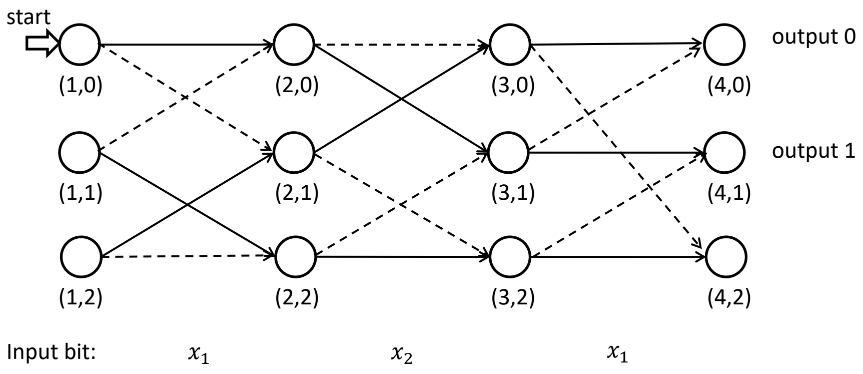

Figure 1: *Example of a branching program of length $L = 3$, width $w = 3$, and input size $\ell_{in} = 2$. Solid edges are labeled with 1 and the dashed edges with 0. For example, on input $x = 10$ (i.e., $x_1 = 1, x_2 = 0$) the program evaluates to 0. (Technically, the above simple example is not a legal branching program since on input $x = 01$ it does not evaluate to 0 or 1, but it is useful to illustrate the concept.)*

To encode a branching program, we associate a public key $\mathbf{A}_{i,j}$ with each vertex (i,j) of the branching program. For each level $i \in [L]$ we pick two random secrets $\mathbf{S}_{i,0}, \mathbf{S}_{i,1}$ and create two encodings $\mathbf{C}_{i,0}, \mathbf{C}_{i,1}$ where $\mathbf{C}_{i,b}$ encodes $\mathbf{S}_{i,b}$ simultaneously along the w edges $\{\mathbf{A}_{i,0} \to \mathbf{A}_{i+1,\pi_{i,b}(0)}\ , \ldots,\ \mathbf{A}_{i,w-1} \to \mathbf{A}_{i+1,\pi_{i,b}(w-1)}\}$ that are labeled with the bit b. For any input $x \in \{0,1\}^{\ell_{in}}$ we can then "evaluate" the encoded branching program on x to get:

$$\mathbf{D} := \mathbf{A}_{1,0} \cdot \left(\prod_{i=1}^{L} \mathbf{C}_{i, x_{(i \bmod \ell_{in})}} \right) \quad \text{satisfying} \quad \mathbf{D} \approx \left(\prod_{i=1}^{L} \mathbf{S}_{i, x_{(i \bmod \ell_{in})}} \right) \cdot \mathbf{A}_{L+1, P(x)}.$$

Note that this "evaluation" does not allow us to recover the output $P(x)$, but only gives us an LWE sample with respect to the matrix $\mathbf{A}_{L+1,P(x)}$ which depends on the output.

We can also encode a branching program P with ℓ_{out}-bit output, by thinking of it as a sequence of boolean branching programs $P = (P^{(1)}, \ldots, P^{(\ell_{out})})$ for each output bit, where all the programs have a common length L, width w, and access pattern in which the i'th level reads the input bit $(i \bmod \ell_{in})$. We essentially encode each boolean program $P^{(k)}$ separately as described above with fresh and independent public keys $\mathbf{A}^{(k)}_{i,j}$ but we use the same secrets $\mathbf{S}_{i,0}, \mathbf{S}_{i,1}$ across all programs. This allows us to evaluate

[2]We depart from the usual definition of branching programs by insisting that the input-bits are accessed in a fixed order where step i reads bit $i \bmod \ell_{in}$. However, this is without loss of generality since any branching program that reads the input in an arbitrary order can be converted into one of this form at the expense of increasing the length by a factor of ℓ_{in}.

the entire sequence of encoded programs on some input x and derive a sequence of LWE samples $\mathbf{D}^{(k)} \approx \mathbf{S}^* \cdot \mathbf{A}^{(k)}_{L+1,P^{(k)}(x)}$ with a common secret $\mathbf{S}^* = \left(\prod_{i=1}^{L} \mathbf{S}_{i,x_{(i \bmod \ell_{in})}}\right)$. In other words, for each output bit k we get an LWE sample with the secret $\mathbf{S}^*$ and one of two possible matrices $\mathbf{A}^{(k)}_{L+1,0}$ or $\mathbf{A}^{(k)}_{L+1,1}$ depending on the value of that bit. We show that under the LWE assumption the above encoding is "semantically secure", meaning that it completely hides the program P.

From Encoding to Obfuscation. We use the above idea to obfuscate the compute-and-compare program $\mathbf{CC}[f, y]$ where the function $f : \{0,1\}^{\ell_{in}} \to \{0,1\}^{\ell_{out}}$ can be computed via a polynomial-size branching program $P = (P^{(1)}, \ldots, P^{(\ell_{out})})$ and the target value is $y \in \{0,1\}^{\ell_{out}}$. To do so, we simply encode the program P as described above, but instead of choosing all of the public keys $\mathbf{A}^{(k)}_{i,j}$ randomly we choose the keys at the last level $L+1$ to satisfy $\sum_{k=1}^{\ell_{out}} \mathbf{A}^{(k)}_{L+1,y_k} = \mathbf{0}$. If y has sufficiently large (pseudo-)entropy given f than, by the leftover hash lemma, this is statistically close to choosing the public keys at random and therefore, by the semantic security of the encoding scheme for branching programs, the obfuscation does not reveal any partial information about f or y. To evaluate the obfuscated program on x, we evaluate the sequence of encoded branching programs to get LWE samples $\mathbf{D}^{(k)} \approx \mathbf{S}^* \cdot \mathbf{A}^{(k)}_{L+1,P^{(k)}(x)}$ and check if $\sum_{k=1}^{\ell_{out}} \mathbf{D}^{(k)} \approx \mathbf{0}$.

This gives us our basic obfuscation scheme but several issues remain. Firstly, it only works for functions f which can be represented by polynomial length branching programs rather than all polynomial size circuits or polynomial time Turing Machines. Secondly, in order to set the parameters in a way that balances correctness and security, we would need y to have "very large" pseudo-entropy which depends polynomially on the length of the branching program L and the security parameter λ. Ideally, we would like to only require that y has some non-trivial pseudo-entropy λ^{ε} or, better yet, just that it is computationally unpredictable given f. We show how to solve these problems via generic transformations described below.

Relation to [GKW17b]. The above technique for encoding branching programs follows closely from ideas developed by Goyal, Koppula and Waters [GKW17b] in the completely unrelated context of constructing circular-security counter-examples for bit-encryption. The technique there is used as part of a larger scheme and is not analyzed modularly. However, implicitly, their work relies on entirely different properties of the encoding compared to our work. In [GKW17b], the branching-programs being encoded are public and there is no requirement that the scheme hides them in any way. Instead, that work relies on hiding the correspondence between the components $\mathbf{C}^{(k)}_{i,b}$ of the encoded branching programs and the input bits b that they correspond to. Their scheme gives out various such components at different times and if a user collects ones corresponding to an input x on which $f(x) = y$ this can be efficiently checked. In our work, we make the correspondence between the components $\mathbf{C}^{(k)}_{i,b}$ and the bits b public, in order to allow the user to evaluate the encoded program on arbitrary inputs, but rely on hiding the actual branching program being encoded.

Upgrading Functionality and Security. Our basic obfuscation scheme for compute-and-compare programs $\mathbf{CC}[f, y]$ only works for functions f represented by branching programs of some polynomial length L and values y with very large pseudo-entropy that exceeds some polynomial bound in the security parameter λ and the branching program length L. We show a series of generic transformations to upgrade the functionality and security of our scheme.

Firstly, we can reduce the requirement on the pseudo-entropy of y to only exceeding some small threshold λ^{ε} for some constant $\varepsilon > 0$. We do so by applying a pseudo-random generator (PRG) G and using our obfuscator on the program $\mathbf{CC}[G \circ f, G(y)]$. We need an injective PRG in $\mathbf{NC}^1$ that takes as input any seed y with pseudo-entropy λ^{ε} and outputs an arbitrarily large number of pseudo-random bits. Luckily, we have such PRGs under LWE.

Secondly, we can "bootstrap" our obfuscator for branching programs into one for circuits or Turing Machines by using a (leveled) fully homomorphic encryption (FHE) scheme with decryption in $\mathbf{NC}^1$, which is known to exist under LWE. A similar type of bootstrapping was used to convert iO for branching programs into iO for circuits in [GGH$^+$13b], although in our scenario we can get away with an even simpler variant of this trick. To obfuscate the program $\mathbf{CC}[f, y]$ where f is an arbitrary circuit or Turing Machine, we first encrypt f via the FHE scheme and make the ciphertext $\mathsf{ct} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(f)$ public. We then obfuscate the program $\mathbf{CC}[\mathsf{Dec}_{\mathsf{sk}}, y]$ which is essentially a "plaintext-equality tester" that checks if an input ciphertext decrypts to y. Since $\mathsf{Dec}_{\mathsf{sk}}$ is in $\mathbf{NC}^1$, we can rely on our basic obfuscation construction for branching programs to accomplish this. To evaluate the obfuscated program on an input x we first perform a homomorphic computation over ct to derive a ciphertext $\mathsf{ct}^* = \mathsf{Enc}_{\mathsf{pk}}(f(x))$ and then run the obfuscated plaintext-equality tester on ct^*. To argue security, notice that when y has sufficient pseudo-entropy given f then the obfuscated program $\mathbf{CC}[\mathsf{Dec}_{\mathsf{sk}}, y]$ can be simulated without knowledge of sk and therefore it hides sk, y. We can then rely on the semantic security of the encryption scheme to also argue that the ciphertext ct also hides f. If the function f is represented as a Turing Machine then our obfuscator is succinct since it only encrypts f but doesn't need to run it at obfuscation time. Summarizing, the above approach generically transforms a compute-and-compare obfuscator for branching programs into one for circuits and Turing Machines.

Thirdly, we can reduce the requirement on the distribution of y even further and only insist that it is computationally unpredictable given f (for example, f may include a one-way permutation of y in its description so that y has no pseudo-entropy given f but still remains computationally unpredictable). To do so, we use the same trick as previously by taking a PRG G and obfuscating the program $\mathbf{CC}[G \circ f, G(y)]$, but now we need an injective PRG that converts any computationally unpredictable source y into a long pseudo-random output (but we no longer need the PRG to be in $\mathbf{NC}^1$). Such PRGs exist in the random oracle model or assuming the existence of extremely lossy functions (ELFs) [Zha16], which in turn exists assuming exponential security of the DDH in elliptic curve groups.

Lastly, we construct an obfuscator for multi-bit compute-and-compare programs $\mathbf{MBCC}[f, y, z](x)$ which output a message z if $f(x) = y$ and otherwise output $\perp$. We again rely on a PRG G and interpret $G(y)$ as outputting a series of blocks $G_0(y), G_1(y) \ldots, G_{\ell_{msg}}(y)$ where $\ell_{msg} := |z|$ and each block is sufficiently large. To obfuscate $\mathbf{MBCC}[f, y, z]$ we instead obfuscate a series of single-bit compute-and-compare programs $P_0 = \mathbf{CC}[G_0 \circ f, G_0(y)], P_1 = \mathbf{CC}[G_1 \circ f, u_1], \ldots, P_{\ell_{msg}} = \mathbf{CC}[G_{\ell_{msg}} \circ f, u_{\ell_{msg}}]$ where we set $u_i := G_i(y)$ if $z_i = 1$ and $u_i := \overline{G_i(y)}$ (denoting the bit-wise complement) if $z_i = 0$. Let $(\tilde{P}_0, \ldots, \tilde{P}_{\ell_{msg}})$ be the obfuscated programs. On input x we can then evaluate $\tilde{P}_0(x)$ and if it outputs 0 we output $\perp$. Otherwise we can recover each bit z_i of z by setting $z_i := \tilde{P}_i(x)$. Security of the multi-bit obfuscator follows from the security of the single-bit one and the security of the PRG.

2 Notation and Preliminaries

We use the notation $[n] \overset{\text{def}}{=} \{1, \ldots, n\}$. For a distribution or a random variable X, we let $x \leftarrow X$ denote the process of sampling x according to X. Similarly for a randomized algorithm f we let $y \leftarrow f(x)$ denote the process of running $f(x)$ with fresh randomness and taking y as the output. For a set $\mathcal{X}$ we let $x \overset{\$}{\leftarrow} \mathcal{X}$ denote sampling x uniformly at random from $\mathcal{X}$.

Statistical Distance and Entropy. We write $x \leftarrow S$ when sampling x uniformly at random from the finite set S, and $x \leftarrow A$ when sampling x using the probabilistic algorithm $A(\cdot)$. For random variables X, Y with support $\mathcal{X}, \mathcal{Y}$ respectively, we define the *statistical distance*

$$\mathbf{SD}(X, Y) \overset{\text{def}}{=} \frac{1}{2} \sum_{u \in \mathcal{X} \cup \mathcal{Y}} |\Pr[X = u] - \Pr[Y = u]| \, .$$

We write $X \overset{\mathrm{s}}{\approx}_{\varepsilon} Y$ if $\mathbf{SD}(X,Y) \leq \varepsilon$. We say that two ensembles of random variables $X = \{X_\lambda\}, Y = \{Y_\lambda\}$ are *statistically indistinguishable*, denoted by $X \overset{\mathrm{s}}{\approx} Y$, if $\mathbf{SD}(X_\lambda, Y_\lambda) = \mathsf{negl}(\lambda)$. The *min-entropy* of a random variable X, denoted as $\mathbf{H}_\infty(X)$, is defined as $\mathbf{H}_\infty(X) \overset{\text{def}}{=} -\log(\max_x \Pr[X = x])$. The *(average) conditional min-entropy* of a random variable X conditioned on a correlated variable Y, denoted as $\mathbf{H}_\infty(X|Y)$, is defined as

$$\mathbf{H}_\infty(X|Y) \overset{\text{def}}{=} -\log\left(\mathop{\mathbf{E}}_{y \leftarrow Y}\left[\max_x \Pr[X = x | Y = y]\right]\right).$$

The optimal probability of an unbounded adversary guessing X given the correlated value Y, is $2^{-\mathbf{H}_\infty(X|Y)}$. We rely on the following two lemmas.

Lemma 2.1 (Leftover Hash Lemma [ILL89, DORS08]). *Let $\mathcal{H}$ be a universal hash function family consisting of function $h : \mathcal{X} \to \mathcal{Y}$. Let X, Z be random variables such that $\mathbf{H}_\infty(X|Z) \geq \log|\mathcal{Y}| + 2\log(1/\varepsilon)$ for some $\varepsilon > 0$ and X is supported over $\mathcal{X}$. Let H, Y be uniformly random and independent over $\mathcal{H}, \mathcal{Y}$ respectively. Then $\mathbf{SD}(\ (H, H(X), Z)\ ,\ (H, Y, Z)\ ) \leq \varepsilon$.*

Lemma 2.2 ([DORS08]). *Let X, Y, Z be (possibly dependent) random variables, where the support of Z is of size $\leq 2^\ell$. Then $\mathbf{H}_\infty(X|Y,Z) \geq \mathbf{H}_\infty(X|Y) - \ell$.*

Indistinguishability, Pseudo-entropy and Unpredictability. We say that two ensembles of random variables $X = \{X_\lambda\}, Y = \{Y_\lambda\}$ are *computationally indistinguishable*, denoted by $X \overset{\mathrm{c}}{\approx} Y$, if for all (non-uniform) PPT distinguishers $\mathcal{A}$ we have $|\Pr[\mathcal{A}(1^\lambda, X_\lambda) = 1] - \Pr[\mathcal{A}(1^\lambda, Y_\lambda) = 1]| = \mathsf{negl}(\lambda)$. We define the conditional pseudo-entropy of X conditioned on Y as follows.

Definition 2.3 (Conditional (HILL) Pseudo-Entropy [HILL99, HLR07]). *Let $X = \{X_\lambda\}, Y = \{Y_\lambda\}$ be ensembles of jointly distributed random variables. We define the conditional pseudo-entropy of X conditioned on Y to be at least $\ell(\lambda)$, denoted by $\mathbf{H}_{\mathsf{HILL}}(X|Y) \geq \ell(\lambda)$ if there exist some $X' = \{X'_\lambda\}$ jointly distributed with Y such that $(X, Y) \overset{\mathrm{c}}{\approx} (X', Y)$ and $\mathbf{H}_\infty(X'_\lambda | Y_\lambda) \geq \ell(\lambda)$.*

Definition 2.4 (Computational Unpredictability). *Let $X = \{X_\lambda\}, Y = \{Y_\lambda\}$ be (possibly dependent) ensembles of random variables. We say that X is* computationally unpredictable *given Y if for all (non-uniform) PPT adversaries $\mathcal{A}$ we have $\Pr[\mathcal{A}(1^\lambda, Y_\lambda) = X_\lambda] = \mathsf{negl}(\lambda)$.*

2.1 Lattices and LWE

Notation. For any integer $q \geq 2$, we let $\mathbb{Z}_q$ denote the ring of integers modulo q. We represent elements of $\mathbb{Z}_q$ as integers in the range $(-q/2, q/2]$ and define the absolute value $|x|$ of $x \in \mathbb{Z}_q$ by taking its representative in this range. For a vector $\mathbf{c} \in \mathbb{Z}_q^n$ we write $||\mathbf{c}||_\infty \leq \beta$ if each entry c_i in $\mathbf{c}$ satisfies $|c_i| \leq \beta$. Similarly, for a matrix $\mathbf{C} \in \mathbb{Z}_q^{n \times m}$ we write $||\mathbf{C}||_\infty \leq \beta$ if each entry $c_{i,j}$ in $\mathbf{C}$ satisfies $|c_{i,j}| \leq \beta$. We say that a distribution χ over $\mathbb{Z}_q$ is β-bounded if $\Pr[|x| \leq \beta\ :\ x \leftarrow \chi] = 1$. By default, all vectors are assumed to be *row* vectors.

Lemma 2.5 ([Ajt99, GPV08, MP12]). *There exist PPT algorithms $\mathsf{TrapGen}, \mathsf{SamPre}, \mathsf{Sam}$ with the following syntax:*

- $(\mathbf{B}, \mathsf{td}) \leftarrow \mathsf{TrapGen}(1^k, 1^m, q)$ *samples a matrix $\mathbf{B} \in \mathbb{Z}_q^{k \times m}$ with a trapdoor* td*.*
- $\mathbf{C} \leftarrow \mathsf{Sam}(1^m, q)$ *samples a "small" matrix $\mathbf{C} \in \mathbb{Z}_q^{m \times m}$.*
- $\mathbf{C} \leftarrow \mathsf{SamPre}(\mathbf{B}, \mathbf{B}')$ *gets $\mathbf{B}, \mathbf{B}' \in \mathbb{Z}_q^{k \times m}$ along with a trapdoor* td *for $\mathbf{B}$ and samples a "small" matrix $\mathbf{C} \in \mathbb{Z}_q^{m \times m}$ such that $\mathbf{B}\mathbf{C} = \mathbf{B}'$.*

Given integers $k \geq 1, q \geq 2$ there exists some $m^* = O(k \log q)$, $\gamma = O(k\sqrt{\log q})$ such that for all $m \geq m^*$ we have:

1. *For any $(\mathbf{B}, \mathsf{td}) \leftarrow \mathsf{TrapGen}(1^k, 1^m, q)$, $\mathbf{B}' \in \mathbb{Z}_q^{k \times m}$, $\mathbf{C} \leftarrow \mathsf{SamPre}(\mathbf{B}, \mathbf{B}', \mathsf{td})$ we have $\mathbf{BC} = \mathbf{B}'$ and $||\mathbf{C}||_\infty \leq \gamma$ (with probability 1).*

2. *We have the statistical indistinguishability requirement $\mathbf{B} \overset{s}{\approx} \mathbf{B}'$ where $(\mathbf{B}, \mathsf{td}) \leftarrow \mathsf{TrapGen}(1^k, 1^m, q)$, $\mathbf{B}' \overset{\$}{\leftarrow} \mathbb{Z}_q^{k \times m}$.*

3. *We have the statistical indistinguishability requirement $(\mathbf{B}, \mathsf{td}, \mathbf{C}) \overset{s}{\approx} (\mathbf{B}, \mathsf{td}, \mathbf{C}')$ where $(\mathbf{B}, \mathsf{td}) \leftarrow \mathsf{TrapGen}(1^k, 1^m, q)$, $\mathbf{C} \leftarrow \mathsf{Sam}(1^m, q)$, $\mathbf{B}' \overset{\$}{\leftarrow} \mathbb{Z}_q^{k \times m}$, $\mathbf{C}' \leftarrow \mathsf{SamPre}(\mathbf{B}, \mathbf{B}', \mathsf{td})$.*

All statistical distances are negligible in k and therefore also in the security parameter λ when $k = \lambda^{\Omega(1)}$.

Learning with Errors (LWE). The learning with errors (LWE) assumption was introduced by Regev in [Reg05]. We define several variants.

Definition 2.6 ([Reg05])**.** *Let n, q be integers and χ a probability distribution over $\mathbb{Z}_q$, all parameterized by the security parameter λ. The (n, q, χ)-LWE assumption says that for all polynomial m the following distributions are computationally indistinguishable*

$$(\mathbf{A}, \mathbf{sA} + \mathbf{e}) \overset{c}{\approx} (\mathbf{A}, \mathbf{u}) \qquad : \mathbf{A} \overset{\$}{\leftarrow} \mathbb{Z}_q^{n \times m}, \mathbf{s} \overset{\$}{\leftarrow} \mathbb{Z}_q^n, \mathbf{e} \leftarrow \chi^m, \mathbf{u} \overset{\$}{\leftarrow} \mathbb{Z}_q^m.$$

The work of [ACPS09] showed that the (n, q, χ)-LWE assumption above also implies security when the secret is chosen from the error distribution χ:

$$(\mathbf{A}, \mathbf{sA} + \mathbf{e}) \overset{c}{\approx} (\mathbf{A}, \mathbf{u}) \qquad : \mathbf{A} \overset{\$}{\leftarrow} \mathbb{Z}_q^{n \times m}, \mathbf{s} \overset{\$}{\leftarrow} \chi^n, \mathbf{e} \leftarrow \chi^m, \mathbf{u} \overset{\$}{\leftarrow} \mathbb{Z}_q^m.$$

Via a simple hybrid argument, we also get security when $\mathbf{S}$ is a matrix rather than a vector:

$$(\mathbf{A}, \mathbf{SA} + \mathbf{E}) \overset{c}{\approx} (\mathbf{A}, \mathbf{U}) \qquad : \mathbf{A} \overset{\$}{\leftarrow} \mathbb{Z}_q^{n \times m}, \mathbf{S} \overset{\$}{\leftarrow} \chi^{n \times n}, \mathbf{E} \leftarrow \chi^{n \times m}, \mathbf{U} \overset{\$}{\leftarrow} \mathbb{Z}_q^{n \times m} \tag{1}$$

The above variant of (n, q, χ)-LWE is the one we will rely on in this work.

The works of [Reg05, Pei09, BLP+13] show that the LWE assumption is as hard as (quantum) solving GapSVP and SIVP under various parameter regimes. In particular, we will assume for any polynomial $p = p(\lambda)$ there exists some polynomial dimension $n = n(\lambda)$, a modulus $q = q(\lambda) = 2^{\lambda^{O(1)}}$, and a distribution $\chi = \chi(\lambda)$ which is $\beta = \beta(\lambda)$ bounded such that $q > (\lambda \cdot \beta)^p$ and the (n, q, χ)-LWE assumption holds. Furthermore we can ensure that $\mathbf{H}_\infty(\chi) \geq \omega(\log \lambda)$. We refer to the above as the LWE assumption when we don't specify parameters. This is known to be as hard as solving GapSVP and (quantum) SIVP with sub-exponential approximation factors, which is believed to be hard.

Lastly, we will rely on the following fact.

Claim 2.7. *If χ is a distribution over $\mathbb{Z}_q$ such that $\mathbf{H}_\infty(\chi) \geq \omega(\log \lambda)$ then for any polynomial $n = n(\lambda)$, the probability that $\mathbf{S} \leftarrow \chi^{n \times n}$ is invertible is $1 - \mathsf{negl}(\lambda)$.*

2.2 Fully Homomorphic Encryption

A fully homomorphic encryption (FHE) scheme $\mathsf{FHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ consists of procedures:

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ generates a public-key pk and a secret key sk.

- $\mathsf{ct} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(b)$ encrypts a bit $b \in \{0, 1\}$ under public-key pk to get a ciphertext ct.

- $b = \mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct})$ decrypts a ciphertext ct using the secret key sk.

- $\mathsf{ct}^* = \mathsf{Eval}_{\mathsf{pk}}(f, \mathsf{ct}_1, \ldots, \mathsf{ct}_n)$ homomorphically evaluates a circuit $f : \{0,1\}^n \to \{0,1\}$ over the ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_n$.

Although we assume that the scheme natively only supports 1-bit plaintexts, we can extend the notation to arbitrary length messages. For a message $x \in \{0,1\}^n$ we write $\mathsf{ct} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(x)$ to denote $\mathsf{ct} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_n)$ where $\mathsf{ct}_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(x_i)$ and x_i is the i'th bit of x. Similarly for $\mathsf{ct} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_n)$ we write $x = \mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct})$ to denote $x = (x_1, \ldots, x_n)$ where $x_i = \mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}_i)$. For a function $f : \{0,1\}^n \to \{0,1\}^m$ and $\mathsf{ct} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_n)$ we also write $\mathsf{ct}^* = \mathsf{Eval}_{\mathsf{pk}}(f, \mathsf{ct})$ to denote $\mathsf{ct}^* = (\mathsf{ct}^*_1, \ldots, \mathsf{ct}^*_m)$ where $\mathsf{ct}^*_k = \mathsf{Eval}_{\mathsf{pk}}(f^{(k)}, \mathsf{ct}_1, \ldots, \mathsf{ct}_n)$ and $f^{(k)}$ computes the k'th output-bit of f.

- *Correctness:* For all $x \in \{0,1\}^n$, and all circuits $f : \{0,1\}^n \to \{0,1\}^m$ we have

$$\Pr[\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}^*) = f(x) \ : \ (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda), \mathsf{ct} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(x), \mathsf{ct}^* = \mathsf{Eval}_{\mathsf{pk}}(f, \mathsf{ct})] = 1.$$

- *Compactness:* There exists some fixed polynomial $p(\lambda)$ such that for all $f : \{0,1\}^n \to \{0,1\}$ and all bit-ciphertexts $(\mathsf{ct}_1, \ldots, \mathsf{ct}_n)$ the ciphertext $\mathsf{ct}^* \leftarrow \mathsf{Eval}_{\mathsf{pk}}(f, \mathsf{ct}_1, \ldots, \mathsf{ct}_n)$ is of size $|\mathsf{ct}^*| = p(\lambda)$.

- *Semantic Security:* The scheme satisfies the standard notion of semantic security of encryption.

An FHE scheme has *decryption in* $\mathbf{NC}^1$ if the circuit $\mathsf{Dec}_{\mathsf{sk}}(\cdot)$ is in $\mathbf{NC}^1$. In particular, it can be computed by a circuit of some depth $d(\lambda) = O(\log \lambda)$.

We refer to the above definition as a *true FHE.* A *leveled FHE* is a relaxation where the key generation algorithm takes as an input an additional parameter 1^d and we only require correctness to hold for all circuits f of depth d. Although the size of the public key pk and the run-time of the encryption algorithm can depend on d, we still insist on the size of the secret key sk, the size of the ciphertexts ct that are produced by the encryption or evaluation algorithm, and the run-time of the decryption algorithm to only depend on the security parameter λ. Moreover, for leveled FHE with decryption in $\mathbf{NC}^1$ we still require that the decryption circuit is of some fixed depth of $d(\lambda) = O(\log \lambda)$.

Under the LWE assumption, there exists a leveled FHE scheme with decryption in $\mathbf{NC}^1$. Under the LWE assumption and an additional circular-security assumption there exists a true FHE scheme with decryption in $\mathbf{NC}^1$ [BV11, BGV12, GSW13, BV14].

2.3 Pseudo-Random Generators with Weak Seeds

We rely on pseudo-random generators (PRGs) that convert "weak" seeds (which may not be uniformly random but have some pseudo-entropy or are computationally unpredictable) into a pseudo-random output. We insist on PRGs with arbitrarily large polynomial stretch. We also consider additional restrictions, such as PRGs that are injective or are in $\mathbf{NC}^1$.

Definition 2.8 (PRGs for Weak Seeds)**.** *Let $\mathcal{D}$ be a class of distributions $D = \{D_\lambda\}$ where $(s, \mathsf{aux}) \leftarrow D_\lambda$ outputs a seed $s \in \{0,1\}^{n(\lambda)}$ and some auxiliary information* aux. *A PRG $(\mathsf{Gen}, \mathcal{G})$ for the class $\mathcal{D}$ consists of a family of polynomial-time functions $\mathcal{G} = \{G : \{0,1\}^n \to \{0,1\}^m\}$ sampled via $G \leftarrow \mathsf{Gen}(1^n, 1^m)$. We require that for every $D \in \mathcal{D}$ and for every polynomial $m(\lambda)$ we have*

$$(G, G(s), \mathsf{aux}) \overset{c}{\approx} (G, u, \mathsf{aux})$$

where $G \leftarrow \mathsf{Gen}(1^{n(\lambda)}, 1^{m(\lambda)})$, $(s, \mathsf{aux}) \leftarrow D_\lambda, u \overset{\$}{\leftarrow} \{0,1\}^{m(\lambda)}$.

- *A PRG for* α-pseudo-entropy seeds *is a PRG for the class $\mathcal{D}$ of distributions where $\mathbf{H}_{\mathsf{HILL}}(s \mid \mathsf{aux}) \geq \alpha(\lambda)$. A PRG for* unpredictable *seeds is a PRG for the class $\mathcal{D}$ of distributions where s is computationally unpredictable given* aux.

- *A PRG is* injective *if there exists some polynomial $m^*(n)$ such that for all $m \geq m^*(n)$ the function $G \leftarrow \mathsf{Gen}(1^n, 1^m)$ is injective with overwhelming probability. A PRG is* strongly injective *if G is injective even when restricted to only the first $m^*(n)$ bits of output.*
- *A PRG is in $\mathbf{NC}^1$ if for all $G \leftarrow \mathsf{Gen}(1^n, 1^m)$, we can compute G by a circuit of depth $d(n) = O(\log n)$. In particular, the depth is independent of m.*

PRGs for Pseudo-Entropy Seeds. The work of [BPR12] shows that, under the LWE assumption, the function $G_{\mathbf{A}}(\mathbf{s}) = \lfloor \mathbf{s}\mathbf{A} \rceil_p$ is a PRG for a uniformly random seed $\mathbf{s} \overset{\$}{\leftarrow} \mathbb{Z}_q^n$, where the function $G_{\mathbf{A}}$ is parameterized by a random matrix $\mathbf{A} \overset{\$}{\leftarrow} \mathbb{Z}_q^{n \times m}$ and $\lfloor \cdot \rceil_p$ is a "rounding" operation. Furthermore it is easy to show that this PRG is strongly injective and is in $\mathbf{NC}^1$. The work of [AKPW13], building on [GKPV10], shows that the above PRG is also secure for α-pseudo-entropy seeds $\mathbf{s} \in \{0,1\}^n$, when $\alpha(\lambda) = \lambda^{\varepsilon}$ for any $\varepsilon > 0$.

Claim 2.9 ([GKPV10, BPR12, AKPW13])**.** *Under LWE, there exists a PRG which is secure for λ^{ε}-pseudo-entropy seeds for any $\varepsilon > 0$. Furthermore, the PRG is strongly injective and is in $\mathbf{NC}^1$.*

PRGs for Unpredictable Seeds. PRGs for unpredictable seeds exist in the random-oracle model and can be made strongly injective by choosing a sufficiently large output size relative to the input size. Therefore we can also assume that standard cryptographic hash functions satisfy this security notion. Recently, the work of Zhandry [Zha16] constructs such PRGs using *extremely lossy functions* (ELFs), which can in turn be instantiated under the exponential hardness of DDH over elliptic curves. Moreover, Zhandry shows that the resulting PRGs are injective. (We will not rely on such PRGs being in $\mathbf{NC}^1$.)

3 Obfuscation Definitions

We begin by giving a general definition of *distributional VBB obfuscation.* To keep our definition general, we define obfuscation for a class of programs $\mathcal{P}$ without specifying how programs $P \in \mathcal{P}$ are represented (e.g., branching programs, circuits, Turing Machines). We assume that a program has an associated set of parameters $P.\mathsf{params}$ (e.g., input size, output size, circuit size, etc.) which we are not required to hide.

Definition 3.1 (Distributional VBB)**.** *Consider a family of programs $\mathcal{P}$ and let Obf be a PPT algorithm, which takes as input a program $P \in \mathcal{P}$, a security parameter $\lambda \in \mathbb{N}$, and outputs a program $\tilde{P} \leftarrow \mathsf{Obf}(1^\lambda, P)$. Let $\mathcal{D}$ be a class of distribution ensembles $D = \{D_\lambda\}_{\lambda \in \mathbb{N}}$ that sample $(P, \mathsf{aux}) \leftarrow D_\lambda$ with $P \in \mathcal{P}$. We say that Obf is an* obfuscator *for the distribution class $\mathcal{D}$ over the program family $\mathcal{P}$, if it satisfies the following properties:*

1. Functionality Preserving: *There is some negligible function $\nu(\lambda) = \mathsf{negl}(\lambda)$ such that for all programs $P \in \mathcal{P}$ with input size n we have*

$$\Pr[\forall x \in \{0,1\}^n \ : \ P(x) = \tilde{P}(x) \mid \tilde{P} \leftarrow \mathsf{Obf}(1^\lambda, P)] \geq 1 - \nu(\lambda),$$

   where the probability is over the coin tosses of Obf.

2. Distributional Virtual Black-Box: *For every (non-uniform) polynomial size adversary $\mathcal{A}$, there exists a (non-uniform) polynomial size simulator Sim, such that for every* distribution ensemble *$D = \{D_\lambda\} \in \mathcal{D}$, and every (non-uniform) polynomial size predicate $\varphi : \mathcal{P} \to \{0,1\}$:*

$$\left| \Pr_{(P,\mathsf{aux}) \leftarrow D_\lambda}[\mathcal{A}(\mathsf{Obf}(1^\lambda, P), \mathsf{aux}) = \varphi(P)] - \Pr_{(P,\mathsf{aux}) \leftarrow D_\lambda}[\mathsf{Sim}^P(1^\lambda, P.\mathsf{params}, \mathsf{aux}) = \varphi(P)] \right| = \mathsf{negl}(\lambda)$$

   where Sim^P has black-box access to the program P.

Distributional Indistinguishability. We also consider an alternative security definition called *distributional indistinguishability*, which implies distributional VBB (as discussed below) but is much simpler and easier to work with.

Definition 3.2 (Distributional Indistinguishability). *An obfuscator* Obf *for the distribution class* $\mathcal{D}$ *over a family of program* $\mathcal{P}$*, satisfies* distributional indistinguishability *if there exists a (non-uniform) PPT simulator* Sim*, such that for every distribution ensemble* $D = \{D_\lambda\} \in \mathcal{D}$*, we have*

$$(\mathsf{Obf}(1^\lambda, P), \mathsf{aux}) \overset{c}{\approx} (\mathsf{Sim}(1^\lambda, P.\mathsf{params}), \mathsf{aux}),$$

where $(P, \mathsf{aux}) \leftarrow D_\lambda$*.*

Note that distributional indistinguishability does not give the simulator black-box access to the program P at all. This definition makes sense when obfuscating *evasive programs* in which case black-box access to the program P is useless.

We now show that distributional indistinguishability implies distributional VBB (a similar but more restricted result was also shown in [BVWW16]). In more detail, to get distributional VBB for some class $\mathcal{D}$ we will need to distributional indistinguishability to hold for a slightly larger "augmented" class $\mathcal{D}' = \mathsf{aug}(\mathcal{D})$ where we can add an arbitrary 1-bit predicate of the program to the auxiliary input.

Definition 3.3 (Predicate Augmentation). *For a distribution class* $\mathcal{D}$*, we define its* augmentation under predicates*, denoted* $\mathsf{aug}(\mathcal{D})$*, as follows. For any (non-uniform) polynomial-time predicate* $\varphi : \{0,1\}^* \to \{0,1\}$ *and any* $D = \{D_\lambda\} \in \mathcal{D}$ *the class* $\mathsf{aug}(\mathcal{D})$ *includes the distribution* $D' = \{D'_\lambda\}$ *where* D'_λ *samples* $(P, \mathsf{aux}) \leftarrow D_\lambda$*, computes* $\mathsf{aux}' = (\mathsf{aux}, \varphi(P))$ *and outputs* (P, aux')*.*

Theorem 3.4. *For any family of programs* $\mathcal{P}$ *and a distribution class* $\mathcal{D}$ *over* $\mathcal{P}$*, if an obfuscator* Obf *satisfies distributional-indistinguishability (Definition 3.2) for the class of distributions* $\mathsf{aug}(\mathcal{D})$ *then it also satisfies distributional-VBB security for the distribution class* $\mathcal{D}$ *(Definition 3.1).*

Proof. Let Sim be a simulator for Obf as per the definition of distributional indistinguishability. Let $D = \{D_\lambda\} \in \mathcal{D}$ be a distribution and let $\varphi : \mathcal{P} \to \{0,1\}$ be a polynomial-time predicate. Then by the distributional indistinguishability of Obf for the class $\mathsf{aug}(\mathcal{D})$ we have:

$$(\mathsf{Obf}(1^\lambda, P), \varphi(P), \mathsf{aux}) \overset{c}{\approx} (\mathsf{Sim}(1^\lambda, P.\mathsf{params}), \varphi(P), \mathsf{aux}) \qquad (2)$$

where $(P, \mathsf{aux}) \leftarrow \mathcal{D}_\lambda$.

For any poly-time adversary $\mathcal{A}$ define the simulator $\widetilde{\mathsf{Sim}}^P(1^\lambda, P.\mathsf{params}, \mathsf{aux}) = \mathcal{A}(\mathsf{Sim}(1^\lambda, P.\mathsf{params}), \mathsf{aux})$. We claim that $\widetilde{\mathsf{Sim}}$ is a valid simulator satisfying the definition of distributional VBB security since:

$$\begin{aligned}
&\left| \Pr_{(P,\mathsf{aux})\leftarrow D_\lambda}[\mathcal{A}(\mathsf{Obf}(1^\lambda, P), \mathsf{aux}) = \varphi(P)] - \Pr_{(P,\mathsf{aux})\leftarrow D_\lambda}[\widetilde{\mathsf{Sim}}^P(1^\lambda, P.\mathsf{params}, \mathsf{aux}) = \varphi(P)] \right| \\
= &\left| \Pr_{(P,\mathsf{aux})\leftarrow D_\lambda}[\mathcal{A}(\mathsf{Obf}(1^\lambda, P), \mathsf{aux}) = \varphi(P)] - \Pr_{(P,\mathsf{aux})\leftarrow D_\lambda}[\mathcal{A}(\mathsf{Sim}(1^\lambda, P.\mathsf{params}), \mathsf{aux}) = \varphi(P)] \right| \\
= &\ \mathsf{negl}(\lambda).
\end{aligned}$$

The last line follows from (2) by defining a distinguisher $\mathcal{B}(\tilde{P}, b, \mathsf{aux})$ which outputs 1 iff $\mathcal{A}(\tilde{P}, \mathsf{aux}) = b$. This proves the theorem. □

3.1 Defining Obfuscation for Compute-and-Compare Programs

Given a program $f : \{0,1\}^{\ell_{in}} \to \{0,1\}^{\ell_{out}}$ along with a target value $y \in \{0,1\}^{\ell_{out}}$, we define the compute-and-compare program:

$$\mathbf{CC}[f, y](x) = \begin{cases} 1 & \text{if } f(x) = y \\ 0 & \text{otherwise} \end{cases}$$

We assume that programs $\mathbf{CC}[f, y]$ have some canonical description that makes it it easy to recover the components f, y from $\mathbf{CC}[f, y]$. We define three distinct classes of compute-and-compare programs depending on whether f is represented as a branching program, a circuit, or a Turing Machine.

Branching Programs. We define the class $\mathcal{P}_{\mathbf{CC}}^{\mathbf{BP}}$ of compute-and-compare programs $\mathbf{CC}[f, y]$ where f is a *permutation branching program* (see Section 4.3 for a formal definition). In this case we define $\mathbf{CC}[f, y].\mathsf{params} = (1^L, 1^{\ell_{in}}, 1^{\ell_{out}})$ where L denotes the length of the branching program f.

Circuits. We define the class $\mathcal{P}_{\mathbf{CC}}^{\mathbf{CIRC}}$ to consist of programs $\mathbf{CC}[f, y]$ where f is represented as a circuit. We define $\mathbf{CC}[f, y].\mathsf{params} = (1^{|f|}, 1^{\ell_{in}}, 1^{\ell_{out}})$ where $|f|$ denotes the circuit size.

Turing Machines. Lastly we define the class $\mathcal{P}_{\mathbf{CC}}^{\mathbf{TM}}$ of compute-and-compare programs $\mathbf{CC}[f, y]$ where the function f is given as a Turing Machine with some fixed run-time t. The main advantage of considering Turing Machines instead of circuits is that the run-time of the obfuscator $\tilde{P} \leftarrow \mathsf{Obf}(1^\lambda, \mathbf{CC}[f, y])$ and the size of the obfuscated program $\tilde{P}$ can be sub-linear in the run-time t. When we consider an obfuscator for Turing Machines, we also require that the run-time of the obfuscated program $\tilde{P}$, which is itself a Turing Machine, is $\mathsf{poly}(\lambda, t)$. We define $\mathbf{CC}[f, y].\mathsf{params} = (1^{|f|}, 1^{\ell_{in}}, 1^{\ell_{out}}, t)$ where $|f|$ denotes the Turing Machine description size and t denotes the run-time.

Classes of Distributions. We will consider distribution ensembles $\mathcal{D}$ over compute-and-compare programs where each distribution $D = \{D_\lambda\}$ in $\mathcal{D}$ is polynomial-time samplable. We define the following classes of distributions:

- *Unpredictable:* The class of unpredictable distributions $\mathcal{D}_{\mathbf{UNP}}$ consists of ensembles $D = \{D_\lambda\}$ over $(\mathbf{CC}[f, y], \mathsf{aux})$ such that y is computationally unpredictable given (f, aux). (See Definition 2.4)
- *α-Pseudo-Entropy:* For a function $\alpha(\lambda)$, the class of α-pseudo-entropy distributions $\mathcal{D}_{\alpha\text{-}\mathbf{PE}}$ consists of ensembles $D = \{D_\lambda\}$ such that $(\mathbf{CC}[f, y], \mathsf{aux}) \leftarrow D_\lambda$ satisfies $\mathbf{H}_{\mathsf{HILL}}(y \mid (f, \mathsf{aux})) \geq \alpha(\lambda)$. For a two-argument function $\alpha(\lambda, L)$, we define $\mathcal{D}_{\alpha\text{-}\mathbf{PE}}$ analogously but require $\mathbf{H}_{\mathsf{HILL}}(y \mid (f, \mathsf{aux})) \geq \alpha(\lambda, L)$ where L is the length of the branching program f.

Distributional Indistinguishability by Default. When considering obfuscation for compute-and-compare programs and the above classes of distributions, we can safely focus on distributional indistinguishability (Definition 3.2) as our default security notion since it automatically implies distributional-VBB security (Definition 3.1) by Theorem 3.4. In particular, it is easy to see that the class $\mathcal{D}_{\mathbf{UNP}}$ is already closed under predicate augmentation $\mathsf{aug}(\mathcal{D}_{\mathbf{UNP}}) \subseteq \mathcal{D}_{\mathbf{UNP}}$ since an additional 1 bit of information about y preserves unpredictability. Furthermore, for any function α, we have $\mathsf{aug}(\mathcal{D}_{\alpha\text{-}\mathbf{PE}}) \subseteq \mathcal{D}_{(\alpha-1)\text{-}\mathbf{PE}}$ since an additional 1 bit of information about y decreases its min-entropy by at most 1 bit.

4 Basic Obfuscation Construction

In this section, we construct our "basic obfuscator" for compute and compare programs $\mathbf{CC}[f, y]$ where f is a polynomial-length branching program and y has high pseudo-entropy exceeding some polynomial threshold $\alpha(\lambda, L)$ in the security parameter λ and the branching program length L. In particular, we will prove the following theorem.

Theorem 4.1. *Under the LWE assumption, there exists some polynomial $\alpha = \alpha(\lambda, L)$ in the security parameter λ and branching program length L, for which there is an obfuscator for compute-and-compare branching programs $\mathcal{P}_{\mathbf{CC}}^{\mathbf{BP}}$ which satisfies distributional indistinguishability for the class of α-pseudo-entropy distributions $\mathcal{D}_{\alpha\text{-}\mathbf{PE}}$.*

4.1 Parameters

Throughout this section we rely on the following parameters:

- q: an LWE modulus
- n, m: matrix dimensions
- χ: a distribution over $\mathbb{Z}_q$
- β: the distribution χ is β-bounded
- w: branching program width; for concreteness we can set $w = 5$

The above parameters are chosen in a way that depends on the security parameter λ and the branching program length L to ensure that the following conditions hold:

1. The modulus satisfies $q > (4m\beta)^L \lambda^{\omega(1)}$.
2. The distribution χ has super-logarithmic entropy, $\mathbf{H}_\infty(\chi) > \omega(\log \lambda)$.
3. In Lemma 2.5, if we set $k = n \cdot w$ then the values $m^* = O(k \log q)$ and $\gamma = O(k\sqrt{\log q})$ specified by the lemma satisfy $m \geq m^*$ and $\beta \geq \gamma$.
4. The (n, q, χ)-LWE assumption holds.

The general LWE assumption we discussed in the preliminaries allows us to choose the above parameters as a function of λ, L so that the above conditions are satisfied.

4.2 Directed Encodings

Definition 4.2 (Directed Encoding)**.** *Let $\mathbf{A}_i, \mathbf{A}_j \in \mathbb{Z}_q^{n\times m}$. A directed encoding of a secret $\mathbf{S} \in \mathbb{Z}_q^{n\times n}$ with respect to an edge $\mathbf{A}_i \to \mathbf{A}_j$ and noise level β is a value $\mathbf{C} \in \mathbb{Z}_q^{m\times m}$ such that $\mathbf{A}_i\mathbf{C} = \mathbf{S}\mathbf{A}_j + \mathbf{E}$ where $||\mathbf{S}||_\infty, ||\mathbf{C}||_\infty, ||\mathbf{E}||_\infty \leq \beta$. We define the set of all such encodings by:*

$$\mathcal{E}^{\beta}_{\mathbf{A}_i\to\mathbf{A}_j}(\mathbf{S}) \overset{def}{=} \{\mathbf{C} \ : \ \mathbf{A}_i\mathbf{C} = \mathbf{S}\mathbf{A}_j + \mathbf{E} \ \ \textit{and} \ \ ||\mathbf{S}||_\infty, ||\mathbf{C}||_\infty, ||\mathbf{E}||_\infty \leq \beta\}.$$

It's easy to see that the above definition implies that

$$\mathbf{C}_1 \in \mathcal{E}^{\beta_1}_{\mathbf{A}_1\to\mathbf{A}_2}(\mathbf{S}_1) \ \ , \ \ \mathbf{C}_2 \in \mathcal{E}^{\beta_2}_{\mathbf{A}_2\to\mathbf{A}_3}(\mathbf{S}_2) \quad \Rightarrow \quad \mathbf{C}_1\mathbf{C}_2 \in \mathcal{E}^{2m\beta_1\beta_2}_{\mathbf{A}_1\to\mathbf{A}_3}(\mathbf{S}_1\mathbf{S}_2) \tag{3}$$

since

$$\begin{aligned}
\mathbf{A}_1\mathbf{C}_1\mathbf{C}_2 &= (\mathbf{S}_1\mathbf{A}_2 + \mathbf{E}_1)\mathbf{C}_2 \ : \ ||\mathbf{E}_1||_\infty \leq \beta_1 \\
&= \mathbf{S}_1(\mathbf{S}_2\mathbf{A}_3 + \mathbf{E}_2) + \mathbf{E}_1\mathbf{C}_2 \ : \ ||\mathbf{E}_2||_\infty \leq \beta_2 \\
&= \mathbf{S}_1\mathbf{S}_2\mathbf{A}_3 + \mathbf{S}_1\mathbf{E}_2 + \mathbf{E}_1\mathbf{C}_2 \ : \ ||\mathbf{S}_1\mathbf{E}_2 + \mathbf{E}_1\mathbf{C}_2||_\infty \leq n\beta_1\beta_2 + m\beta_1\beta_2 \leq 2m\beta_1\beta_2
\end{aligned}$$

and $||\mathbf{S}_1\mathbf{S}_2||_\infty \leq n\beta_1\beta_2 \leq 2m\beta_1\beta_2$ as well as $||\mathbf{C}_1\mathbf{C}_2||_\infty \leq m\beta_1\beta_2 \leq 2m\beta_1\beta_2$.

In particular, by iteratively applying equation (3), we get the following claim:

Claim 4.3. *If $\mathbf{C}_i \in \mathcal{E}^{\beta}_{\mathbf{A}_i\to\mathbf{A}_{i+1}}(\mathbf{S}_i)$ for $i \in [L]$ then $(\mathbf{C}_1\mathbf{C}_2\cdots\mathbf{C}_L) \in \mathcal{E}^{\beta(2m\beta)^{L-1}}_{\mathbf{A}_1\to\mathbf{A}_{L+1}}(\mathbf{S}_1\mathbf{S}_2\cdots\mathbf{S}_L)$.*

Directed Encoding Scheme. Next we show how to create directed encodings. We construct a directed encoding scheme that lets us create an encoding $\mathbf{C}$ of a secret $\mathbf{S}$ with respect to w separate edges $\{\mathbf{A}_0 \to \mathbf{A}'_0\ ,\ \dots\ ,\ \mathbf{A}_{w-1} \to \mathbf{A}'_{w-1}\}$ simultaneously.

Construction 4.4. *We define the algorithms* $(\mathsf{DE.TrapGen}, \mathsf{DE.Encode})$*:*

- $(\mathbf{B}, \mathsf{td}_\mathbf{B}) \leftarrow \mathsf{DE.TrapGen}()$*: Output* $(\mathbf{B}, \mathsf{td}_\mathbf{B}) \leftarrow \mathsf{TrapGen}(1^{w\cdot n}, 1^m, q)$ *where* $\mathsf{TrapGen}(1^k, 1^m, q)$ *is defined in Lemma 2.5. We parse* $\mathbf{B} \in \mathbb{Z}_q^{w\cdot n\times m}$ *as* $\mathbf{B} = \begin{bmatrix} \mathbf{A}_0 \\ \dots \\ \mathbf{A}_{w-1} \end{bmatrix}$ *with* $\mathbf{A}_i \in \mathbb{Z}_q^{n\times m}$*.*

- $\mathbf{C} \leftarrow \mathsf{DE.Encode}(\mathbf{B} \to \mathbf{B}', \mathbf{S}, \mathsf{td}_\mathbf{B})$*: Parse* $\mathbf{B} = \begin{bmatrix} \mathbf{A}_0 \\ \dots \\ \mathbf{A}_{w-1} \end{bmatrix}$, $\mathbf{B}' = \begin{bmatrix} \mathbf{A}'_0 \\ \dots \\ \mathbf{A}'_{w-1} \end{bmatrix}$*. Set*

$$\mathbf{H} := (\mathbf{I}_w \otimes \mathbf{S}) \cdot \mathbf{B}' + \mathbf{E} = \begin{bmatrix} \mathbf{S}\cdot\mathbf{A}'_0 + \mathbf{E}_0 \\ \dots \\ \mathbf{S}\cdot\mathbf{A}'_{w-1} + \mathbf{E}_{w-1} \end{bmatrix} \quad \text{where} \quad \mathbf{E} = \begin{bmatrix} \mathbf{E}_0 \\ \dots \\ \mathbf{E}_{w-1} \end{bmatrix} \leftarrow \chi^{w\cdot n\times m}.$$

  Output $\mathbf{C} \leftarrow \mathsf{SamPre}(\mathbf{B}, \mathbf{H}, \mathsf{td}_\mathbf{B})$ *where the* SamPre *algorithm is defined in Lemma 2.5.*

We prove two properties for the above directed encoding scheme. One is a correctness property, saying that the value $\mathbf{C}$ sampled above is indeed an encoding of $\mathbf{S}$ along each of the w edges $\mathbf{A}_j \to \mathbf{A}'_j$. The second is a security property, saying that if we encode the same secret $\mathbf{S}$ many times with respect to different sets of edges $\mathbf{B}_k \to \mathbf{B}'_k$ then this can be simulated without knowing either $\mathbf{B}_k$ or $\mathbf{B}'_k$.

Claim 4.5 (Correctness)**.** *For every* $\mathbf{S}$ *with* $||\mathbf{S}||_\infty \le \beta$*, for all* $(\mathbf{B}, \mathsf{td}_\mathbf{B}) \leftarrow \mathsf{DE.TrapGen}()$*, all* $\mathbf{B}' \in \mathbb{Z}_q^{w\cdot n\times m}$ *and all* $\mathbf{C} \leftarrow \mathsf{DE.Encode}(\mathbf{B} \to \mathbf{B}', \mathbf{S}, \mathsf{td}_\mathbf{B})$ *it holds that:*

$$\forall j \in \{0, \dots, w-1\} \ : \ \mathbf{C} \in \mathcal{E}^\beta_{\mathbf{A}_j \to \mathbf{A}'_j}(\mathbf{S}) \quad \text{where} \quad \mathbf{B} = \begin{bmatrix} \mathbf{A}_0 \\ \dots \\ \mathbf{A}_{w-1} \end{bmatrix}, \mathbf{B}' = \begin{bmatrix} \mathbf{A}'_0 \\ \dots \\ \mathbf{A}'_{w-1} \end{bmatrix}.$$

Proof. The SamPre algorithm outputs an encoding $\mathbf{C} \in \mathbb{Z}_q^{m\times m}$ with $||\mathbf{C}||_\infty \le \beta$ such that

$$\mathbf{BC} = (\mathbf{I}_w \otimes \mathbf{S}) \cdot \mathbf{B}' + \mathbf{E} \quad \Rightarrow \quad \forall j \in \{0, \dots, w-1\} \ \ \mathbf{A}_j\mathbf{C} = \mathbf{S}\mathbf{A}'_j + \mathbf{E}_j$$

where $\mathbf{E} \leftarrow \chi^{w\cdot n\times m}$, and thus $||\mathbf{E}||_\infty, ||\mathbf{E}_j||_\infty \le \beta$.Therefore $\mathbf{C} \in \mathcal{E}^\beta_{\mathbf{A}_j \to \mathbf{A}'_j}(\mathbf{S})$. □

Claim 4.6 (Security)**.** *Let* $\ell = \ell(\lambda)$ *be any polynomial on the security parameter. Under the* (n, q, χ)*-LWE assumption, there exists an efficiently samplable distribution* $\mathsf{DE.Sam}()$ *such that the following two distributions are computationally indistinguishable:*

$$(\mathbf{B}_k, \mathbf{B}'_k, \mathbf{C}_k, \mathsf{td}_{\mathbf{B}_k})_{k\in[\ell]} \overset{c}{\approx} (\mathbf{B}_k, \mathbf{B}'_k, \mathbf{C}'_k, \mathsf{td}_{\mathbf{B}_k})_{k\in[\ell]} \tag{4}$$

where $\mathbf{S} \leftarrow \chi^{n\times n}$ *and for all* $k \in [\ell]$*:*

$$\begin{aligned} (\mathbf{B}_k, \mathsf{td}_{\mathbf{B}_k}) &\leftarrow \mathsf{DE.TrapGen}(1^\lambda), & \mathbf{B}'_k &\overset{\$}{\leftarrow} \mathbb{Z}_q^{w\cdot n\times m}, \\ \mathbf{C}_k &\leftarrow \mathsf{DE.Encode}(\mathbf{B}_k \to \mathbf{B}'_k, \mathbf{S}, \mathsf{td}_{\mathbf{B}_k}), & \mathbf{C}'_k &\leftarrow \mathsf{DE.Sam}(). \end{aligned}$$

Proof. We define $\mathsf{DE.Sam}()$ to output $\mathsf{Sam}(1^m, q)$ where Sam is defined in Lemma 2.5. We show the indistinguishability of the above distributions with the following sequence of hybrids.

Hybrid 0. This is the left-hand side of equation (4). Here for all $k \in [\ell]$ the encodings are sampled as $\mathbf{C}_k \leftarrow \mathsf{DE.Encode}(\mathbf{B}_k \to \mathbf{B}'_k, \mathbf{S}, \mathsf{td}_{\mathbf{B}_k})$ which means sampling $\mathbf{H}_k := (\mathbf{I}_w \otimes \mathbf{S}) \cdot \mathbf{B}'_k + \mathbf{E}_k$ and computing $\mathbf{C}_k \leftarrow \mathsf{SamPre}(\mathbf{B}_k, \mathbf{H}_k, \mathsf{td}_{\mathbf{B}_k})$.

Hybrid 1. Here for all $k \in [\ell]$, we replace the matrices $\mathbf{H}_k$ in hybrid 0 with a uniformly random matrices $\mathbf{U}_k \overset{\$}{\leftarrow} \mathbb{Z}_q^{w \cdot n \times m}$. In other words, this is the distribution

$$(\mathbf{B}_k, \mathbf{B}'_k, \hat{\mathbf{C}}_k, \mathsf{td}_{\mathbf{B}_k})_{k \in [\ell]}$$

where $\hat{\mathbf{C}}_k \leftarrow \mathsf{SamPre}(\mathbf{B}_k, \mathbf{U}_k, \mathsf{td}_{\mathbf{B}_k})$.

By the (n, q, χ)-LWE assumption (Equation 1) hybrids 0 and 1 are indistinguishable. In particular, we are relying on LWE with the secret matrix $\mathbf{S}$ and $m \cdot w \cdot \ell$ samples where the coefficients come from the random matrices $\mathbf{B}'_k$ to replace all of the matrices $\{\mathbf{H}_k\}_{k \in [\ell]}$ with $\{\mathbf{U}_k\}_{k \in [\ell]}$. In the reduction from LWE, we can sample all the values $\mathbf{B}_k, \mathsf{td}_{\mathbf{B}_k}$ ourselves.

Hybrid 2. This is the right-hand side of equation (4). Here, for all $k \in [\ell]$, the encodings are sampled as $\mathbf{C}'_k \leftarrow \mathsf{DE.Sam}()$.

By Lemma 2.5, hybrids 1 and 2 are indistinguishable. In particular, we can replace $\hat{\mathbf{C}}_k \leftarrow \mathsf{SamPre}(\mathbf{B}_k, \mathbf{U}_k, \mathsf{td}_{\mathbf{B}_k})$ with $\mathbf{C}'_k \leftarrow \mathsf{DE.Sam}()$, for $k \in [\ell]$ one-by-one in a sequence of ℓ intermediate steps to show the indistinguishability of hybrids 1 and 2.

□

4.3 Obfuscating Compute-and-Compare Branching Programs

We now use the directed encoding scheme to construct and obfuscator for compute-and-compare *branching programs*. First, we formally define our notion of branching programs. Then we define an *encoding scheme for branching programs*. Lastly, we show how to turn this encoding scheme into an obfuscator.

Branching Programs (BPs). A boolean permutation branching program P of input size ℓ_{in}, length L and width w, is described by a sequence of $2L$ permutations

$$\pi_{i,b} \; : \; \{0, \ldots, w-1\} \to \{0, \ldots, w-1\}$$

for $i \in [L], b \in \{0,1\}$.

Given a program $P = (\pi_{i,b})_{i \in [L], b \in \{0,1\}}$ we can evaluate $P(x)$ on an arbitrary input $x \in \{0,1\}^{\ell_{in}}$. We do by defining the start state $v_1 = 0$. Then in a sequence of steps $i = 1, \ldots, L$ we define $v_{i+1} = \pi_{i, x_{(i \bmod \ell_{in})}}(v_i)$. In other words, in each step i we read the bit in position $(i \bmod \ell_{in})$ of x and this determines which of the two permutations $\pi_{i,0}, \pi_{i,1}$ to apply to the current state v_i. We define the final state v_{L+1} to be the output of the program. A valid branching program ensures that $v_{L+1} \in \{0,1\}$ for all inputs x.

Note that we assume a fixed ordering in which the input bits are accessed, where the i'th step reads the input bit $(i \bmod \ell_{in})$.[3] This departs from standard definitions that allow the program to read the input in an arbitrary order. However, we can easily convert any arbitrary branching program into one that reads its input in the above fixed order by blowing up the length of the branching program by a factor of at most ℓ_{in}.

By Barrington's theorem [Bar89], any $\mathbf{NC}^1$ circuit can be converted into a branching program with constant-width $w = 5$ and polynomial-length. In particular, any circuit with input-size ℓ_{in} and depth

[3]Any other fixed access pattern where the i'th step read the t_i'th input bit would work equally well as long as the locations $t_i \in [\ell_{in}]$ are fixed/public and the same for all branching programs. We restrict ourselves to $t_i = (i \bmod \ell_{in})$ for simplicity.

d can be computed by a branching program of length $\ell_{in} \cdot 4^d$, where the factor of ℓ_{in} comes from our insistence on having a fixed access pattern to the input.

We also consider a branching program P with ℓ_{out}-bit output to consist of ℓ_{out} separate boolean branching programs $P = \left(P^{(k)}\right)_{k\in[\ell_{out}]}$ for each output bit, where all the programs have a common length L, width w, and access pattern in which the i'th level reads the input bit ($i \bmod \ell_{in}$). To evaluate $P(x)$ we separately evaluate each of the programs $P^{(k)}(x)$ to get each output bit.

Encoding BPs. We first show how to encode a permutation branching program $P = \left(P^{(k)}\right)_{k\in[\ell_{out}]}$ with ℓ_{out}-bit output. This is not an obfuscation scheme yet since it does not allow us to evaluate the encoded program and learn the output. Instead, when we encode the program, we specify two matrices $\mathbf{A}_0^{(k)}, \mathbf{A}_1^{(k)}$ for each output bit $k \in [\ell_{out}]$. When we evaluate the encoded branching program on some input x we will get LWE tuples $\mathbf{D}^{(k)} \approx \mathbf{S}^*\mathbf{A}^{(k)}_{P^{(k)}(x)}$ with respect to some common secret $\mathbf{S}^*$ and matrices $\mathbf{A}^{(k)}_{P^{(k)}(x)}$ that depend on the output $P(x)$.

Construction 4.7. *We define the algorithms* ($\mathsf{BP.Encode}, \mathsf{BP.Eval}$) *as follows.*

- $\mathsf{BP.Encode}\left(P, (\mathbf{A}_0^{(k)}, \mathbf{A}_1^{(k)})_{k\in[\ell_{out}]}\right)$: *Takes as input* $\mathbf{A}_0^{(k)}, \mathbf{A}_1^{(k)} \in \mathbb{Z}_q^{n\times m}$ *and a branching program* $P = \left(P^{(k)}\right)_{k\in[\ell_{out}]}$ *with* ℓ_{in}*-bit input,* ℓ_{out}*-bit output and length* L*. Parse* $P^{(k)} = (\pi^{(k)}_{i,b})_{i\in[L],b\in\{0,1\}}$.
  - *For* $k \in [\ell_{out}]$ *and* $i \in [L]$ *sample* $(\mathbf{B}_i^{(k)}, \mathsf{td}_{\mathbf{B}_i^{(k)}}) \leftarrow \mathsf{DE.TrapGen}()$ *with* $\mathbf{B}_i^{(k)} = \begin{bmatrix} \mathbf{A}^{(k)}_{i,0} \\ \dots \\ \mathbf{A}^{(k)}_{i,w-1} \end{bmatrix}$.
  - *For* $k \in [\ell_{out}]$ *sample the matrix* $\mathbf{B}^{(k)}_{L+1} = \begin{bmatrix} \mathbf{A}^{(k)}_{L+1,0} \\ \dots \\ \mathbf{A}^{(k)}_{L+1,w-1} \end{bmatrix}$ *by setting* $\mathbf{A}^{(k)}_{L+1,0} := \mathbf{A}^{(k)}_0, \mathbf{A}^{(k)}_{L+1,1} := \mathbf{A}^{(k)}_1$ *and sampling* $\mathbf{A}^{(k)}_{L+1,j} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$ *for* $j \in \{2, \dots, w-1\}$.
  - *For* $i \in [L]$, $b \in \{0,1\}$, *sample* $\mathbf{S}_{i,b} \leftarrow \chi^{n\times n}$.
  - *For* $i \in [L]$, $b \in \{0,1\}$, $k \in [\ell_{out}]$ *sample:* $\mathbf{C}^{(k)}_{i,b} \leftarrow \mathsf{DE.Encode}(\mathbf{B}_i^{(k)} \to \pi^{(k)}_{i,b}(\mathbf{B}^{(k)}_{i+1}), \mathbf{S}_{i,b}, \mathsf{td}_{\mathbf{B}_i^{(k)}})$,
    where (abusing notation) we define $\pi(\mathbf{B}_i^{(k)}) = \begin{bmatrix} \mathbf{A}^{(k)}_{i,\pi(0)} \\ \dots \\ \mathbf{A}^{(k)}_{i,\pi(w-1)} \end{bmatrix}$.
  - *Finally, output the sequence* $\widehat{P} = \left(\mathbf{A}^{(k)}_{1,0}, \left(\mathbf{C}^{(k)}_{i,b}\right)_{i\in[L],b\in\{0,1\}}\right)_{k\in[\ell_{out}]}$.
- $\mathsf{BP.Eval}(\widehat{P}, x)$. *To evaluate* $\widehat{P}$ *on input* $x \in \{0,1\}^{\ell_{in}}$*, the evaluation algorithm for all* $k \in [\ell]$ *computes*

$$\mathbf{D}^{(k)} := \mathbf{A}^{(k)}_{1,0} \cdot \left(\prod_{i=1}^{L} \mathbf{C}^{(k)}_{i,x_{(i \bmod \ell_{in})}}\right)$$

  and outputs the sequence $\left(\mathbf{D}^{(k)}\right)_{k\in[\ell_{out}]}$.

We now analyze a correctness and a security property that the above scheme satisfies. The correctness property says that when we evaluate the encoded program $\widehat{P}$ on x we get LWE samples $\mathbf{D}^{(k)} \approx \mathbf{S}^*\mathbf{A}^{(k)}_{P^{(k)}(x)}$ with respect to some common secret $\mathbf{S}^*$. The security property says that the above encoding completely hides the branching program P (and in particular the choice of permutation $\pi^{(k)}_{i,b}$) being encoded.

Claim 4.8 (Correctness)**.** *For every branching program $P = \left(P^{(k)}\right)_{k\in[\ell_{out}]}$ with ℓ_{in}-bit input and ℓ_{out}-bit output, for all choices of $(\mathbf{A}_0^{(k)}, \mathbf{A}_1^{(k)})_{k\in[\ell_{out}]}$, and for all $x \in \{0,1\}^{\ell_{in}}$ the following holds. For*

$$\widehat{P} \leftarrow \mathsf{BP.Encode}\left(P, (\mathbf{A}_0^{(k)}, \mathbf{A}_1^{(k)})_{k\in[\ell_{out}]}\right) \quad , \quad \left(\mathbf{D}^{(k)}\right)_{k\in[\ell_{out}]} = \mathsf{BP.Eval}(\widehat{P}, x)$$

there exist $\mathbf{S}^* \in \mathbb{Z}_q^{n\times n}$, $\mathbf{E}^{(k)} \in \mathbb{Z}_q^{n\times m}$ such that $\mathbf{D}^{(k)} = \mathbf{S}^* \cdot \mathbf{A}^{(k)}_{P^{(k)}(x)} + \mathbf{E}^{(k)}$ and $||\mathbf{E}^{(k)}||_\infty \le \beta(2m\beta)^{L-1}$.

Proof. In the evaluation of $P^{(k)}$ on x, let $v_1^{(k)} := 0$ and for $i \in \{2,\dots,L\}$ let $v_{i+1}^{(k)} = \pi^{(k)}_{i,x_i}(v_i)$ be the states during the execution of the branching program so that $v^{(k)}_{L+1} = P^{(k)}(x)$. For convenience, we define $x_i = x_{(i \bmod \ell_{in})}$ for $i \in [L]$.

In the creation of $\widehat{P}$, we choose $\mathbf{C}^{(k)}_{i,x_i} \leftarrow \mathsf{DE.Encode}(\mathbf{B}_i^{(k)} \to \pi^{(k)}_{i,x_i}(\mathbf{B}^{(k)}_{i+1}), \mathbf{S}_{i,x_i}, \mathsf{td}_{\mathbf{B}_i^{(k)}})$ where

$$\mathbf{B}_i^{(k)} = \begin{bmatrix} \mathbf{A}^{(k)}_{i,0} \\ \dots \\ \mathbf{A}^{(k)}_{i,w-1} \end{bmatrix} \quad , \quad \pi^{(k)}_{i,x_i}(\mathbf{B}^{(k)}_{i+1}) = \begin{bmatrix} \mathbf{A}^{(k)}_{i+1,\pi^{(k)}_{i,x_i}(0)} \\ \dots \\ \mathbf{A}^{(k)}_{i+1,\pi^{(k)}_{i,x_i}(w-1)} \end{bmatrix}.$$

By Claim 4.5, we therefore have $\mathbf{C}^{(k)}_{i,x_i} \in \mathcal{E}^{\beta}_{\mathbf{A}^{(k)}_{i,v_i} \to \mathbf{A}^{(k)}_{i+1,v_{i+1}}}(\mathbf{S}_{i,x_i})$. By Claim 4.3, we have $\left(\prod_{i=1}^L \mathbf{C}^{(k)}_{i,x_i}\right) \in \mathcal{E}^{\beta(2m\beta)^{L-1}}_{\mathbf{A}^{(k)}_{1,0} \to \mathbf{A}^{(k)}_{L+1,P^{(k)}(x)}}(\mathbf{S}^*)$ where $\mathbf{S}^* = \prod_{i=1}^L \mathbf{S}_{i,x_i}$. By the definition of directed encodings, this means that

$$\mathbf{D}^{(k)} = \mathbf{A}^{(k)}_{1,0}\left(\prod_{i=1}^{L} \mathbf{C}^{(k)}_{i,x_i}\right) = \mathbf{S}^*\mathbf{A}^{(k)}_{P^{(k)}(x)} + \mathbf{E}^{(k)}$$

where $||\mathbf{S}^*||_\infty, ||\mathbf{E}^{(k)}||_\infty \le \beta(2m\beta)^{L-1}$. $\square$

Claim 4.9 (Security)**.** *Under the (n,q,χ)-LWE assumption, there exists a PPT simulator $\widehat{\mathsf{Sim}}$, such that for all ensembles of permutation branching programs P of length L input size ℓ_{in} and output-size ℓ_{out} (all parameterized by λ), the following two distributions are indistinguishable*

$$\mathsf{BP.Encode}\left(P, (\mathbf{A}_0^{(k)}, \mathbf{A}_1^{(k)})_{k\in[\ell_{out}]}\right) \overset{c}{\approx} \widehat{\mathsf{Sim}}(1^\lambda, (1^L, 1^{\ell_{in}}, 1^{\ell_{out}})) \tag{5}$$

where $\mathbf{A}_0^{(k)}, \mathbf{A}_1^{(k)} \overset{\$}{\leftarrow} \mathbb{Z}_q^{n\times m}$.

Proof. For this proof, let $\mathsf{DE.Sam}()$ be the algorithm from Claim 4.6. First, we define the simulator $\widehat{\mathsf{Sim}}(1^\lambda, (1^L, 1^{\ell_{in}}, 1^{\ell_{out}}))$ in the following way.

- Sample $\mathbf{A}^{(k)}_{1,0} \overset{\$}{\leftarrow} \mathbb{Z}_q^{n\times m}$ for $k \in [\ell_{out}]$.
- Sample $\mathbf{C}^{(k)}_{i,b} \leftarrow \mathsf{DE.Sam}()$ for $k \in [\ell_{out}]$, $b \in \{0,1\}$ and $i \in [L]$.
- Output the simulated encoding $\widehat{P} = \left(\mathbf{A}^{(k)}_{1,0}, \left(\mathbf{C}^{(k)}_{i,b}\right)_{i\in[L], b\in\{0,1\}}\right)_{k\in[\ell_{out}]}$.

We prove indistinguishability via a sequence of hybrids defined below for $j \in \{0,\dots,L\}$.

Hybrid $H_{j,0}$:

$\mathbf{B}_{L+1} \leftarrow \mathbb{Z}_q^{w\cdot n\times m}$

```
for $i \in [L]$, $k \in [\ell_{out}]$ and $b \in \{0,1\}$ do
    if $i > j$ then
        $\mathbf{B}_i^{(k)} \xleftarrow{\$} \mathbb{Z}_q^{w\cdot n\times m}$
        $\mathbf{C}_{i,b}^{(k)} \leftarrow \mathsf{DE.Sam}()$
    else
        $(\mathbf{B}_i^{(k)}, \mathsf{td}_{\mathbf{B}_i^{(k)}}) \leftarrow \mathsf{DE.TrapGen}()$
        $\mathbf{S}_{i,b} \leftarrow \chi^{n\times n}$
        $\mathbf{C}_{i,b}^{(k)} \leftarrow \mathsf{DE.Encode}(\mathbf{B}_i^{(k)} \to \pi_{i,b}^{(k)}(\mathbf{B}_{i+1}^{(k)}), \mathbf{S}_{i,b}, \mathsf{td}_{\mathbf{B}_i^{(k)}})$
    end if
end for
output the sequence $\widehat{P} = \left(\mathbf{A}_{1,0}^{(k)}, \left(\mathbf{C}_{i,b}^{(k)}\right)_{i\in[L],b\in\{0,1\}}\right)_{k\in[\ell_{out}]}$.
```

Hybrid $H_{j,1}$: This is the same as $H_{j,0}$ except that when $i = j$ we now sample:

$$\mathbf{C}_{i,0}^{(k)} \leftarrow \mathsf{DE.Sam}()$$

Hybrid $H_{j,2}$: This is the same as $H_{j,0}$ except that when $i = j$ we now sample:

$$\mathbf{C}_{i,0}^{(k)} \leftarrow \mathsf{DE.Sam}(), \mathbf{C}_{i,1}^{(k)} \leftarrow \mathsf{DE.Sam}()$$

Notice that $H_{L,0}$ and $H_{0,0}$ are respectively the left-hand and right-hand sides of Eq (5). The proof of security then follows from the following observations.

- For all j, $H_{j,0} \overset{c}{\approx} H_{j,1}$.
  This follows by applying security Claim 4.6 on directed encodings $\left(\mathbf{C}_{j,0}^{(k)}\right)_{k\in[\ell_{out}]}$ with the secret $\mathbf{S}_{j,0}$ and the matrices $\mathbf{B}_k = \mathbf{B}_j^{(k)}$, $\mathbf{B}'_k = \pi_{j,b}^{(k)}(\mathbf{B}_{j+1}^{(k)})$. Since in hybrids $H_{j,0}, H_{j,1}$ the matrix $\mathbf{B}_{j+1}^{(k)}$ is uniformly random, so is $\mathbf{B}'_k$. Note that in the reduction, we can sample all the other matrices $\mathbf{B}_i^{(k)}$ for $i \notin \{j, j+1\}$ and also all the encodings $\mathbf{C}_{i,b}^{(k)}$ for $(i,b) \neq (j,0)$ ourselves. To sample the encodings $\mathbf{C}_{j,1}^{(k)}$ we need the trapdoor $\mathsf{td}_{\mathbf{B}_j^{(k)}}$ which is provided by the security game in Claim 4.6.

- For all j, $H_{j,1} \overset{c}{\approx} H_{j,2}$.
  This follows identically as above by applying security Claim 4.6 on directed encodings $\left(\mathbf{C}_{j,1}^{(k)}\right)_{k\in[\ell_{out}]}$ with the secret $\mathbf{S}_{j,1}$ and the matrices $\mathbf{B}_k = \mathbf{B}_j^{(k)}$, $\mathbf{B}'_k = \pi_{j,b}^{(k)}(\mathbf{B}_{j+1}^{(k)})$.

- For all j, $H_{j,2} \overset{c}{\approx} H_{j-1,0}$.
  The only difference between these hybrids is that in $H_{j,2}$ we sample $(\mathbf{B}_j^{(k)}, \mathsf{td}_{\mathbf{B}_j^{(k)}}) \leftarrow \mathsf{DE.TrapGen}()$ while in $H_{j-1,0}$ we sample $\mathbf{B}_j^{(k)} \xleftarrow{\$} \mathbb{Z}_q^{w\cdot n\times m}$. This is indistinguishable by Lemma 2.5.

Combining the above we get that $H_{0,0} \overset{c}{\approx} H_{L,0}$ which proves the claimed Eq (5). □

Obfuscating BPs. Finally, we are ready to construct an obfuscator for compute-and-compare programs $\mathbf{CC}[f, y]$ where $f : \{0,1\}^{\ell_{in}} \to \{0,1\}^{\ell_{out}}$ is a permutation branching program of length L. To do so, we simply use the BP encoding scheme to encode f but we choose the matrices $\mathbf{A}_0^{(k)}, \mathbf{A}_1^{(k)}$ to satisfy $\sum_{k=1}^{\ell_{out}} \mathbf{A}_{y_k}^{(k)} = \mathbf{0}$. Then, to evaluate the obfuscated program on x, we evaluate the encoded program to get matrices $\mathbf{D}^{(k)}$ and check that $\sum_{k=1}^{\ell_{out}} \mathbf{D}^{(k)} \approx \mathbf{0}$.

Construction 4.10. *Our construction of an obfuscator* Obf *for compute-and-compare branching programs is defined as follows.*

$\mathsf{Obf}(1^\lambda, \mathbf{CC}[f,y])$ *: Let f be a BP with input size ℓ_{in}, output size ℓ_{out}, length L and width w.*

- *For all $k \in [\ell_{out}], b \in \{0,1\}$, except for $(k,b) = (\ell_{out}, y_{\ell_{out}})$, sample $\mathbf{A}_0^{(k)}, \mathbf{A}_1^{(k)} \leftarrow \mathbb{Z}_q^{n\times m}$.*
- *Set $\mathbf{A}_{y_{\ell_{out}}}^{(\ell_{out})} := -\sum_{k=1}^{\ell_{out}-1} \mathbf{A}_{y_k}^{(k)}$.*
- *Set $\widehat{f} \leftarrow \mathsf{BP.Encode}\left(f, (\mathbf{A}_0^{(k)}, \mathbf{A}_1^{(k)})_{k\in[\ell_{out}]}\right)$.*
- *Create a program $\tilde{P}[\widehat{f}]$ that takes as input $x \in \{0,1\}^{\ell_{in}}$ and does the following:*
  - *Compute $\left(\mathbf{D}^{(k)}\right)_{k\in[\ell_{out}]} = \mathsf{BP.Eval}(\widehat{f}, x)$. Let $\mathbf{D}^* = \sum_{k=1}^{\ell_{out}} \mathbf{D}^{(k)}$.*
  - *If $||\mathbf{D}^*||_\infty \le \ell_{out} \cdot \beta \cdot (2m\beta)^{L-1}$ then output 1 and otherwise output 0.*

  Output $\tilde{P}[\widehat{f}]$.

We now show that our obfuscator satisfies correctness and security.

Claim 4.11 (Correctness)**.** *There exists a negligible function $\nu(\lambda) = \mathsf{negl}(\lambda)$ such that for all branching programs f with input size ℓ_{in} and output size ℓ_{out} and for all $y \in \{0,1\}^{\ell_{out}}$ we have*

$$\Pr_{\tilde{P}\leftarrow\mathsf{Obf}(1^\lambda,\mathbf{CC}[f,y])}\left[\forall x \in \{0,1\}^{\ell_{in}} \ : \ \tilde{P}(x) = \mathbf{CC}[f,y](x)\right] \ge 1 - \nu(\lambda).$$

Proof. Let $\tilde{P}[\widehat{f}] \leftarrow \mathsf{Obf}(1^\lambda, \mathbf{CC}[f,y])$ with $\widehat{f} \leftarrow \mathsf{BP.Encode}\left(f, (\mathbf{A}_0^{(k)}, \mathbf{A}_1^{(k)})_{k\in[\ell_{out}]}\right)$. By Claim 4.8, for all $x \in \{0,1\}^{\ell_{in}}$ we have that $\left(\mathbf{D}^{(k)}\right)_{k\in[\ell_{out}]} = \mathsf{BP.Eval}(\widehat{f}, x)$ satisfies $\mathbf{D}^{(k)} = \mathbf{S}^* \cdot \mathbf{A}_{f^{(k)}(x)}^{(k)} + \mathbf{E}^{(k)}$ for some $\mathbf{S}^*, \mathbf{E}^{(k)}$ with $||\mathbf{E}^{(k)}||_\infty \le \beta(2m\beta)^{L-1}$.

Note that $\mathbf{S}^*$ is a product of matrices $\mathbf{S}_{i,b} \leftarrow \chi^{n\times n}$ each of which is invertible with overwhelming probability by Claim 2.7. Therefore, with overwhelming probability, for all x, the matrix $\mathbf{S}^*$ derived as above is invertible. Let us condition on this event for the remainder of the argument.

For all x such that $f(x) = y$, we have $\mathbf{D}^* = \sum_{k=1}^{\ell_{out}} \mathbf{D}^{(k)} = \sum_{k=1}^{\ell_{out}} (\mathbf{S}^*\mathbf{A}_{y_k}^{(k)} + \mathbf{E}^{(k)}) = \sum_{k=1}^{\ell_{out}} \mathbf{E}^{(k)}$ which satisfies $||\mathbf{D}^*||_\infty \le \ell_{out} \cdot \beta \cdot (2m\beta)^{L-1}$. Therefore, $\tilde{P}[\widehat{f}](x) = \mathbf{CC}[f,y](x) = 1$ with probability 1.

For all x such that $f(x) \neq y$, we have $\mathbf{D}^* = \sum_{k=1}^{\ell_{out}} \mathbf{D}^{(k)} = \sum_{k=1}^{\ell_{out}} (\mathbf{S}^*\mathbf{A}_{f^{(k)}(x)}^{(k)} + \mathbf{E}^{(k)})$. Since $||\sum_{k=1}^{\ell_{out}} \mathbf{E}^{(k)}||_\infty \le \ell_{out} \cdot \beta \cdot (2m\beta)^{L-1}$ the only way that $||\mathbf{D}^*||_\infty \le \ell_{out} \cdot \beta \cdot (2m\beta)^{L-1}$ could occur is if $||\sum_{k=1}^{\ell_{out}} \mathbf{S}^*\mathbf{A}_{f^{(k)}(x)}^{(k)}||_\infty \le 2\ell_{out} \cdot \beta \cdot (2m\beta)^{L-1}$. Since $\mathbf{S}^*$ is invertible and independent of the uniformly random matrices $\mathbf{A}_{f^{(k)}(x)}^{(k)}$, the sum $\sum_{k=1}^{\ell_{out}} \mathbf{S}^*\mathbf{A}_{f^{(k)}(x)}^{(k)}$ is uniformly random over $\mathbb{Z}_q^{n\times m}$. Therefore, the probability that for any such x we have $||\sum_{k=1}^{\ell_{out}} \mathbf{S}^*\mathbf{A}_{f^{(k)}(x)}^{(k)}||_\infty \le 2\ell_{out} \cdot \beta \cdot (2m\beta)^{L-1}$ is $\le \frac{2\ell_{out}\cdot\beta\cdot(2m\beta)^{L-1}}{q}$. By the union bound, the probability that there exists such an x is

$$\le \frac{2^{\ell_{in}} \cdot 2 \cdot \ell_{out} \cdot \beta \cdot (2m\beta)^{L-1}}{q} \le \frac{\ell_{out} \cdot (4m\beta)^L}{q} \le \lambda^{-\omega(1)}.$$

Therefore the probability that there exists an x such that $f(x) \neq y$ and $\tilde{P}[\widehat{f}](x) = 1$ is negligible. □

Claim 4.12 (Security)**.** *Let $\alpha(\lambda, L) = n \cdot m \cdot \log(q) + \omega(\log\lambda) = \mathsf{poly}(\lambda, L)$. Then there exists a PPT simulator Sim, such that for every distribution ensemble $D = \{D_\lambda\} \in \mathcal{D}_{\alpha-\mathbf{PE}}$ over $\mathcal{P}_{\mathbf{CC}}^{\mathbf{BP}}$ (see Section 3.1) the following two distributions are indistinguishable*

$$(\mathsf{Obf}(1^\lambda, \mathbf{CC}[f,y]), \mathsf{aux}) \overset{c}{\approx} (\mathsf{Sim}(1^\lambda, (1^L, 1^{\ell_{in}}, 1^{\ell_{out}})), \mathsf{aux})$$

where $(\mathbf{CC}[f,y], \mathsf{aux}) \leftarrow D_\lambda$.

Proof. Define the simulator $\mathsf{Sim}(1^\lambda, (1^L, 1^{\ell_{in}}, 1^{\ell_{out}}))$ to run $\widehat{f} \leftarrow \widehat{\mathsf{Sim}}(1^\lambda, (1^L, 1^{\ell_{in}}, 1^{\ell_{out}}))$, where $\widehat{\mathsf{Sim}}$ is the BP encodings simulator from Claim 4.9, and output the program $\tilde{P}[\widehat{f}]$.

To show indistinguishability we rely on a hybrid argument. We define the following distributions.

REAL: This is the real distribution $(\tilde{P}, \mathsf{aux})$ where $(\mathbf{CC}[f,y], \mathsf{aux}) \leftarrow D_\lambda$ and $\tilde{P}[\widehat{f}] \leftarrow \mathsf{Obf}(1^\lambda, \mathbf{CC}[f,y])$.

HYB: This is the distribution $(\tilde{P}, \mathsf{aux})$ where $(\mathbf{CC}[f,y], \mathsf{aux}) \leftarrow D_\lambda$ but we modify the execution of $\tilde{P}[\widehat{f}] \leftarrow \mathsf{Obf}(1^\lambda, \mathbf{CC}[f,y])$ and select $\mathbf{A}^{(\ell_{out})}_{y_{\ell_{out}}} \overset{\$}{\leftarrow} \mathbb{Z}_q^{n \times m}$ uniformly at random instead of setting $\mathbf{A}^{(\ell_{out})}_{y_{\ell_{out}}} := -\sum_{k=1}^{\ell_{out}-1} \mathbf{A}^{(k)}_{y_k}$.

$\mathsf{REAL} \overset{c}{\approx} \mathsf{HYB}$ are indistinguishable by the leftover-hash lemma (Lemma 2.1). In particular, notice that:

$$\mathbf{A}^{(\ell_{out})}_{y_{\ell_{out}}} = -\sum_{k=1}^{\ell_{out}-1} \mathbf{A}^{(k)}_{y_k} = \sum_{k=1}^{\ell_{out}-1} (y_k(\mathbf{A}^{(k)}_0 - \mathbf{A}^{(k)}_1) - \mathbf{A}^{(k)}_0)$$

If we define the hash function family $h(y_1, \ldots, y_{\ell_{out}-1}) = \sum_{k=1}^{\ell_{out}-1}(y_k(\mathbf{A}^{(k)}_0 - \mathbf{A}^{(k)}_1) - \mathbf{A}^{(k)}_0)$ then this family is universal. Furthermore $\mathbf{H}_{\mathsf{HILL}}((y_1, \ldots, y_{\ell_{out}-1}) | f, \mathsf{aux}) \geq \alpha - 1 = n \cdot m \cdot \log(q) + \omega(\log \lambda)$ by Lemma 2.2 (and the definition of HILL entropy). Therefore, by the leftover-hash lemma (in combination with the definition of HILL entropy), we get that $\mathbf{A}^{(\ell_{out})}_{y_{\ell_{out}}} = -\sum_{k=1}^{\ell_{out}-1} \mathbf{A}^{(k)}_{y_k}$ is indistinguishable from uniformly random even conditioned on (f, aux).

SIM: This is the simulated distribution $(\tilde{P}, \mathsf{aux})$ where $(\mathbf{CC}[f,y], \mathsf{aux}) \leftarrow D_\lambda$ and $\tilde{P}[\widehat{f}] \leftarrow \mathsf{Sim}(1^\lambda, (1^L, 1^{\ell_{in}}, 1^{\ell_{out}}))$.

The only difference from HYB is that instead of selecting $\widehat{f} \leftarrow \mathsf{BP.Encode}\left(P, (\mathbf{A}^{(k)}_0, \mathbf{A}^{(k)}_1)_{k \in [\ell_{out}]}\right)$ where the matrices $\mathbf{A}^{(k)}_b$ are uniformly random, we now select $\widehat{f} \leftarrow \widehat{\mathsf{Sim}}(1^\lambda, (1^L, 1^{\ell_{in}}, 1^{\ell_{out}}))$.

$\mathsf{HYB} \overset{c}{\approx} \mathsf{SIM}$ are indistinguishable by Claim 4.9.

Therefore $\mathsf{REAL} \overset{c}{\approx} \mathsf{SIM}$ which proves the claim. □

The combination of Claim 4.11 and Claim 4.12 proves Theorem 4.1.

5 Upgrading Functionality and Security

We now show how to upgrade the functionality and security properties of our basic obfuscation scheme using generic transformations. In particular, our basic scheme allows us to obfuscate compute-and-compare programs $\mathbf{CC}[f,y]$ where f is a branching program of length L and y has at least $\alpha(\lambda, L)$ bits of pseudo-entropy conditioned on (f, aux) for some large polynomial α. We propose a series of upgrades to the basic obfuscator as follows:

- In Section 5.1 we show how to get security for a larger class of distributions where, for any constant $\varepsilon > 0$, we only require y to have at least λ^ε bits of pseudo-entropy conditioned on (f, aux). We still require that f is a polynomial-length branching program but the amount of pseudo-entropy required from y is now independent of its length L.

- In Section 5.2 we then show how to upgrade functionality and allow f to be an arbitrary polynomial-size circuit or even a polynomial-time Turing Machine. In the latter case our obfuscator is succinct.

- In Section 5.3 we then show how to improve security even further to allow for all *unpredictable* distributions, where y is only computationally unpredictable given (f, aux).

- Lastly, in Section 5.4, we show how to extend our results to multi-bit compute and compare programs $\mathbf{MBCC}[f,y,z]$ which output a message z iff $f(x) = y$.

5.1 Compiler: From Large to Small Pseudo-Entropy

So far, we constructed an obfuscator Obf for compute-and-compare branching programs $\mathcal{P}_{\mathbf{CC}}^{\mathbf{BP}}$ with security for α-pseudo-entropy distributions $\mathcal{D}_{\alpha\text{-}\mathbf{PE}}$ where $\alpha = \alpha(\lambda, L)$ is some large polynomial that depends on the security parameter λ and the branching program length L. We now show how to convert any such obfuscator Obf into an obfuscator Obf' for $\mathcal{P}_{\mathbf{CC}}^{\mathbf{BP}}$ which satisfies distributional indistinguishability for the class of distributions $\mathcal{D}_{\alpha'\text{-}\mathbf{PE}}$ having some small amount of pseudo-entropy $\alpha'(\lambda) = \lambda^{\varepsilon}$ for any $\varepsilon > 0$.

Construction 5.1. *Let* Obf *be an obfuscator for* $\mathcal{P}_{\mathbf{CC}}^{\mathbf{BP}}$ *with distributional indistinguishability for* $\mathcal{D}_{\alpha\text{-}\mathbf{PE}}$ *for some polynomial* $\alpha = \alpha(\lambda, L)$. *Let* (Gen, G) *be an injective PRG in* $\mathbf{NC}^1$ *for* α'*-pseudo-entropy seeds, with injectivity parameter* $m^*(n)$ *as in Definition 2.8.*

$\mathsf{Obf}'(1^\lambda, \mathbf{CC}[f, y])$**:** *Let* $f : \{0,1\}^{\ell_{in}} \to \{0,1\}^{\ell_{out}}$ *be a branching program. Let* L' *be the length of the branching program needed to compute* $G \circ f$ *for* $G \leftarrow \mathsf{Gen}(1^{\ell_{out}}, \cdot)$.[4] *Set* $\ell'_{out} := \max\{\alpha(\lambda, L'), m^*(\ell_{out})\}$.

- *Sample* $G \leftarrow \mathsf{Gen}(1^{\ell_{out}}, 1^{\ell'_{out}})$. *Define* $(G \circ f)(x) = G(f(x))$.
- *Output* $\tilde{P} \leftarrow \mathsf{Obf}(1^\lambda, \mathbf{CC}[G \circ f, G(y)])$.

Theorem 5.2. *Under the stated conditions, the constructed obfuscator* Obf' *is an obfuscator for compute-and-compare branching programs* $\mathcal{P}_{\mathbf{CC}}^{\mathbf{BP}}$ *which satisfies distributional indistinguishability for the class of* α'*-pseudo-entropy distributions* $\mathcal{D}_{\alpha'\text{-}\mathbf{PE}}$.

In particular, under the LWE assumption, there exists an obfuscator Obf' *for* $\mathcal{P}_{\mathbf{CC}}^{\mathbf{BP}}$ *which satisfies distributional indistinguishability for the class* $\mathcal{D}_{\lambda^{\varepsilon}\text{-}\mathbf{PE}}$ *for any* $\varepsilon > 0$.

Proof. The correctness of the obfuscator Obf' follows from that of Obf and the fact that the PRG is injective. Furthermore, we rely on the fact that $G \circ f$ can be computed by a branching program of polynomial length L'.

To argue security, let $D' = \{D'_\lambda\} \in \mathcal{D}_{\alpha'\text{-}\mathbf{PE}}$. Define $D = \{D_\lambda\}$ to be the distribution $(G \circ f, G(y), \mathsf{aux})$ where $(\mathbf{CC}[f, y], \mathsf{aux}) \leftarrow D'_\lambda$ and $G \leftarrow \mathsf{Gen}(1^{\ell_{out}}, 1^{\ell'_{out}})$. By the security of the PRG, since y has α'-pseudo-entropy given (f, aux), we get that $G(y)$ is pseudo-random given (G, f, aux) and therefore $\mathbf{H}_{\mathsf{HILL}}(G(y) \mid G \circ f, \mathsf{aux}) \geq \ell'_{out} \geq \alpha(\lambda, L^*)$ which means that $D \in \mathcal{D}_{\alpha\text{-}\mathbf{PE}}$. Therefore, by the security of the obfuscator Obf with simulator Sim we have

$$(\mathsf{Obf}'(1^\lambda, \mathbf{CC}[f, y]), \mathsf{aux}) = (\mathsf{Obf}(1^\lambda, \mathbf{CC}[G \circ f, G(y)]), \mathsf{aux}) \overset{c}{\approx} (\mathsf{Sim}(1^\lambda, (1^{L^*}, 1^{\ell_{in}}, 1^{\ell'_{out}})), \mathsf{aux}).$$

By defining $\mathsf{Sim}'(1^\lambda, (1^L, 1^{\ell_{in}}, 1^{\ell_{out}})) = \mathsf{Sim}(1^\lambda, (1^{L'}, 1^{\ell_{in}}, 1^{\ell'_{out}}))$ we see that Sim' is a good simulator for Obf' proving that it satisfies distributional indistinguishability for the class of α'-pseudo-entropy distributions $\mathcal{D}_{\alpha'\text{-}\mathbf{PE}}$. This proves the first part of the theorem.

By instantiating the PRG with the one from Claim 2.9, the second part of the theorem follows. $\square$

How low can you go? Although the obfuscator we constructed satisfies distributional indistinguishability for distributions with even a small amount of pseudo-entropy $\alpha'(\lambda) = \lambda^{\varepsilon}$ for any $\varepsilon > 0$, the exact security necessarily deteriorates as ε approaches 0. Therefore, in applications, we will generally avoid "abusing" this theorem, and will mainly apply it on distributions where $\alpha'(\lambda)$ is a sufficiently large polynomial in λ. However, we will crucially rely on the fact that $\alpha'(\lambda)$ only depends on the security parameter λ and does not depend on the branching program length L.

[4] We rely on the fact that the circuit depth of G, and therefore also the branching-program length L', does not depend on the output length ℓ'_{out} of the PRG.

5.2 Compiler: From BPs to Circuits and TMs

Let Obf be an obfuscator for the class of compute-and-compare branching programs $\mathcal{P}^{\mathbf{BP}}_{\mathbf{CC}}$. We show how to bootstrap it to construct an obfuscator Obf' for the class of compute-and-compare circuits $\mathcal{P}^{\mathbf{CIRC}}_{\mathbf{CC}}$ or even Turing Machines $\mathcal{P}^{\mathbf{TM}}_{\mathbf{CC}}$. Our compiler relies on *fully homomorphic encryption* (FHE) with decryption in $\mathbf{NC}^1$. To obfuscate $\mathbf{CC}[f, y]$, we encrypt the circuit f via an FHE scheme and then obfuscate a "plaintext-equality tester" $\mathbf{CC}[\mathsf{Dec}_{\mathsf{sk}}, y]$ that allows users to test whether an arbitrary ciphertext decrypts to y.

Construction 5.3. *Let* $\mathsf{FHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *be a fully homomorphic encryption scheme with decryption in* $\mathbf{NC}^1$ *and let* Obf *be an obfuscator for compute-and-compare branching programs* $\mathcal{P}^{\mathbf{BP}}_{\mathbf{CC}}$. *We construct an obfuscator* Obf' *for compute-and-compare circuits* $\mathcal{P}^{\mathbf{CIRC}}_{\mathbf{CC}}$.

$\mathsf{Obf}'(1^\lambda, \mathbf{CC}[g, y])$**:** *On input a circuit* $g : \{0,1\}^{\ell_{in}} \to \{0,1\}^{\ell_{out}}$ *and* $y \in \{0,1\}^{\ell_{out}}$ *do the following.*

- *Sample* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.
- *Encrypt the circuit* g *to get* $\mathsf{ct} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(g)$.
- *Construct the polynomial-size branching program for the function* f_{sk} *defined as*

  $$f_{\mathsf{sk}}(\mathsf{ct}^*) \stackrel{def}{=} (\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}^*_1), \ldots, \mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}^*_{\ell_{out}}))$$

  where $\mathsf{ct}^* = (\mathsf{ct}^*_1, \ldots, \mathsf{ct}^*_{\ell_{out}})$ *consists of* ℓ_{out} *bit-ciphertexts.*
- *Compute the obfuscated plaintext-equality tester:* $\tilde{T} \leftarrow \mathsf{Obf}(1^\lambda, \mathbf{CC}[f_{\mathsf{sk}}, y])$.
- *Create the program* $\tilde{P}[\tilde{T}, \mathsf{pk}, \mathsf{ct}]$ *which contains* $\tilde{T}, \mathsf{pk}, \mathsf{ct}$ *hard-coded and, on input* $x \in \{0,1\}^{\ell_{in}}$, *works as follows:*
  - *Compute* $\mathsf{ct}^* = \mathsf{Eval}_{\mathsf{pk}}(U_x, \mathsf{ct})$, *where* U_x *is the universal circuit that takes as input a circuit* g *and outputs* $U_x(g) \stackrel{def}{=} g(x)$. *Output the bit* $b = \tilde{T}(\mathsf{ct}^*)$.

  Output the program $\tilde{P} = \tilde{P}[\tilde{T}, \mathsf{pk}, \mathsf{ct}]$.

The above construction also works using a leveled FHE instead of true FHE. In this case the only thing that changes is that we sample $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^d)$ where d is the depth of the universal circuit U_x that takes as input a circuit of size $|g|$.

Theorem 5.4. *Assume that* FHE *is a (leveled) fully homomorphic encryption scheme with decryption in* $\mathbf{NC}^1$ *and* Obf *is an obfuscator for compute-and-compare branching programs* $\mathcal{P}^{\mathbf{BP}}_{\mathbf{CC}}$ *which satisfies distributional indistinguishability for* α*-pseudo-entropy distributions* $\mathcal{D}_{\alpha\text{-}\mathbf{PE}}$ *where* $\alpha = \alpha(\lambda)$ *is some polynomial in (only) the security parameter* λ. *Then* Obf' *is an obfuscator for all compute-and-compare circuits* $\mathcal{P}^{\mathbf{CIRC}}_{\mathbf{CC}}$ *which satisfies distributional indistinguishability for* $\mathcal{D}_{\alpha\text{-}\mathbf{PE}}$.

In particular, under the LWE assumption, there is an obfuscator for all compute-and-compare circuits $\mathcal{P}^{\mathbf{CIRC}}_{\mathbf{CC}}$ *which satisfies distributional indistinguishability for the class* $\mathcal{D}_{\lambda^\varepsilon\text{-}\mathbf{PE}}$ *for any* $\varepsilon > 0$.

Proof. Correctness of the obfuscator Obf' follows directly from that of the obfuscator Obf and from the correctness of the FHE scheme.

To argue the security of Obf', we need to construct a simulator $\mathsf{Sim}'(1^\lambda, (1^{|g|}, 1^{\ell_{in}}, 1^{\ell_{out}}))$ that simulates $\tilde{P}[\tilde{T}, \mathsf{pk}, \mathsf{ct}]$ which is equivalent to simulating the components $(\tilde{T}, \mathsf{pk}, \mathsf{ct})$. We define the simulator Sim' to sample $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, $\mathsf{ct} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0^{|g|})$ and $\tilde{T} \leftarrow \mathsf{Sim}(1^\lambda, (1^L, 1^{\ell'_{in}}, 1^{\ell_{out}}))$ where Sim is the simulator for Obf.

To show indistinguishability, let $D' = \{D'_\lambda\}$ be any distribution in $\mathcal{D}_{\alpha\text{-}\mathbf{PE}}$ which samples $(\mathbf{CC}[g, y], \mathsf{aux}') \leftarrow D'_\lambda$. We need to show the indistinguishability of the real and simulated $(\tilde{T}, \mathsf{pk}, \mathsf{ct})$ even given aux'. We do so by introducing an intermediate hybrid distribution. In particular, we first show that $\mathsf{REAL} \overset{c}{\approx} \mathsf{HYB} \overset{c}{\approx} \mathsf{SIM}$ where:

$$\mathsf{REAL} \overset{\text{def}}{=} \left((\tilde{T}, \mathsf{pk}, \mathsf{ct}), \mathsf{aux}' \;:\; \begin{array}{c} (g, y, \mathsf{aux}') \leftarrow D'_\lambda, (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda), \mathsf{ct} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(g) \\ \tilde{T} \leftarrow \mathsf{Obf}(1^\lambda, \mathbf{CC}[f_{\mathsf{sk}}, y]) \end{array} \right)$$

$$\mathsf{HYB} \overset{\text{def}}{=} \left((\tilde{T}, \mathsf{pk}, \mathsf{ct}), \mathsf{aux}' \;:\; \begin{array}{c} (g, y, \mathsf{aux}') \leftarrow D'_\lambda, (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda), \mathsf{ct} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(g) \\ \tilde{T} \leftarrow \mathsf{Sim}(1^\lambda, (1^{L(\lambda)}, 1^{\ell'_{in}}, 1^{\ell_{out}})) \end{array} \right)$$

$$\mathsf{SIM} \overset{\text{def}}{=} \left((\tilde{T}, \mathsf{pk}, \mathsf{ct}), \mathsf{aux}' \;:\; \begin{array}{c} (\mathbf{CC}[g, y], \mathsf{aux}') \leftarrow D'_\lambda, (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda), \mathsf{ct} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0^{|g|}) \\ \tilde{T} \leftarrow \mathsf{Sim}(1^\lambda, (1^{L(\lambda)}, 1^{\ell'_{in}}, 1^{\ell_{out}})) \end{array} \right)$$

Firstly, to show $\mathsf{REAL} \overset{c}{\approx} \mathsf{HYB}$ we define the distribution $D = \{D_\lambda\}$ as follows:

- Sample $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, $(\mathbf{CC}[g, y], \mathsf{aux}') \leftarrow D'_\lambda$, $\mathsf{ct} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(g)$.
- Output $(\mathbf{CC}[f_{\mathsf{sk}}, y], \mathsf{aux} = (\mathsf{aux}', \mathsf{pk}, \mathsf{ct}))$.

Then for $(\mathbf{CC}[f_{\mathsf{sk}}, y], \mathsf{aux}) \leftarrow D_\lambda$ we have $\mathbf{H}_{\mathsf{HILL}}(y|f_{\mathsf{sk}}, \mathsf{aux}) = \mathbf{H}_{\mathsf{HILL}}(y|g, \mathsf{aux}') \geq \alpha(\lambda)$ and therefore we can rely on the security of the obfuscator Obf to argue that $\tilde{T} \leftarrow \mathsf{Obf}(1^\lambda, \mathbf{CC}[f_{\mathsf{sk}}, y])$ is indistinguishable from $\tilde{T} \leftarrow \mathsf{Sim}(1^\lambda, (1^{L(\lambda)}, 1^{\ell'_{in}}, 1^{\ell_{out}}))$ even given $\mathsf{aux} = (\mathsf{aux}', \mathsf{pk}, \mathsf{ct})$.

Next, $\mathsf{HYB} \overset{c}{\approx} \mathsf{SIM}$ simply follows from the semantic security of the FHE scheme since the secret key sk does not appear in either distribution.

Combining the above we get $\mathsf{REAL} \overset{c}{\approx} \mathsf{SIM}$ which proves the distributional indistinguishability security of the obfuscation scheme. □

Succinct Obfuscation for Turing Machines. We can also use the obfuscator Obf' constructed above as an obfuscator for the class $\mathcal{P}^{\mathbf{TM}}_{\mathbf{CC}}$ of programs $\mathbf{CC}[g, y]$ where the function $g : \{0,1\}^{\ell_{in}} \to \{0,1\}^{\ell_{out}}$ is represented as a Turing Machine with some fixed run-time t. In this case the only changes are:

- We use the FHE scheme to encrypt the Turing Machine g (instead of a circuit).
- We define $\tilde{P} = \tilde{P}[\tilde{T}, \mathsf{pk}, \mathsf{ct}, t]$ analogously to how it was defined before but include the parameter t explicitly. In the evaluation of $\tilde{P}'$, we now construct the universal circuit U_x which takes as input a Turing Machine g (instead of a circuit) and runs $g(x)$ for t steps.

The security and correctness analysis of this obfuscator remain exactly the same as in the case of circuits (Theorem 5.4). However, now our obfuscator is succinct meaning that the the run-time of the obfuscator and the size of the obfuscated program are sub-linear in the run-time t.

In particular, if we use a true FHE scheme, then our obfuscator is *truly succinct* meaning that the run-time of the obfuscation procedure $\tilde{P} \leftarrow \mathsf{Obf}(1^\lambda, \mathbf{CC}[g, y])$ and the size of the obfuscated program $|\tilde{P}|$ is $\mathsf{poly}(\lambda, |g|, |y|, \log t)$ where $|g|$ is the Turing Machine description size and t is the run-time.

If we instead use a leveled FHE, we get an obfuscator which is *weakly succinct*, meaning that the run-time of the obfuscator and the size of the obfuscated program is $\mathsf{poly}(\lambda, |g|, |y|, \log t, d)$ where d is the depth of the circuit computing g.

5.3 Compiler: From Pseudo-Entropy to Unpredictability

We now show how to upgrade the security of an obfuscator. In particular we start with an obfuscator which is secure for α-pseudo-entropy distributions $\mathcal{D}_{\alpha\text{-}\mathbf{PE}}$ for some polynomial $\alpha = \alpha(\lambda)$ and we show how to use it to construct an obfuscator Obf' which is secure for a larger class of all unpredictable distributions $\mathcal{D}_{\mathbf{UNP}}$ where y is computationally unpredictable given (g, aux); see Section 3.1.

In fact, out construction is essentially identical to Construction 5.1 which reduces the amount of pseudo-entropy needed from y by using a PRG. In this section, we also use a PRG to reduce the requirement on y from pseudo-entropy to unpredictability. In this case, we need an injective PRG for *unpredictable seeds* (but we do not need it to be in $\mathbf{NC}^1$).

Construction 5.5. *Let* Obf *be an obfuscator for compute-and-compare circuits* $\mathcal{P}^{\mathbf{CIRC}}_{\mathbf{CC}}$ *(respectively, Turing Machines* $\mathcal{P}^{\mathbf{TM}}_{\mathbf{CC}}$*) which satisfies distributional indistinguishability* α*-pseudo-entropy distributions* $\mathcal{D}_{\alpha\text{-}\mathbf{PE}}$ *for some polynomial* $\alpha = \alpha(\lambda)$*. Let* (Gen, G) *be an injective PRG for unpredictable seeds with some injectivity parameter* $m^* = m^*(\lambda, n)$ *as in Definition 2.8.*

$\mathsf{Obf}'(1^\lambda, \mathbf{CC}[f, y])$ *On input a circuit (resp. Turing Machine)* $f : \{0,1\}^{\ell_{in}} \to \{0,1\}^{\ell_{out}}$ *and* $y \in \{0,1\}^{\ell_{out}}$.

- *Sample* $G \leftarrow \mathsf{Gen}(1^{\ell_{out}}, 1^{\max\{\alpha(\lambda), m^*(\lambda, \ell_{out})\}})$*. Define* $(G \circ f)(x) = G(f(x))$.
- *Output* $\tilde{P} \leftarrow \mathsf{Obf}(1^\lambda, \mathbf{CC}[G \circ f, G(y)])$.

Theorem 5.6. *Assume* Obf *is an obfuscator for compute-and-compare circuits* $\mathcal{P}^{\mathbf{CIRC}}_{\mathbf{CC}}$ *which satisfies distributional indistinguishability for* α*-pseudo-entropy distributions* $\mathcal{D}_{\alpha\text{-}\mathbf{PE}}$ *where* $\alpha = \alpha(\lambda)$ *is some polynomial and let* (Gen, G) *be an injective PRG for unpredictable seeds. Then* Obf' *is an obfuscator for* $\mathcal{P}^{\mathbf{CIRC}}_{\mathbf{CC}}$ *which satisfies distributional indistinguishability for all unpredictable distributions* $\mathcal{D}_{\mathbf{UNP}}$.

If Obf *is a truly (resp. weakly) succinct obfuscator for the class* $\mathcal{P}^{\mathbf{TM}}_{\mathbf{CC}}$ *then so is* Obf'.

The proof of the above theorem is essentially identical to the proof of Theorem 5.2.

5.4 Compiler: From One-Bit to Multi-Bit Output

Given a function $f : \{0,1\}^{\ell_{in}} \to \{0,1\}^{\ell_{out}}$ along with a target value $y \in \{0,1\}^{\ell_{out}}$ and a message $z \in \{0,1\}^{\ell_{msg}}$, we define the multi-bit compute-and-compare program:

$$\mathbf{MBCC}[f, y, z](x) = \begin{cases} z & \text{if } f(x) = y \\ \perp & \text{otherwise} \end{cases}$$

We define the class $\mathcal{P}^{\mathbf{CIRC}}_{\mathbf{MBCC}}$ and $\mathcal{P}^{\mathbf{TM}}_{\mathbf{MBCC}}$ to consist of canonical descriptions of such programs $\mathbf{MBCC}[f, y, z]$ where f is represented as a circuit and a Turing Machine respectively.

We define classes of distributions $\mathcal{D}$ analogously to the definitions for standard compute-and-compare programs. In particular, we define the class unpredictable distributions $\mathcal{D}_{\mathbf{UNP}}$ to consist of ensembles $D = \{D_\lambda\}$ over $(\mathbf{MBCC}[f, y, z], \mathsf{aux})$ such that y is unpredictable given (f, z, aux) (See Definition 2.4). For a function $\alpha(\lambda)$, we define the class of α-pseudo-entropy distributions $\mathcal{D}_{\alpha\text{-}\mathbf{PE}}$ to consists of ensembles $D = \{D_\lambda\}$ such that $(\mathbf{MBCC}[f, y, z], \mathsf{aux}) \leftarrow D_\lambda$ satisfies $\mathbf{H}_{\mathsf{HILL}}(y \mid (f, z, \mathsf{aux})) \geq \alpha(\lambda)$.

We now show how to construct an obfuscator Obf' for multi-bit compute-and-compare programs using an obfuscator Obf for standard compute-and-compare programs along with a strongly injective PRG.

Construction 5.7. *Let* Obf *be an obfuscator for* $\mathcal{P}^{\mathbf{CIRC}}_{\mathbf{CC}}$ *(or* $\mathcal{P}^{\mathbf{TM}}_{\mathbf{CC}}$*) which satisfies distributional indistinguishability for* β*-pseudo-entropy distributions* $\mathcal{D}_{\beta\text{-}\mathbf{PE}}$*. Let* $(\mathsf{Gen}, \mathcal{G})$ *be a strongly injective PRG with injectivity parameter* $m^* = m^*(n)$ *as in Definition 2.8. Let* $\beta = \beta(\lambda)$ *be some parameter.*

$\mathsf{Obf}'(1^\lambda, \mathbf{MBCC}[f, y, z])$ *On input a circuit (or TM)* $f : \{0,1\}^{\ell_{in}} \to \{0,1\}^{\ell_{out}}$*,* $y \in \{0,1\}^{\ell_{out}}$ *and* $z \in \{0,1\}^{\ell_{msg}}$*:*

- *Let* $\gamma = \max\{m^*(\ell_{out}), \beta(\lambda)\}$.
- *Compute* $G \leftarrow \mathsf{Gen}(1^{\ell_{out}}, 1^{\gamma \cdot (\ell_{msg}+1)})$.
  We think of $G(s)$ *as outputting* $\ell_{msg} + 1$ *blocks of* γ *bits each and we use the notation* $G_i(s)$ *to denote the i'th block of output. Let* $(y_0, \ldots, y_{\ell_{msg}}) = G(y)$.

- *Compute* $\tilde{P}_0 \leftarrow \mathsf{Obf}(1^\lambda, \mathbf{CC}[G_0 \circ f, y_0])$ *and for* $i = 1, \ldots, \ell_{msg}$ *compute* $\tilde{P}_i \leftarrow \mathsf{Obf}(1^\lambda, \mathbf{CC}[G_i \circ f], y_i)$ *if* $z_i = 1$ *or* $\tilde{P}_i \leftarrow \mathsf{Obf}(1^\lambda, \mathbf{CC}[G_i \circ f, \overline{y}_i])$ *if* $z_i = 0$*. Here we use* $\overline{y}_i$ *to denote flipping all bits of of* y_i*.*
- *Output the program* $\tilde{P} = \tilde{P}[\tilde{P}_0, \tilde{P}_1, \ldots, \tilde{P}_{\ell_{out}}]$ *which gets an input* $x \in \{0,1\}^{\ell_{in}}$ *and proceeds as follows:*
  - *If* $\tilde{P}_0(x) = 0$ *then output* $\perp$*.*
  - *Else output a string* $z \in \{0,1\}^{\ell_{msg}}$ *by setting* $z_i = \tilde{P}_i(x)$*.*

Theorem 5.8. *Assume* Obf *is an obfuscator for* $\mathcal{P}^{\mathbf{CIRC}}_{\mathbf{CC}}$ *which satisfies distributional indistinguishability for* β*-pseudo-entropy distributions* $\mathcal{D}_{\beta\text{-}\mathbf{PE}}$ *and* $(\mathsf{Gen}, \mathcal{G})$ *is a strongly injective PRG for* α*-pseudo-entropy seeds for some polynomials* $\alpha(\lambda), \beta(\lambda)$ *in* λ*. Then* Obf' *constructed above is an obfuscator for* $\mathcal{P}^{\mathbf{CIRC}}_{\mathbf{MBCC}}$ *which satisfies distributional indistinguishability for* α*-pseudo-entropy distributions* $\mathcal{D}_{\alpha\text{-}\mathbf{PE}}$*. In particular, under the LWE assumption, for any constant* $\varepsilon > 0$ *there exists an obfuscator for* $\mathcal{P}^{\mathbf{CIRC}}_{\mathbf{MBCC}}$ *with distributional indistinguishability for distributions* $\mathcal{D}_{\lambda^\varepsilon\text{-}\mathbf{PE}}$*. Furthermore:*

- *If* $(\mathsf{Gen}, \mathcal{G})$ *is a strongly injective PRG for unpredictable seeds then* Obf' *satisfies distributional indistinguishability for all unpredictable distributions* $\mathcal{D}_{\mathbf{UNP}}$*.*
- *If* Obf *is a truly (resp. weakly) succinct obfuscator for Turing Machines* $\mathcal{P}^{\mathbf{TM}}_{\mathbf{CC}}$ *then* Obf' *is a truly (resp. weakly) succinct obfuscator for Turing Machines* $\mathcal{P}^{\mathbf{TM}}_{\mathbf{MBCC}}$*.*

Proof. Correctness of the obfuscator Obf follows from that of Obf' and the strong injectivity of the PRG. In particular $f(x) \neq y$ then, by injectivity, $G_0(f(x)) \neq y_0$, which means that $\tilde{P}_0(x) = 0$ and $\tilde{P}(x)$ outputs $\perp$. If $f(x) = y$ then $\tilde{P}_0(x) = 1$ and $\tilde{P}_i(x) = z_i$ so $\tilde{P}(x) = z$.

For security, let Sim be the simulator for Obf. We use it to define the simulator Sim' for Obf'. Let $\mathsf{params}_i = (1^{|G_i \circ f|}, 1^{\ell_{in}}, 1^\gamma)$ and $\mathsf{params}' = (1^{|f|}, 1^{\ell_{in}}, 1^{\ell_{out}}, 1^{\ell_{msg}})$. We define $\mathsf{Sim}'(1^\lambda, \mathsf{params}')$:

- For $i = 0, \ldots, \ell_{msg}$: sample $\tilde{P}_i \leftarrow \mathsf{Sim}(1^\lambda, \mathsf{params}_i)$.
- Output $\tilde{P}[\tilde{P}_0, \ldots, \tilde{P}_{\ell_{msg}}]$ defined the same way as in the construction of Obf'.

To prove security, we need to show that the following distributions over $(\tilde{P}_0, \ldots, \tilde{P}_{\ell_{msg}}, \mathsf{aux})$ are indistinguishable:

1. Select $(\mathbf{MBCC}[f, y, z], \mathsf{aux}) \leftarrow D_\lambda$, $G \leftarrow \mathsf{Gen}(1^{\ell_{out}}, 1^{\gamma \cdot (\ell_{msg}+1)})$ and set $\tilde{P}_i \leftarrow \mathsf{Obf}(1^\lambda, \mathbf{CC}[G_i \circ f, \hat{y}_i])$ where $\hat{y}_i = y_i$ if $z_i = 1$ and $\hat{y}_i = \overline{y}_i$ if $z_i = 0$.
2. Select $(\mathbf{MBCC}[f, y, z], \mathsf{aux}) \leftarrow D_\lambda$ and $\tilde{P}_i \leftarrow \mathsf{Sim}(1^\lambda, \mathsf{params}_i)$.

We do so via the hybrid argument where we switch the programs $\tilde{P}_i$ from distribution (1) to distribution (2) one at a time for $i = 0, \ldots, \ell_{msg}$. The PRG ensures that $(y_0, \ldots, y_{\ell_{msg}})$ is indistinguishable from uniform even given f, z, aux and therefore $\mathbf{H}_{\mathsf{HILL}}(\hat{y}_i \mid f, z, \{\hat{y}_j : j > i\}) \geq \gamma \geq \alpha$. Therefore we can rely on the security of the obfuscator for the i'th program $\mathbf{CC}[G_i \circ f, \hat{y}_i]$ by thinking of the remaining programs $\tilde{P}_j \leftarrow \mathsf{Obf}(1^\lambda, \mathbf{CC}[G_j \circ f, \hat{y}_j])$ for $j > i$ as auxiliary input in each step of the hybrid argument. □

6 Applications

Obfuscating Conjunctions. A conjunction is a function of the form (e.g.)

$$P(x_1, \ldots, x_{\ell_{in}}) = x_2 \wedge \neg x_5 \wedge x_7 \wedge \ldots \wedge \neg x_{\ell_{in}}.$$

In other words, conjunctions correspond exactly to programs $\mathbf{CC}[f, y]$ where $f : \{0,1\}^{\ell_{in}} \to \{0,1\}^{\ell_{out}}$ outputs a subset of the input bits $f(x_1, \ldots, x_{\ell_{in}}) = (x_{i_1}, \ldots, x_{i_{\ell_{out}}})$ corresponding to the variables that appear in the conjunction and $y_j = 1$ if the literal x_{i_j} appears in the conjunction or $y_j = 0$ if the literal $\neg x_{i_j}$ appears. The work of [BR13] constructed an obfuscator for conjunctions using multi-linear maps and later [BVWW16] constructed one using a non-standard variant of the Ring-LWE assumptions called "entropic Ring-LWE". Our work allows us to obfuscate a significantly larger and more expressive set of programs, but it offers several advantages even if we restrict our attention solely to conjunctions. Most importantly, our work only relies on standard LWE wheres [BVWW16] relied on entropic Ring-LWE. Additionally, we get qualitatively stronger security since our obfuscator even hides that fact that the obfuscated program is a conjunction function (rather than any other compute-and-compare program with the same size parameters). Lastly, we can get security for a broader class of distributions. The work of [BVWW16] required that y has sufficient min-entropy given f while (under appropriate assumptions) we only require that y is computationally unpredictable given f.

Affine Functions. Generalizing on conjunctions, our obfuscator allows us to obfuscate arbitrary *affine testers* which are parameterized by a matrix $\mathbf{A}$ and a vector $\mathbf{y}$ and test whether an input $\mathbf{x}$ satisfies $\mathbf{A}\mathbf{x} \stackrel{?}{=} \mathbf{y}$. This corresponds to the compute-and-compare program $\mathbf{CC}[f_{\mathbf{A}}, \mathbf{y}]$ where $f_{\mathbf{A}}(x) = \mathbf{A} \cdot \mathbf{x}$. We can think of conjunctions as a special case where the matrix $\mathbf{A}$ is a selector matrix that selects some subset of the coordinates of $\mathbf{x}$. For security, we require that $\mathbf{y}$ has pseudo-entropy or is computationally unpredictable given $\mathbf{A}$.

Private Secure Sketches. Secure sketches were introduced in [DORS08] as a tool to correct errors in some source without reducing its entropy too much. For example, we can take a "sketch" of some biometric scan in a way that allows us to correct errors in future scans of the same biometric without significantly reducing the entropy of the biometric. This is important if we want to use biometrics for identification or key derivation.

In more detail, let $\mathcal{M}$ be some metric space with distance $\Delta(\cdot, \cdot)$. A (k, ℓ, t) secure sketch over $\mathcal{M}$ consists of algorithms $(\mathsf{SS}, \mathsf{Rec})$ with the following properties:

- For all $w, w' \in \mathcal{M}$ with $\Delta(w, w') \leq t$ we have $\mathsf{Rec}(w', \mathsf{SS}(w)) = w$.
- For all random variables W over $\mathcal{M}$, if $\mathbf{H}_\infty(W) \geq k$ then $\mathbf{H}_\infty(W|\mathsf{SS}(W)) \geq k - \ell$.

Although the sketch $\mathsf{SS}(W)$ is guaranteed to reduce the entropy of W by at most ℓ bits, it may reveal various sensitive information about W. The work of [DS05] considered a notion of private secure sketches which required an additional property:

- For every adversary $\mathcal{A}$ there exists a simulator Sim such that for every source W over $\mathcal{M}$ with $\mathbf{H}_\infty(W) \geq k$ and every predicate φ it holds that $\Pr[\mathcal{A}(\mathsf{SS}(W)) = \varphi(W)] - \Pr[\mathsf{Sim}() = \varphi(W)] \leq \varepsilon$.

The work of [DS05] showed how to construct private secure sketches for some metrics, but only at a cost in parameters and generality. It appears to often be much simpler to construct a non-private sketch than it is to construct a private one.

Here we give a generic method to construct a computationally private secure sketch $(\mathsf{SS}', \mathsf{Rec}')$ from any non-private sketch $(\mathsf{SS}, \mathsf{Rec})$ by using an obfuscator Obf for multi-bit compute-and-compare circuits $\mathcal{P}^{\mathbf{CIRC}}_{\mathbf{MBCC}}$. Let $\mathcal{H}$ be a family of pairwise-independent hash functions. Define:

- $\mathsf{SS}'(w)$: Let $\sigma \leftarrow \mathsf{SS}(w)$, $h \leftarrow \mathcal{H}$. Define the circuit $f_{h,\sigma}(w') \stackrel{\text{def}}{=} h(\mathsf{Rec}(w', \sigma))$. Output $\sigma' \leftarrow \mathsf{Obf}(1^\lambda, \mathbf{MBCC}[f_{h,\sigma}, h(w), \sigma])$.
- $\mathsf{Rec}'(w', \sigma')$: Interpret σ' as an obfuscated program and compute $\sigma'(w')$. If the output is $\sigma \neq \perp$ then output $\mathsf{Rec}(w', \sigma)$.

Assume the original secure sketch is a (k, ℓ, t) secure sketch. We choose the output length α of the hash functions h to ensure that $k - \ell = \alpha + \omega(\log \lambda)$ and $\alpha = \lambda^{\Omega(1)}$. This gives a $(k, \ell + \alpha, t)$ secure sketch. Assume that Obf is distributional-VBB secure obfuscator for distributions $\mathcal{D}_{\alpha\text{-}\mathbf{PE}}$, which we constructed under LWE. Then the above construction is a computationally private secure sketch. In particular, by the leftover hash lemma, $h(w)$ has α bits of pseudo-entropy conditioned on σ, h and so we can rely on the distributional-VBB security of the obfuscator to argue that the obfuscated program $\mathsf{Obf}(1^\lambda, f_{h,\sigma}, h(w), \sigma)$ can be simulated and therefore does not reveal any predicate of w.

Plaintext Equality Checker. We can take any encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ and use the secret key sk to create an obfuscated *plaintext equality tester* which checks if a ciphertext is an encryption of some target plaintext y. To do so, we simple obfuscate the compute-and-compare program $\mathbf{CC}[\mathsf{Dec}_{\mathsf{sk}}(\cdot), y]$. Or, more generally, we can evaluate an arbitrary polynomial-time function g on the plaintext and test if $g(\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct})) = y$ by obfuscating $\mathbf{CC}[g \circ \mathsf{Dec}_{\mathsf{sk}}, y]$. As long as the target plaintext y has sufficient pseudo-entropy (or unpredictability), the obfuscated program can be simulated without knowing sk and therefore it does not break the semantic security of the encryption scheme. In other words, semantic security continues to hold if we add $\tilde{P} \leftarrow \mathsf{Obf}(1^\lambda, \mathbf{CC}[g \circ \mathsf{Dec}_{\mathsf{sk}}, y])$ to the public key of the scheme. This is true since the obfuscated program $\tilde{P}$ can be simulated without knowledge of sk and therefore it cannot harm semantic security.

We already implicitly used this idea of obfuscating a plaintext equality tester in our compiler from branching-program obfuscation to obfuscation for circuits/TM. In particular, when this idea is used in conjunction with fully homomorphic encryption (FHE) it is especially powerful since it allows a user to perform arbitrary operations on encrypted data and test whether the result matches the target y.

We note that securely adding a zero-test (a plaintext equality tester where the target is $y = 0$) to FHE is essentially equivalent to constructing multi-linear maps. In our case, we can only achieve security when we add a plaintext equality tester with a high pseudo-entropy target y. It would be interesting to see whether this has tool has further applications or connections to multi-linear maps.

6.1 From Attribute-Based Encryption to One-Sided Predicate Encryption

An *attribute based encryption (ABE)* scheme allows us to create secret keys sk_C corresponding to a circuit C and ciphertexts ct encrypting a message msg with respect to some attribute x. Given any such secret key sk_C and ciphertext ct it is possible to recover the message msg iff $C(x) = 1$. In fact, the adversary may see many secret keys $\mathsf{sk}_{C_1}, \ldots, \mathsf{sk}_{C_q}$ but as long as all of the circuits evaluate to $C_i(x) = 0$ the adversary will not learn anything about the encrypted message msg. The work of [GVW13] gave the first construction of ABE for all circuits under the LWE assumption. This was later improved along several dimensions by [BGG$^+$14].

A *one-sided predicate encryption (PE)* is analogous to ABE except that it aims to also hide the attribute x from users that aren't qualified to decrypt. In particular as long as the adversary only gets secret keys $\mathsf{sk}_{C_1}, \ldots, \mathsf{sk}_{C_q}$ such that $C_i(x) = 0$ the adversary will not learn anything about the attribute x or the encrypted message msg. The work of [GVW15] constructed PE for all circuits by carefully combining the ABE construction of [BGG$^+$14] with FHE. The main idea there relied on the fact that the ABE of [BGG$^+$14] already achieved some partial attribute hiding properties.

We now show how to generically upgrade any ABE to PE using obfuscation for compute-and-compare programs. The main advantage of our construction is conceptual simplicity. Another advantage of our construction is that it uses ABE generically and therefore any future advances in ABE will directly translate into analogous advances in PE. For example, in all current constructions of ABE and PE, the run-time of the encryption algorithm depends polynomially on the maximal depth of the supported circuits. There is no known analogue of "bootstrapping", which was used to overcome this obstacle in the case of FHE, even if we're willing to make a circular security assumption. Our result shows that getting rid of the dependence on circuit depth in ABE will also translate to getting rid of it in PE (at least if we're willing to make a circular security assumption.)

The main idea of our construction goes as follows. To encrypt a message msg under an attribute x, the PE encryption procedure $\mathsf{ct}' \leftarrow \mathsf{Enc}'(\mathsf{mpk}, x, \mathsf{msg})$ first runs an ABE encryption $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, x, y)$ encrypting a uniformly random value y under the attribute x. Note that ct may reveal x completely. Therefore, instead of outputting ct, the PE scheme obfuscates the multi-bit compute-and-compare program $\mathbf{MBCC}[f_{\mathsf{ct}}, y, \mathsf{msg}]$, where the circuit $f_{\mathsf{ct}}(\mathsf{sk}_C) \overset{\text{def}}{=} \mathsf{Dec}_{\mathsf{sk}_C}(\mathsf{ct})$ performs ABE decryption, and sets the obfuscated program to be the PE ciphetext. To decrypt we evaluate the PE ciphertext on the ABE secret key sk_C as an input. If an adversary only gets secret keys sk_C for circuits C on which $C(x) = 0$ then ABE security ensures that y is pseudo-random even given $f_{\mathsf{ct}}, \mathsf{msg}$ and therefore the obfuscated program can be simulated without knowledge of x, msg.

ABE and PE Definitions. A *one-sided predicate-encryption* (PE) scheme consists of algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ and domains $\mathcal{C} = \{\mathcal{C}_\lambda\}, \mathcal{M} = \{\mathcal{M}_\lambda\}, \mathcal{X} = \{\mathcal{X}_\lambda\}$ defined as follows:

- $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$ outputs a master public/secret key pair $\mathsf{mpk}, \mathsf{msk}$.
- $\mathsf{sk}_C \leftarrow \mathsf{KeyGen}(\mathsf{msk}, C)$ takes as input a circuit $C \in \mathcal{C}$ and outputs sk_C.
- $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, x, \mathsf{msg})$ encrypts a message $\mathsf{msg} \in \mathcal{M}$ with respect to an attribute $x \in \mathcal{X}$.
- $\mathsf{msg} = \mathsf{Dec}(\mathsf{sk}_C, \mathsf{ct})$ outputs msg if $C(x) = 1$.

For correctness, we require that for all $\mathsf{msg} \in \mathcal{M}, x \in \mathcal{X}, C \in \mathcal{C}$ such that $C(x) = 1$ we have

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_C, \mathsf{ct}) = \mathsf{msg} \;:\; \begin{array}{c} (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda), \\ \mathsf{sk}_C \leftarrow \mathsf{KeyGen}(\mathsf{msk}, C), \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, x, \mathsf{msg}) \end{array}\right] = 1 - \mathsf{negl}(\lambda)$$

and for all $\mathsf{msg} \in \mathcal{M}, x \in \mathcal{X}, C \in \mathcal{C}$ such that $C(x) = 0$ we have

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_C, \mathsf{ct}) = \bot \;:\; \begin{array}{c} (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda), \\ \mathsf{sk}_C \leftarrow \mathsf{KeyGen}(\mathsf{msk}, C), \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, x, \mathsf{msg}) \end{array}\right] = 1 - \mathsf{negl}(\lambda)$$

For security, we consider the following game $\mathsf{PEGame}^b_{\mathcal{A}}(1^\lambda)$ with an adversary $\mathcal{A}$ and a bit $b \in \{0, 1\}$.

- Run $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$ and give mpk to $\mathcal{A}$.
- The adversary $\mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}$ gets access to the key generation oracle and eventually outputs two tuples (x_0, msg_0), $(x_1, \mathsf{msg}_1) \in \mathcal{X} \times \mathcal{M}$.
- The challenger encrypts $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, x_b, \mathsf{msg}_b)$ and give ct to $\mathcal{A}$.
- The adversary $\mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}$ gets further access to the key generation oracle and eventually outputs a bit b' which we define as the output of the game.

An adversary $\mathcal{A}$ in the above game is legal if all of its queries C to the key generation oracle satisfy $C(x_0) = C(x_1) = 0$. We require that for all legal PPT adversaries $\mathcal{A}$ we have

$$|\Pr[\mathsf{PEGame}^0_{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathsf{PEGame}^1_{\mathcal{A}}(1^\lambda) = 1]| = \mathsf{negl}(\lambda).$$

A PE scheme is *selective secure* if the adversary has to choose $(x_0, \mathsf{msg}_0), (x_1, \mathsf{msg}_1)$ before seeing mpk or making key-generation queries.

An attribute based encryption *ABE* scheme is defined the same way as PE except that the adversary has to choose the same attribute $x_0 = x_1$. This captures the fact that we are not hiding the attribute.

Construction: ABE to PE. Let $\mathcal{E} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be an ABE scheme with some domains $\mathcal{C}, \mathcal{M}, \mathcal{X}$. Let Obf be an obfuscator for multi-bit compute-and-compare programs $\mathcal{P}^{\mathbf{CIRC}}_{\mathbf{MBCC}}$ or $\mathcal{P}^{\mathbf{TM}}_{\mathbf{MBCC}}$ which satisfies distributional indistinguishability for α-pseudo-entropy distributions $\mathcal{D}_{\alpha\text{-}\mathbf{PE}}$ for some polynomial $\alpha = \alpha(\lambda)$. We assume that $\{0,1\}^\alpha \subseteq \mathcal{M}$. For any $\mathcal{M}' = \{0,1\}^{m(\lambda)}$ we construct a PE scheme $\mathcal{E}' = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}', \mathsf{Dec}')$ which has identical $\mathsf{Setup}, \mathsf{KeyGen}$ as the ABE scheme and domains $\mathcal{C}, \mathcal{M}', \mathcal{X}$. The scheme is defined as follows.

- $\mathsf{ct}' \leftarrow \mathsf{Enc}'(\mathsf{mpk}, x, \mathsf{msg})$: Choose $y \xleftarrow{\$} \{0,1\}^\alpha$, and $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, x, y)$. Let f_{ct} be a function that has ct hard-coded, takes as input sk_C, and outputs $f_{\mathsf{ct}}(\mathsf{sk}_C) = \mathsf{Dec}(\mathsf{sk}_C, \mathsf{ct})$. Output the obfuscated program $\mathsf{ct}' \leftarrow \mathsf{Obf}(1^\lambda, \mathbf{MBCC}[f_{\mathsf{ct}}, y, \mathsf{msg}])$.
- $\mathsf{msg} = \mathsf{Dec}'(\mathsf{sk}_C, \mathsf{ct}')$: Interpret ct' as an obfuscated program and run it on input sk_C.

Theorem 6.1. *If $\mathcal{E}$ is an ABE scheme and Obf is a an obfuscator for multi-bit compute-and-compare programs (either $\mathcal{P}^{\mathbf{CIRC}}_{\mathbf{MBCC}}$ or $\mathcal{P}^{\mathbf{TM}}_{\mathbf{MBCC}}$) which satisfies distributional indistinguishability for $\mathcal{D}_{\alpha\text{-}\mathbf{PE}}$ then $\mathcal{E}'$ is a PE scheme. If $\mathcal{E}$ is selectively secure then so is $\mathcal{E}'$.*

Proof. Correctness of the PE scheme follows directly from the correctness of the ABE scheme and the obfuscation scheme.

For security, we show that the games $\mathsf{PEGame}^0_{\mathcal{A}}(1^\lambda)$ and $\mathsf{PEGame}^1_{\mathcal{A}}(1^\lambda)$ are indistinguishable via a sequence of hybrid arguments.

Hybrid 0. This is $\mathsf{PEGame}^0_{\mathcal{A}}(1^\lambda)$. Note that the challenge ciphertext ct' is computed by choosing $y \xleftarrow{\$} \{0,1\}^\alpha$, $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, x_0, y)$ and $\mathsf{ct}' \leftarrow \mathsf{Obf}(1^\lambda, \mathbf{MBCC}[f_{\mathsf{ct}}, y, \mathsf{msg}_0])$.

Hybrid 1. In this game, the challenge ciphertext ct' is computed by choosing $y \xleftarrow{\$} \{0,1\}^\alpha$, $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, x_0, 0^\alpha)$ and $\mathsf{ct}' \leftarrow \mathsf{Obf}(1^\lambda, \mathbf{MBCC}[f_{\mathsf{ct}}, y, \mathsf{msg}_0])$. In other words, we now use the ABE to encrypt the message 0^α instead of y. Hybrids 0 and 1 are indistinguishable by the security of the ABE scheme.

Hybrid 2. In this game, the challenge ciphertext ct' is computed by choosing $y \xleftarrow{\$} \{0,1\}^\alpha$, $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, x_0, 0^\alpha)$ and $\mathsf{ct}' \leftarrow \mathsf{Sim}(1^\lambda, \mathsf{params})$ where Sim is the simulator of the obfuscation scheme and params are the parameters of the program f_{ct}. In other words, we now give a simulated program instead of a correctly obfuscated one. Hybrids 1 and 2 are indistinguishable by the security of the obfuscation scheme. Note that the value y does not appear anywhere else in the game except in the obfuscated program and has α bits of real entropy even conditioned on f_{ct} and everything else the adversary sees during the game.

Hybrid 3. This game is the same as Hybrid 1 but with (x_1, msg_1) instead of (x_0, msg_0). It is indistinguishable from Hybrid 2 by the same reasoning as was used to show indistinguishability of hybrids 1 and 2.

Hybrid 4. This is $\mathsf{PEGame}^1_{\mathcal{A}}(1^\lambda)$ which is the same as Hybrid 0 but with (x_1, msg_1) instead of (x_0, msg_0). It is indistinguishable from Hybrid 3 by the same reasoning as was used to show indistinguishability of hybrids 0 and 1.

This completes the proof of security. □

Efficiency. If we start with an obfuscator for circuits then the encryption time will be at least as large as the maximal circuit size of the circuits $C \in \mathcal{C}$ supported by the ABE. However, if we start with a truly succinct obfuscator for Turing Machines then the encryption time and the size of the ciphertext in our PE scheme will only depend on the ciphertext size of the underlying ABE. If we start with a weakly succinct obfuscator for Turing Machines then the encryption time in our PE will also depend on the depth of the decryption circuit of the ABE.

This means that, under the LWE assumption, we recover the result of [GVW15] with essentially the same asymptotic efficiency. However, if in the future someone comes up with a new construction of truly succinct ABE (where the encryption time does not depend on the circuit size or depth at all), then we will be able to plug it in to our construction (assuming truly succinct FHE) and get a construction of truly succinct PE.

6.2 From Witness Encryption to Null iO

A *witness encryption (WE)* [GGSW13] scheme allows us to encrypt a message m with respect to an arbitrary NP statement x so that anybody that has a witness w for x can decrypt the message. On the other hand, if the statement x is false, then the scheme computationally hides the encrypted message.

An indistinguishability obfuscation (iO) [BGI$^+$01, GGH$^+$13b] scheme allows us to obfuscate circuits so that for any two circuits C, C' which are functionally equivalent, meaning that for all inputs x we have $C(x) = C'(x)$, the obfuscation of C is indistinguishable from that of C'. A weaker notion of iO, that we call *null iO* or *niO*, only requires security to hold for *null circuits* C, C' such that for all x we have $C(x) = C'(x) = 0$. Note that we still require the obfuscator to be correct on all circuits including ones that are not necessarily null.

It's clear that iO implies niO. Also, the work of [GGH$^+$13b] showed that iO implies WE, but it turns out that their result actually also shows that even niO implies WE. In particular, using niO, we can encrypt a message m under an NP statement x by obfuscating the circuit $C[x, m]$ which takes as input a witness w and verifies that it is a valid witness for x – if so it outputs m and else it outputs 0. If x is a false statement, then $C[x, m]$ is a null circuit and therefore we can use niO security to argue that the obfuscations of $C[x, m]$ (corresponding to an encryption of m) and $C[x, m']$ (corresponding to an encryption of m') are indistinguishable. Therefore iO implies niO implies WE.

What about the reverse directions; does WE imply niO and does niO imply iO? Previously nothing about the reverse directions was known. In this work, we use obfuscation for compute-and-compare programs to convert WE into niO. In other words, we show that under the LWE assumption WE indeed does imply niO. It remains a fascinating open problem if niO implies iO.

Definitions of Witness Encryption and Null iO. We now introduce the definitions of the two primitives we consider in this section.

Definition 6.2 (Witness Encryption)**.** *A Witness Encryption scheme for some NP language L (with corresponding witness relation R) consists of PPT algorithms* $(\mathsf{Enc}, \mathsf{Dec})$ *such that the following holds.*

Correctness: *For all $x \in L$ with witness w such that $R(x, w)$ holds and for all messages $m \in \{0,1\}^*$ we have* $\Pr[\mathsf{Dec}(w, \mathsf{Enc}(1^\lambda, x, m)) = m] = 1$

Security: *For any ensembles $x = \{x_\lambda\}$, $m = \{m_\lambda\}$, $m' = \{m'_\lambda\}$ such that for all $\lambda \in \mathbb{N}$ we have $x_\lambda \notin L$ and $|m_\lambda| = |m'_\lambda|$, we require that the following holds:*

$$\mathsf{Enc}(1^\lambda, x_\lambda, m_\lambda) \overset{c}{\approx} \mathsf{Enc}(1^\lambda, x_\lambda, m'_\lambda)$$

Note: We can assume without loss of generality that a ciphertext $\mathsf{ct} \leftarrow \mathsf{Enc}(1^\lambda, x, m)$ contains the statement x in the clear and that the decryption procedure $\mathsf{Dec}(w, \mathsf{ct})$ checks if w is a valid witness for x and if not it outputs $\perp$. In other words, this guarantees that $\mathsf{Dec}(w, \mathsf{Enc}(1^\lambda, x, m)) = m$ iff $(x, w) \in R$.

Definition 6.3 (Null iO)**.** *An null iO (niO) obfuscation scheme satisfies the following properties.*

Correctness: *There is a negligible function ν such that for all circuits $C : \{0,1\}^n \to \{0,1\}$:*

$$\Pr[\forall x \in \{0,1\}^n \ : \ C(x) = \tilde{C}(x) \mid \tilde{C} \leftarrow \mathsf{Obf}(1^\lambda, C)] \geq 1 - \nu(\lambda),$$

where the probability is over the coin tosses of Obf*.*

Security: *Let $C = \{C_\lambda\}, C' = \{C'_\lambda\}$ be two ensembles of circuits with equal input length $n(\lambda)$ and circuit size, which are furthermore* everywhere null *meaning that for all $x \in \{0,1\}^{n(\lambda)}$ we have $C_\lambda(x) = C'_\lambda(x) = 0$. Then we require that:* $\mathsf{Obf}(1^\lambda, C_\lambda) \overset{c}{\approx} \mathsf{Obf}(1^\lambda, C'_\lambda)$.

Construction: From WE to niO. We now show how to use obfuscation for compute-and-compare programs to go from WE to niO. Let $(\mathsf{Enc}, \mathsf{Dec})$ be a witness encryption scheme for the circuit satisfiability language

$$L = \{C \ : \ C \text{ is a boolean circuit } \exists x \ : \ C(x) = 1\}$$

with the natural witness relation $(C, x) \in R$ if $C(x) = 1$. Let Obf be an obfuscator for compute-and-compare circuits $\mathcal{P}^{\mathbf{CIRC}}_{\mathbf{CC}}$ which satisfies distributional indistinguishability for $\alpha(\lambda)$-pseudo-entropy distributions $\mathcal{D}_{\alpha-\mathbf{PE}}$ for some polynomial α. We build a niO obfuscator $\mathsf{Obf}'(1^\lambda, C)$ which takes as input a circuit with input size n and does the following.

- Choose a random $y \leftarrow \{0,1\}^{\alpha(\lambda)}$ and set $\mathsf{ct} \leftarrow \mathsf{Enc}(1^\lambda, C, y)$.
- Let $f_{\mathsf{ct}}(x)$ be a circuit that takes as input $x \in \{0,1\}^n$ and outputs $\mathsf{Dec}(x, \mathsf{ct})$.
- Compute $\tilde{P} \leftarrow \mathsf{Obf}(1^\lambda, \mathbf{CC}[f_{\mathsf{ct}}, y])$ and output it.

Theorem 6.4. *If* $(\mathsf{Enc}, \mathsf{Dec})$ *is a witness encryption and* Obf *is an obfuscator for $\mathcal{P}^{\mathbf{CIRC}}_{\mathbf{CC}}$ which satisfies distributional indistinguishability for distribution class $\mathcal{D}_{\alpha-\mathbf{PE}}$ for some polynomial α then* Obf' *is a secure niO scheme. In particular, under the LWE assumption, the above construction converts any witness encryption scheme into a niO scheme.*

Proof. Correctness of the niO obfuscator Obf' holds by the correctness of the WE scheme and the obfuscator Obf. In particular, with overwhelming probability over the choice of encryption/obfuscation randomness we have: $\tilde{P}(x) = 1$ iff $f_{\mathsf{ct}}(x) = y$ iff $\mathsf{Dec}(x, \mathsf{ct}) = y$ iff $C(x) = 1$.

To show security of Obf, let $C = \{C_\lambda\}$ be a circuit ensembles that is everywhere null. We first rely on the security of the WE scheme to argue that y is pseudo-random even conditioned on ct.

$$(\ (y, \mathsf{ct}) \ : \ y \leftarrow \{0,1\}^{\alpha(\lambda)}, \mathsf{ct} \leftarrow \mathsf{Enc}(1^\lambda, C_\lambda, y)) \ \overset{c}{\approx} \ (\ (y, \mathsf{ct}') \ : \ y \leftarrow \{0,1\}^{\alpha(\lambda)}, \mathsf{ct}' \leftarrow \mathsf{Enc}(1^\lambda, C_\lambda, 0^{\alpha(\lambda)})\ )$$

and therefore $\mathbf{H}_{\mathsf{HILL}}(y \mid f_{\mathsf{ct}}) \geq \mathbf{H}_\infty(y \mid f_{\mathsf{ct}'}) = \alpha(\lambda)$. We can then rely on the security of the obfuscator Obf' to argue that the obfuscated circuit can be simulated:

$$\mathsf{Obf}(1^\lambda, \mathbf{CC}[f_{\mathsf{ct}}, y]) \overset{c}{\approx} \mathsf{Sim}(1^\lambda, \mathsf{params}).$$

This shows that for any two circuits $C = \{C_\lambda\}, C' = \{C'_\lambda\}$ that are everywhere null and have the same input-size and circuit-size we have:

$$\mathsf{Obf}'(1^\lambda, C_\lambda) \overset{c}{\approx} \mathsf{Sim}(1^\lambda, \mathsf{params}) \overset{c}{\approx} \mathsf{Obf}'(1^\lambda, C'_\lambda).$$

This shows that Obf' is an niO obfuscator as we wanted. $\square$

6.3 Circular-Security Counterexamples

Definitions of encryption security, such as semantic security, are supposed to guarantee that a ciphertext hides all information about the encrypted plaintext. However, this only holds if the plaintext message is independent of the secret key. An interesting question is whether such definitions also imply security when the plaintext message can depend on the secret key. The most natural variants of this question deal with *circular security* where the plaintext is the secret key itself $\mathsf{Enc}_{\mathsf{pk}}(\mathsf{sk})$ or, more generally, *ℓ-cycle security* where the adversary sees ciphertexts $\mathsf{Enc}_{\mathsf{pk}_1}(\mathsf{sk}_2), \ldots, \mathsf{Enc}_{\mathsf{pk}_{\ell-1}}(\mathsf{sk}_\ell), \mathsf{Enc}_{\mathsf{pk}_\ell}(\mathsf{sk}_1)$. Can we guarantee that such ciphertexts look indistinguishable from the encryptions of any other key-independent plaintexts?

It's easy to see that there are encryption schemes that are semantically secure but are trivially not circular secure; for example we can take any encryption scheme and modify the encryption procedure to output the secret key in the clear if it is ever given as a plaintext. However, such trivial counter-examples go away if we only consider bit-encryption schemes where the message space consists of a single bit and the only way to encrypt a longer message is to encrypt it one bit at a time. Also, such trivial counter-examples don't exist for cycles of length $\ell \geq 2$.

Perhaps we could conjecture that every public-key *bit-encryption* scheme which is semantically secure is also circular secure? The works of [Rot13, KRW15] show counter-examples to this but only under strong non-standard assumptions (multi-linear maps or iO). Even more recently, the work of [GKW17b] provided such a counter-example for *symmetric-key* bit-encryption under the LWE assumption. In this work, we construct such a counter-example for public-key bit-encryption under the LWE assumption.

Perhaps we could instead conjecture that every semantically secure encryption scheme is ℓ-cycle secure for some sufficiently large ℓ? We know there are counter-example schemes which are not secure for $\ell = 2$-cycles [ABBC10, CGH12, BHW15] under bi-linear group assumptions and LWE. Recently, for any polynomial ℓ, we also have counter-example schemes that are not ℓ-cycle secure under iO [KRW15, MO14] and even more recently under LWE [AP16, KW16]. However, the counter-example schemes work for bounded-length cycles where the bound ℓ is fixed first and then we can create a scheme which is ℓ-cycle insecure. (Furthermore, the schemes based on DDH and LWE require common parameters to be used across all schemes.) Therefore this still leaves open the possibility that for every scheme there exists some sufficiently large polynomial ℓ for which it is ℓ-cycle secure. The latest work of [GKW17a] gives a counter-example for unbounded-length key cycles by constructing a single scheme which is ℓ-cycle insecure for all polynomial ℓ, but does so assuming iO. Here we provide such a counter-example for unbounded-length key cycles (without common parameters) under LWE.

In fact we unify the above two goals by giving a single scheme which is a semantically secure public-key bit-encryption scheme and which is neither circular secure nor ℓ-cycle secure for any ℓ. In fact, our counter-example is more dramatic than many of the previous ones since the attacker can completely recover the secret key(s) in full. Therefore, seeing an encryption of the secret key or a cycle of secret keys is not only distinguishable from random but allows an attacker to break the security of all future ciphertexts as well. Perhaps most importantly, our construction relies on a conceptually simple use of our obfuscation for compute-and-compare programs and therefore significantly simplifies and condenses the prior literature.

Construction 6.5. *Let $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be any public-key bit-encryption scheme, and let Obf be an obfuscator for multi-bit compute-and-compare circuits $\mathcal{P}^{\mathbf{CIRC}}_{\mathbf{MBCC}}$ with distributional indistinguishability for α-pseudo-entropy distributions $\mathcal{D}_{\alpha-\mathbf{PE}}$ for some polynomial $\alpha = \alpha(\lambda)$. We define the bit-encryption scheme $\mathcal{E}' = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ as follows.*

$\mathsf{Gen}'(1^\lambda)$**:** *Run* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$ *and* $y \overset{\$}{\leftarrow} \{0,1\}^\alpha$. *Construct the circuit*

$$f_{\mathsf{sk}}(\mathsf{ct}_1, \ldots, \mathsf{ct}_\alpha) \overset{def}{=} (\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}_1), \ldots, \mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}_\alpha))$$

and let $\tilde{P} \leftarrow \mathsf{Obf}(1^\lambda, \mathbf{MBCC}[f_{\mathsf{sk}}, y, \mathsf{sk}])$. *Output* $\mathsf{pk}' = (\tilde{P}, \mathsf{pk}), \mathsf{sk}' = (y, \mathsf{sk})$.

$\mathsf{Enc}'_{\mathsf{pk}'}(b)$: *Output* $\mathsf{Enc}_{\mathsf{pk}}(b)$.

$\mathsf{Dec}'_{\mathsf{sk}'}(\mathsf{ct})$: *Output* $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct})$.

Semantic Security. If $\mathcal{E}$ is a semantically secure public-key bit-encryption scheme and Obf satisfies distributional indistinguishability for the class of α-pseudo-entropy distributions $\mathcal{D}_{\alpha\text{-}\mathbf{PE}}$ then $\mathcal{E}'$ is a semantically secure public-key bit-encryption scheme. The proof of security first relies on the fact that y is uniformly random even given sk and therefore we can use the security of the obfuscation scheme to switch from a real $\tilde{P} \leftarrow \mathsf{Obf}(1^\lambda, \mathbf{MBCC}[f_{\mathsf{sk}}, y, \mathsf{sk}])$ to a simulated $\tilde{P} \leftarrow \mathsf{Sim}(1^\lambda, \mathsf{params})$ which does not depend on the secret key sk. Once we do this, the semantic security of $\mathcal{E}'$ follows directly from that of $\mathcal{E}$.

Circular Insecurity. It is clear that the above scheme $\mathcal{E}'$ is not circular secure in a very strong sense. In particular, given a ciphertext $\mathsf{ct} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_q)$ corresponding to bit-by-bit encryptions of $\mathsf{sk}' = (y, \mathsf{sk})$ we can run $\tilde{P}(\mathsf{ct}_1, \ldots, \mathsf{ct}_\alpha)$ on the first α ciphertexts (which encrypt y) to recover sk. This completely breaks security of the encryption scheme, even for future ciphertexts that don't depend on the secret key.

ℓ-Cycle Insecurity. If the scheme $\mathcal{E}$ is also a (leveled) FHE scheme, then $\mathcal{E}'$ is ℓ-cycle insecure for every polynomial ℓ. At a high level, this allows us to convert an ℓ-cycle to a 1-cycle. A similar idea was used in [GKW17a]. In particular, assume we are given public keys $\mathsf{pk}'_i = (\tilde{P}_i, \mathsf{pk}_i)$ and an encrypted key-cycle

$$\mathsf{Enc}_{\mathsf{pk}_1}(\mathsf{sk}'_2), \ldots, \mathsf{Enc}_{\mathsf{pk}_{\ell-1}}(\mathsf{sk}'_\ell), \mathsf{Enc}_{\mathsf{pk}_\ell}(\mathsf{sk}'_1)$$

where plaintexts $\mathsf{sk}'_i = (y_i, \mathsf{sk}_i)$ are encrypted bit-by-bit.

For every $i \in [\ell]$, we can use the FHE evaluation algorithm to come up with a ciphertext $\mathsf{Enc}_{\mathsf{pk}_i}(\mathsf{sk}'_i)$ encrypting $\mathsf{sk}'_i = (y_i, \mathsf{sk}_i)$ bit by bit. We can take the first α components of this ciphertext to get $\mathsf{ct}^*_i = \mathsf{Enc}_{\mathsf{pk}_i}(y_i)$ and run $\tilde{P}_i(\mathsf{ct}^*_i)$ which outputs sk_i. This allows us to completely recover all of the decryption keys.

The above supposed we have a true FHE scheme. But in fact, we can do the same thing using only a leveled FHE scheme since we can do the above computation in depth $d = \mathsf{poly}(\log \ell, \lambda)$. First, for all j in parallel, combine $\mathsf{ct}_{j \to j+1} = \mathsf{Enc}_{\mathsf{pk}_j}(\mathsf{sk}'_{j+1})$ with $\mathsf{ct}_{j+1 \to j+2} = \mathsf{Enc}_{\mathsf{pk}_{j+1}}(\mathsf{sk}'_{j+2})$ to get $\mathsf{ct}_{j \to j+2} = \mathsf{Enc}_{\mathsf{pk}_j}(\mathsf{sk}'_{j+2})$ (additions are modulo ℓ). Then combine $\mathsf{ct}_{j \to j+2}$ with $\mathsf{ct}_{j \to j+4}$ to get $\mathsf{ct}_{j \to j+4} = \mathsf{Enc}_{\mathsf{pk}_j}(\mathsf{sk}'_{j+4})$. By continuing this process for $\log \ell$ steps we can get $\mathsf{ct}_{i \to i+\ell} = \mathsf{Enc}_{\mathsf{pk}_i}(\mathsf{sk}'_i)$ as we wanted. *(This works directly if ℓ is a power of 2 but it is easy to extend to any ℓ. If $\ell = \sum_{k=0}^{\lfloor \log \ell \rfloor} b_k 2^k$ then, after we compute ciphertexts $\mathsf{ct}_{j \to j+2^k}$ for all j, k in the first $\lfloor \log \ell \rfloor$ steps, we can spend another $\lfloor \log \ell \rfloor$ steps to iteratively compute $\mathsf{ct}_{i \to i + \sum_{k=0}^{q} b_k 2^k}$ for $q = 1, \ldots, \lfloor \log \ell \rfloor$ by combining the appropriate 2^k "jumps" that we computed previously.)* Therefore, by upper bounding $\log \ell < \lambda$, the computation's depth is upper bounded by some fixed polynomial $d = d(\lambda)$ in the security parameter, which does not depend on ℓ, and allows us to do the computation for all polynomial ℓ. Therefore, by using this fixed polynomial d for the depth of the leveled FHE, we get a single scheme which can handle all polynomials ℓ.

Summarizing, we get the following theorem.

Theorem 6.6. *Under the LWE assumption there exists a public-key bit-encryption scheme which is semantically secure but is neither circular secure nor ℓ-cycle secure for any polynomial ℓ.*

6.3.1 Circular (In)secure Compiler[5]

The work of Black, Rogaway, and Shrimpton [BRS03] showed how to transform any semantically secure scheme into a Key-Dependent Message (KDM) secure one in the random-oracle model. A scheme is defined to be KDM secure, if it remains secure even if an adversary has access to encryptions of messages

[5]This result was added subsequently to the otherwise concurrent/independent work of [GKW17c].

that depend arbitrary on the secret key. Clearly, KDM security implies circular security. We show that under the LWE assumption, there exists a semantically secure scheme for which the transformation of [BRS03] *fails* to be circular secure in the standard model, when the random oracle is replaced with *any* hash function family.

The Compiler [BRS03]. Given a public-key encryption scheme $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ and a hash function H, the transformation of [BRS03] defines a new public-key scheme $\mathcal{E}^* = (\mathsf{Gen}^*, \mathsf{Enc}^*, \mathsf{Dec}^*)$ as follows.

- $\mathsf{Gen}^*(1^\lambda)$: Run $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$. Output $\mathsf{pk}^* = \mathsf{pk}$, $\mathsf{sk}^* = \mathsf{sk}$.
- $\mathsf{Enc}^*_{\mathsf{pk}^*}(m)$: Sample $r \overset{\$}{\leftarrow} \{0,1\}^\lambda$. Output $(\mathsf{ct}_1, \mathsf{ct}_2) = (\mathsf{Enc}_{\mathsf{pk}}(r), H(r) \oplus m)$.
- $\mathsf{Dec}^*_{\mathsf{sk}^*}(\mathsf{ct}_1, \mathsf{ct}_2)$: Output $H(\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}_1)) \oplus \mathsf{ct}_2$.

The work of [BRS03] proved that when $H(\cdot)$ is modeled as a random oracle and $\mathcal{E}$ is semantically secure then $\mathcal{E}^*$ is KDM secure. Intuitively, the only way to learn anything from ct_2 is to call the random oracle on r but r is hidden by the semantic security of $\mathcal{E}$.

Circular Insecurity in the Standard Model. As our counter-example we take the public-key bit encryption scheme $\mathcal{E}'$ from Construction 6.5, where the underlying scheme $\mathcal{E}$ used by $\mathcal{E}'$ is a (leveled) FHE scheme. We note that $\mathcal{E}'$ is itself an FHE scheme which we proved to be semantically secure but it is circular insecure in a very strong sense: given *any* ciphertext which decrypts to the secret key we can recover the secret key.

If we apply the compiler of [BRS03] to the scheme $\mathcal{E}'$ and let H by any standard-model hash function (or family of hash functions) then we claim that the resulting scheme $\mathcal{E}^*$ fails to be circular secure. In particular if $(\mathsf{ct}_1, \mathsf{ct}_2) = \mathsf{Enc}^*_{\mathsf{pk}^*}(\mathsf{sk}^*)$ then $\mathsf{ct}_1 = \mathsf{Enc}'_{\mathsf{pk}'}(r)$ and $\mathsf{ct}_2 = H(r) \oplus \mathsf{sk}'$. Since the encryption scheme $\mathcal{E}'$ is an FHE scheme, we can homomorphically evaluate the function $f_{\mathsf{ct}_2}(r) = H(r) \oplus \mathsf{ct}_2$ on the ciphertext ct_1 to get an encryption of sk' under pk'. But by the way that $\mathcal{E}'$ was constructed, any such encryption allows us to recover sk'.

References

[ABBC10] Tolga Acar, Mira Belenkiy, Mihir Bellare, and David Cash. Cryptographic agility and its relation to circular encryption. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 403–422, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany.

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 595–618, Santa Barbara, CA, USA, August 16–20, 2009. Springer, Heidelberg, Germany.

[Ajt99] Miklós Ajtai. Generating hard instances of the short basis problem. In *ICALP*, pages 1–9, 1999.

[AKPW13] Joël Alwen, Stephan Krenn, Krzysztof Pietrzak, and Daniel Wichs. Learning with rounding, revisited - new reduction, properties and applications. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 57–74, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.

[AP16] Navid Alamati and Chris Peikert. Three's compromised too: Circular insecurity for any cycle length from (ring-)LWE. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 659–680, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

[Bar89] David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc^1. *J. Comput. Syst. Sci.*, 38(1):150–164, 1989.

[BBC$^+$14] Boaz Barak, Nir Bitansky, Ran Canetti, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Obfuscation for evasive functions. In Yehuda Lindell, editor, *TCC 2014: 11th Theory of Cryptography Conference*, volume 8349 of *Lecture Notes in Computer Science*, pages 26–51, San Diego, CA, USA, February 24–26, 2014. Springer, Heidelberg, Germany.

[BGG$^+$14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 533–556, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany.

[BGI$^+$01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012: 3rd Innovations in Theoretical Computer Science*, pages 309–325, Cambridge, MA, USA, January 8–10, 2012. Association for Computing Machinery.

[BHW15] Allison Bishop, Susan Hohenberger, and Brent Waters. New circular security counterexamples from decision linear and learning with errors. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 776–800, Auckland, New Zealand, November 30 – December 3, 2015. Springer, Heidelberg, Germany.

[BLP$^+$13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 575–584, Palo Alto, CA, USA, June 1–4, 2013. ACM Press.

[BPR12] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 719–737, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.

[BR13] Zvika Brakerski and Guy N. Rothblum. Obfuscating conjunctions. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 416–434, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.

[BRS03] John Black, Phillip Rogaway, and Thomas Shrimpton. Encryption-scheme security in the presence of key-dependent messages. In Kaisa Nyberg and Howard M. Heys, editors, *SAC 2002: 9th Annual International Workshop on Selected Areas in Cryptography*, volume 2595 of *Lecture Notes in Computer Science*, pages 62–75, St. John's, Newfoundland, Canada, August 15–16, 2003. Springer, Heidelberg, Germany.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *52nd Annual Symposium on Foundations of Computer Science*, pages 97–106, Palm Springs, CA, USA, October 22–25, 2011. IEEE Computer Society Press.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based FHE as secure as PKE. In Moni Naor, editor, *ITCS 2014: 5th Innovations in Theoretical Computer Science*, pages 1–12, Princeton, NJ, USA, January 12–14, 2014. Association for Computing Machinery.

[BVWW16] Zvika Brakerski, Vinod Vaikuntanathan, Hoeteck Wee, and Daniel Wichs. Obfuscating conjunctions under entropic ring LWE. In Madhu Sudan, editor, *ITCS 2016: 7th Innovations in Theoretical Computer Science*, pages 147–156, Cambridge, MA, USA, January 14–16, 2016. Association for Computing Machinery.

[Can97] Ran Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In Burton S. Kaliski Jr., editor, *Advances in Cryptology – CRYPTO'97*, volume 1294 of *Lecture Notes in Computer Science*, pages 455–469, Santa Barbara, CA, USA, August 17–21, 1997. Springer, Heidelberg, Germany.

[CC17] Ran Canetti and Yilei Chen. Constraint-hiding constrained PRFs for NC^1 from LWE. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 446–476, Paris, France, May 8–12, 2017. Springer, Heidelberg, Germany.

[CD08] Ran Canetti and Ronny Ramzi Dakdouk. Obfuscating point functions with multibit output. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 489–508, Istanbul, Turkey, April 13–17, 2008. Springer, Heidelberg, Germany.

[CGH12] David Cash, Matthew Green, and Susan Hohenberger. New definitions and separations for circular security. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012: 15th International Conference on Theory and Practice of Public Key Cryptography*, volume 7293 of *Lecture Notes in Computer Science*, pages 540–557, Darmstadt, Germany, May 21–23, 2012. Springer, Heidelberg, Germany.

[CLLT16] Jean-Sébastien Coron, Moon Sung Lee, Tancrède Lepoint, and Mehdi Tibouchi. Cryptanalysis of GGH15 multilinear maps. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 607–628, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

[CRV10] Ran Canetti, Guy N. Rothblum, and Mayank Varia. Obfuscation of hyperplane membership. In Daniele Micciancio, editor, *TCC 2010: 7th Theory of Cryptography Conference*, volume 5978 of *Lecture Notes in Computer Science*, pages 72–89, Zurich, Switzerland, February 9–11, 2010. Springer, Heidelberg, Germany.

[DORS08] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam D. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.*, 38(1):97–139, 2008.

[DS05] Yevgeniy Dodis and Adam Smith. Correcting errors without leaking partial information. In Harold N. Gabow and Ronald Fagin, editors, *37th Annual ACM Symposium on Theory of Computing*, pages 654–663, Baltimore, MA, USA, May 22–24, 2005. ACM Press.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 1–17, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany.

[GGH$^+$13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual Symposium on Foundations of Computer Science*, pages 40–49, Berkeley, CA, USA, October 26–29, 2013. IEEE Computer Society Press.

[GGH15] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015: 12th Theory of Cryptography Conference, Part II*, volume 9015 of *Lecture Notes in Computer Science*, pages 498–527, Warsaw, Poland, March 23–25, 2015. Springer, Heidelberg, Germany.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 467–476, Palo Alto, CA, USA, June 1–4, 2013. ACM Press.

[GKPV10] Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. Robustness of the learning with errors assumption. In Andrew Chi-Chih Yao, editor, *ICS 2010: 1st Innovations in Computer Science*, pages 230–240, Tsinghua University, Beijing, China, January 5–7, 2010. Tsinghua University Press.

[GKW17a] Rishab Goyal, Venkata Koppula, and Brent Waters. Separating IND-CPA and circular security for unbounded length key cycles. In Serge Fehr, editor, *PKC 2017: 20th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 10174 of *Lecture Notes in Computer Science*, pages 232–246, Amsterdam, The Netherlands, March 28–31, 2017. Springer, Heidelberg, Germany.

[GKW17b] Rishab Goyal, Venkata Koppula, and Brent Waters. Separating semantic and circular security for symmetric-key bit encryption from the learning with errors assumption. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 528–557, Paris, France, May 8–12, 2017. Springer, Heidelberg, Germany.

[GKW17c] Rishab Goyal, Venkata Koppula, and Brent Waters. lockable obfuscation. Cryptology ePrint Archive, Report 2017/274, 2017. `http://eprint.iacr.org/2017/274`.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206, 2008.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of

Lecture Notes in Computer Science, pages 75–92, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.

[GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 545–554, Palo Alto, CA, USA, June 1–4, 2013. ACM Press.

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from LWE. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 503–523, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

[Had00] Satoshi Hada. Zero-knowledge and code obfuscation. In Tatsuaki Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 443–457, Kyoto, Japan, December 3–7, 2000. Springer, Heidelberg, Germany.

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

[HLR07] Chun-Yuan Hsiao, Chi-Jen Lu, and Leonid Reyzin. Conditional computational entropy, or toward separating pseudoentropy from compressibility. In Moni Naor, editor, *Advances in Cryptology – EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 169–186, Barcelona, Spain, May 20–24, 2007. Springer, Heidelberg, Germany.

[ILL89] Russell Impagliazzo, Leonid A. Levin, and Michael Luby. Pseudo-random generation from one-way functions (extended abstracts). In *21st Annual ACM Symposium on Theory of Computing*, pages 12–24, Seattle, WA, USA, May 15–17, 1989. ACM Press.

[KRW15] Venkata Koppula, Kim Ramchen, and Brent Waters. Separations in circular security for arbitrary length key cycles. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015: 12th Theory of Cryptography Conference, Part II*, volume 9015 of *Lecture Notes in Computer Science*, pages 378–400, Warsaw, Poland, March 23–25, 2015. Springer, Heidelberg, Germany.

[KW16] Venkata Koppula and Brent Waters. Circular security separations for arbitrary length cycles from LWE. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 681–700, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

[MO14] Antonio Marcedone and Claudio Orlandi. Obfuscation $\rightarrow$ (IND-CPA security $\not\rightarrow$ circular security). In Michel Abdalla and Roberto De Prisco, editors, *SCN 14: 9th International Conference on Security in Communication Networks*, volume 8642 of *Lecture Notes in Computer Science*, pages 77–90, Amalfi, Italy, September 3–5, 2014. Springer, Heidelberg, Germany.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT*, pages 700–718, 2012.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In Michael Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 333–342, Bethesda, MD, USA, May 31 – June 2, 2009. ACM Press.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th Annual ACM Symposium on Theory of Computing*, pages 84–93, Baltimore, MA, USA, May 22–24, 2005. ACM Press.

[Rot13] Ron Rothblum. On the circular security of bit-encryption. In Amit Sahai, editor, *TCC 2013: 10th Theory of Cryptography Conference*, volume 7785 of *Lecture Notes in Computer Science*, pages 579–598, Tokyo, Japan, March 3–6, 2013. Springer, Heidelberg, Germany.

[Wee05] Hoeteck Wee. On obfuscating point functions. In Harold N. Gabow and Ronald Fagin, editors, *37th Annual ACM Symposium on Theory of Computing*, pages 523–532, Baltimore, MA, USA, May 22–24, 2005. ACM Press.

[Zha16] Mark Zhandry. The magic of ELFs. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 479–508, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

A A Construction from (null) iO

Technically speaking, D-VBB obfuscation for compute-and-compare programs is incomparable to iO. However, here we show a very simple construction of the former from the latter. In fact, we only need null iO rather than full iO.

We sketch the construction of an obfuscator Obf for compute-and-compare programs $\mathcal{P}_{\mathbf{CC}}^{\mathbf{CIRC}}$ that satisfies distributional indistinguishability for the class of distributions $D = \{D_\lambda\}$ for which y is sampled uniformly at random $y \xleftarrow{\$} \{0,1\}^\lambda$ and independently of f, aux. Our construction uses an indistinguishability obfuscator iO and an injective PRG G for uniform seeds. If y has some small amount of pseudo-entropy λ^ε (for any $\varepsilon > 0$) or is computationally unpredictable given (f, aux), we can upgrade security using appropriate PRGs as explained in Sections 5.1, 5.3.

To obfuscate a program $\mathbf{CC}[f, y]$ our compute-and-compare obfuscator simply uses an iO scheme to obfuscate the program $\mathbf{CC}[G \circ f, G(y)]$. Correctness follows from the correctness of iO and the fact that G is injective. To argue security, consider some distribution $(\mathbf{CC}[f, y], \mathsf{aux}) \leftarrow D_\lambda$ where y is random and independent of f, aux. We first replace $G(y)$ with a uniformly random string y'. By the security of G we have $(\mathsf{iO}(\mathbf{CC}[G \circ f, G(y)]), \mathsf{aux}) \overset{c}{\approx} (\mathsf{iO}(\mathbf{CC}[G \circ f, y']), \mathsf{aux})$. Second, y' lies outside the range of G with high probability over its choice. Therefore, there is no input x such that $\mathbf{CC}[G \circ f, y'](x) = 1$, and thus we can replace $\mathbf{CC}[G \circ f, y']$ with the (appropriately padded) constant function $g(x) = 0$. By the security of the iO obfuscator it holds that $(\mathsf{iO}(\mathbf{CC}[G \circ f, y']), \mathsf{aux}) \overset{c}{\approx} (\mathsf{iO}(g), \mathsf{aux})$. Notice that for this construction, a null iO obfuscator is sufficient.