58th Annual IEEE Symposium on Foundations of Computer Science

Obfuscating Compute-and-Compare Programs under LWE

Daniel Wichs
Northeastern University
wichs@ccs.neu.edu

Giorgos Zirdelis
Northeastern University
zirdelis.g@husky.neu.edu

***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.

I. 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] 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.

A. 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

0272-5428/17 $31.00 © 2017 IEEE
DOI 10.1109/FOCS.2017.61

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

B. 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 $\mathbf{Ax} \stackrel{?}{=} \mathbf{y}$, 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.

C. 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.

D. 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 $\{\mathbf{A}_1 \to \mathbf{A}'_1\ ,\ \ldots\ ,\ \mathbf{A}_w \to \mathbf{A}'_w\}$ simultaneously by sampling a matrix $\mathbf{C} \in \mathbb{Z}_q^{m\times m}$ such that

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

This can be done the same way as in the single-edge case given the trapdoor for the matrix $\mathbf{B} = \begin{bmatrix}\mathbf{A}_1\\ \ldots\\ \mathbf{A}_w\end{bmatrix}$ 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 I-D 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]

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: $\mathbf{D} := \mathbf{A}_{1,0} \cdot \left(\prod_{i=1}^{L} \mathbf{C}_{i,x_{(i \bmod \ell_{in})}}\right)$ satisfying $\mathbf{D} \approx \left(\prod_{i=1}^{L} \mathbf{S}_{i,x_{(i \bmod \ell_{in})}}\right) \cdot \mathbf{A}_{L+1,P(x)}$. Note that this "evaluation" does not allow us to recover the output $P(x)$,

[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}.

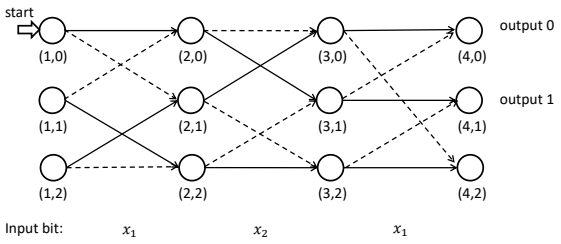

Figure 1. *Example of a branching program of length $L = 3$, width $w = 3$, and input size $\ell_{in} = 2$. Solid edges are labeled with 1 and the dashed edges with 0. For example, on input $x = 10$ (i.e., $x_1 = 1, x_2 = 0$) the program evaluates to 0. (Technically the above simple example is not a legal branching program since there are inputs on which it does no evaluate to 0 or 1, but it is useful to illustrate the concept.)*

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 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.

II. Organization

Due to space limitations, this proceedings version of our paper only presents our "basic obfuscator" for compute-and-compare programs $\mathbf{CC}[f, y]$ where f is a polynomial-length branching program and y has very high pseudo-entropy exceeding some polynomial threshold $\alpha(\lambda, L)$ in the security parameter λ and the branching program length L. In the full version of the paper [WZ17] we show how to reduce the requirements on the pseudo-entropy of y to only exceed λ^ε for an arbitrary $\varepsilon > 0$ or (under additional assumptions) to only require that y is computationally unpredictable. Furthermore, we allow f to be an arbitrary circuit or Turing Machine. Lastly, in the full version we present several applications of obfuscation for compute and compare programs as described in the introduction.

III. Notation and Preliminaries

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 1 ([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* $\mathbf{BC} = \mathbf{B}'$.

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 $(\mathbf{B}, \mathsf{td}) \leftarrow \mathsf{TrapGen}(1^k, 1^m, q)$, $\mathbf{C} \leftarrow \mathsf{Sam}(1^m, q)$, $\mathbf{B}' \xleftarrow{\$} \mathbb{Z}_q^{k \times m}$, $\mathbf{C}' \leftarrow \mathsf{SamPre}(\mathbf{B}, \mathbf{B}', \mathsf{td})$. *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 1 ([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*

$$(\mathbf{A}, \mathbf{sA} + \mathbf{e}) \overset{\text{c}}{\approx} (\mathbf{A}, \mathbf{u})$$

where $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}, \mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^n, \mathbf{e} \leftarrow \chi^m, \mathbf{u} \xleftarrow{\$} \mathbb{Z}_q^m$.

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:

$$(\mathbf{A}, \mathbf{sA} + \mathbf{e}) \overset{\text{c}}{\approx} (\mathbf{A}, \mathbf{u})$$

where $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}, \mathbf{s} \xleftarrow{\$} \chi^n, \mathbf{e} \leftarrow \chi^m, \mathbf{u} \xleftarrow{\$} \mathbb{Z}_q^m$. Via a simple hybrid argument, we also get security when $\mathbf{S}$ is a matrix rather than a vector:

$$(\mathbf{A}, \mathbf{SA} + \mathbf{E}) \overset{\text{c}}{\approx} (\mathbf{A}, \mathbf{U}) \tag{1}$$

where$\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}, \mathbf{S} \xleftarrow{\$} \chi^{n \times n}, \mathbf{E} \leftarrow \chi^{n \times m}, \mathbf{U} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$. 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.

IV. OBFUSCATION DEFINITIONS

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 $\tilde{P} \leftarrow \mathsf{Obf}(1^\lambda, P)$. An obfuscator is required to be functionality preserving, meaning there is some negligible function $\nu(\lambda)$ such that for all programs $P \in \mathcal{P}$ with input size n we have

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

where the probability is over the choice of $\tilde{P} \leftarrow \mathsf{Obf}(1^\lambda, P)$.

In the full version, we give two definitions of security called *distributional VBB* and *distributional indistinguishability* and show that they are equivalent. Here we only define the latter notion.

Definition 2 (Distributional Indistinguishability)**.** *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 $P \in \mathcal{P}$. 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*

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

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

A. Defining Compute-and-Compare Obfuscation

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 V-C 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.

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).

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.

V. 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 very 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 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 $\mathcal{P}^{\mathbf{BP}}_{\mathbf{CC}}$ which satisfies distributional indistinguishability for the class of α-pseudo-entropy distributions $\mathcal{D}_{\alpha\text{-}\mathbf{PE}}$.*

A. 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 1, 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.

B. Directed Encodings

Definition 3 (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 $\mathcal{E}^{\beta}_{\mathbf{A}_i \to \mathbf{A}_j}(\mathbf{S})$ as*

$$\{\mathbf{C} \ : \ \mathbf{A}_i\mathbf{C} = \mathbf{S}\mathbf{A}_j + \mathbf{E} \ \text{ and } \ ||\mathbf{S}||_\infty, ||\mathbf{C}||_\infty, ||\mathbf{E}||_\infty \leq \beta\}.$$

It's easy to see that the above definition implies that for any $\mathbf{C}_1 \in \mathcal{E}^{\beta_1}_{\mathbf{A}_1 \to \mathbf{A}_2}(\mathbf{S}_1)$, $\mathbf{C}_2 \in \mathcal{E}^{\beta_2}_{\mathbf{A}_2 \to \mathbf{A}_3}(\mathbf{S}_2)$ we have $\mathbf{C}_1\mathbf{C}_2 \in \mathcal{E}^{2m\beta_1\beta_2}_{\mathbf{A}_1 \to \mathbf{A}_3}(\mathbf{S}_1\mathbf{S}_2)$.

In particular, we get the following claim:

Claim 1. *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 $\{\mathbf{A}_0 \to \mathbf{A}'_0 \ , \ \ldots \ , \ \mathbf{A}_{w-1} \to \mathbf{A}'_{w-1}\}$ simultaneously. 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 1. We parse $\mathbf{B} \in \mathbb{Z}_q^{w \cdot n \times m}$ as
$$\mathbf{B} = \begin{bmatrix} \mathbf{A}_0 \\ \ldots \\ \mathbf{A}_{w-1} \end{bmatrix}$$
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
$$\mathbf{B} = \begin{bmatrix} \mathbf{A}_0 \\ \ldots \\ \mathbf{A}_{w-1} \end{bmatrix}, \mathbf{B}' = \begin{bmatrix} \mathbf{A}'_0 \\ \ldots \\ \mathbf{A}'_{w-1} \end{bmatrix}.$$
Set
$$\mathbf{H} := (\mathbf{I}_w \otimes \mathbf{S}) \cdot \mathbf{B}' + \mathbf{E} = \begin{bmatrix} \mathbf{S} \cdot \mathbf{A}'_0 + \mathbf{E}_0 \\ \ldots \\ \mathbf{S} \cdot \mathbf{A}'_{w-1} + \mathbf{E}_{w-1} \end{bmatrix}$$
where $\mathbf{E} = \begin{bmatrix} \mathbf{E}_0 \\ \ldots \\ \mathbf{E}_{w-1} \end{bmatrix} \leftarrow \chi^{w \cdot n \times m}$. Output $\mathbf{C} \leftarrow \mathsf{SamPre}(\mathbf{B}, \mathbf{H}, \mathsf{td}_{\mathbf{B}})$ where the SamPre algorithm is defined in Lemma 1.

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 2 (Correctness)**.** *For every* $\mathbf{S}$ *with* $||\mathbf{S}||_\infty \leq \beta$*, 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:*

$$\forall j \in \{0, \ldots, w-1\} \ : \ \mathbf{C} \in \mathcal{E}^{\beta}_{\mathbf{A}_j \to \mathbf{A}'_j}(\mathbf{S})$$

where $\mathbf{B} = \begin{bmatrix} \mathbf{A}_0 \\ \ldots \\ \mathbf{A}_{w-1} \end{bmatrix}, \mathbf{B}' = \begin{bmatrix} \mathbf{A}'_0 \\ \ldots \\ \mathbf{A}'_{w-1} \end{bmatrix}$.

Claim 3 (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:*

$$(\mathbf{B}_k, \mathbf{B}'_k, \mathbf{C}_k, \mathsf{td}_{\mathbf{B}_k})_{k \in [\ell]} \overset{c}{\approx} (\mathbf{B}_k, \mathbf{B}'_k, \mathbf{C}'_k, \mathsf{td}_{\mathbf{B}_k})_{k \in [\ell]} \quad (2)$$

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

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

The proofs of these claims appear in the full version of this paper [WZ17].

C. 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 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 $P(x)$.

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 $\mathbf{B}_i^{(k)} = \begin{bmatrix} \mathbf{A}_{i,0}^{(k)} \\ \ldots \\ \mathbf{A}_{i,w-1}^{(k)} \end{bmatrix}$.
- For $k \in [\ell_{out}]$ sample the matrix $\mathbf{B}_{L+1}^{(k)} = \begin{bmatrix} \mathbf{A}_{L+1,0}^{(k)} \\ \ldots \\ \mathbf{A}_{L+1,w-1}^{(k)} \end{bmatrix}$ by setting $\mathbf{A}_{L+1,0}^{(k)} := \mathbf{A}_0^{(k)}, \mathbf{A}_{L+1,1}^{(k)} := \mathbf{A}_1^{(k)}$ and sampling $\mathbf{A}_{L+1,j}^{(k)} \overset{\$}{\leftarrow} \mathbb{Z}_q^{n \times m}$ for $j \in \{2, \ldots, w-1\}$.
- 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: $\mathbf{C}_{i,b}^{(k)} \leftarrow \mathsf{DE.Encode}(\mathbf{B}_i^{(k)} \to \pi_{i,b}^{(k)}(\mathbf{B}_{i+1}^{(k)}), \mathbf{S}_{i,b}, \mathsf{td}_{\mathbf{B}_i^{(k)}})$, where

[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.

(abusing notation) we define $\pi(\mathbf{B}_i^{(k)}) = \begin{bmatrix} \mathbf{A}_{i,\pi(0)}^{(k)} \\ \cdots \\ \mathbf{A}_{i,\pi(w-1)}^{(k)} \end{bmatrix}$.

– Finally, output the sequence $\widehat{P} = \left(\mathbf{A}_{1,0}^{(k)}, \left(\mathbf{C}_{i,b}^{(k)}\right)_{i\in[L],b\in\{0,1\}}\right)_{k\in[\ell_{out}]}$.

$\mathsf{BP.Eval}(\widehat{P}, x)$. To evaluate $\widehat{P}$ on input $x \in \{0,1\}^{\ell_{in}}$, the evaluation algorithm for all $k \in [\ell]$ computes

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

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 $\mathbf{D}^{(k)} \approx \mathbf{S}^* \mathbf{A}_{P^{(k)}(x)}^{(k)}$ 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 $\pi_{i,b}^{(k)}$) being encoded.

Claim 4 (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*

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

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

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

Claim 5 (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*

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

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

The proofs of the above claims appear in the full version of this paper [WZ17].

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 $\sum_{k=1}^{\ell_{out}} \mathbf{D}^{(k)} \approx \mathbf{0}$.

Our construction of an obfuscator $\mathsf{Obf}(1^\lambda, \mathbf{CC}[f, y])$ for compute-and-compare branching programs is defined as follows. 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 6 (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*

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

where the probability is over $\tilde{P} \leftarrow \mathsf{Obf}(1^\lambda, \mathbf{CC}[f, y])$.

Claim 7 (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 $\mathcal{P}_{\mathbf{CC}}^{\mathbf{BP}}$ 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 $(\mathbf{CC}[f, y], \mathsf{aux}) \leftarrow D_\lambda$.

The proofs of the above claims appear in the full version of this paper [WZ17]. The combination of Claim 6 and Claim 7 proves Theorem 1.

REFERENCES

[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.

[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.

[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.

[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.

[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.

[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.

[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.

[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.

[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.

[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.

[WZ17] Daniel Wichs and Giorgos Zirdelis. Obfuscating compute-and-compare programs under lwe. Cryptology ePrint Archive, Report 2017/276, 2017. http://eprint.iacr.org/2017/276.

[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.