Candidate Indistinguishability Obfuscation and Functional Encryption for all circuits

Sanjam Garg
UCLA
sanjamg@cs.ucla.edu

Craig Gentry
IBM Research
craigbgentry@gmail.com

Shai Halevi
IBM Research
shaih@alum.mit.edu

Mariana Raykova
IBM Research
mariana@cs.columbia.edu

Amit Sahai
UCLA
sahai@cs.ucla.edu

Brent Waters
University of Texas at Austin
bwaters@cs.utexas.edu

July 21, 2013

Abstract

In this work, we study *indistinguishability obfuscation* and *functional encryption* for general circuits:

Indistinguishability obfuscation requires that given any two equivalent circuits C_0 and C_1 of similar size, the obfuscations of C_0 and C_1 should be computationally indistinguishable.

In functional encryption, ciphertexts encrypt inputs x and keys are issued for circuits C. Using the key SK_C to decrypt a ciphertext $\mathrm{CT}_x = \mathsf{Enc}(x)$, yields the value $C(x)$ but does not reveal anything else about x. Furthermore, no collusion of secret key holders should be able to learn anything more than the union of what they can each learn individually.

We give constructions for indistinguishability obfuscation and functional encryption that supports all polynomial-size circuits. We accomplish this goal in three steps:

- We describe a candidate construction for indistinguishability obfuscation for $\mathbf{NC}^1$ circuits. The security of this construction is based on a new algebraic hardness assumption. The candidate and assumption use a simplified variant of multilinear maps, which we call *Multilinear Jigsaw Puzzles.*
- We show how to use indistinguishability obfuscation for $\mathbf{NC}^1$ together with Fully Homomorphic Encryption (with decryption in $\mathbf{NC}^1$) to achieve indistinguishability obfuscation for all circuits.
- Finally, we show how to use indistinguishability obfuscation for circuits, public-key encryption, and non-interactive zero knowledge to achieve functional encryption for all circuits. The functional encryption scheme we construct also enjoys succinct ciphertexts, which enables several other applications.

The first and fifth authors were supported in part from NSF grants 1228984, 1136174, 1118096, 1065276, 0916574 and 0830803, a Xerox Faculty Research Award, a Google Faculty Research Award, an equipment grant from Intel, and an Okawa Foundation Research Grant. The views expressed are those of the author and do not reflect the official policy or position of the National Science Foundation, or the U.S. Government.

The second and third authors were supported by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior National Business Center (DoI/NBC) contract number D11PC20202. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoI/NBC, or the U.S. Government.

The fourth author is supported by NSF Grant No.1017660.

The sixth author is supported by NSF CNS-0915361 and CNS-0952692, CNS-1228599, DARPA N11AP20006, Google Faculty Research award, the Alfred P. Sloan Fellowship, Microsoft Faculty Fellowship, and Packard Foundation Fellowship.

Contents

1 Introduction

In this work we study two long-standing open feasibility questions in cryptography: secure program obfuscation, and functional encryption.

Obfuscation. Roughly speaking, program obfuscation aims to make a computer program "unintelligible" while preserving its functionality. The formal study of program obfuscation was initiated by Barak, Goldreich, Impagliazzo, Rudich, Sahai, Vadhan, and Yang [BGI$^+$01, BGI$^+$12]. Unfortunately, they showed that the most natural simulation-based formulation of program obfuscation (a.k.a. "black-box obfuscation") is impossible to achieve for general programs, in a very strong sense: They showed that there exist *unobfuscatable* functions – a family of functions $\{f_s\}$ such that given *any* circuit that implements f_s, an efficient procedure can extract the secret s; however, any efficient adversary given only black-box access to f_s cannot guess even a single bit of s with non-negligible advantage. Indeed, such unobfuscatable function families exist with very low circuit complexity (i.e. in $\mathbf{TC^0}$ under widely believed intractability assumptions), so one cannot hope for general purpose obfuscators achieving a natural simulation-based definition of obfuscation even for very low complexity classes.

Faced with this impossibility result, Barak et al. [BGI$^+$01, BGI$^+$12] suggested another notion of program obfuscation called *indistinguishability obfuscation*: An indistinguishability obfuscator $i\mathcal{O}$ for a class of circuits $\mathcal{C}$ guarantees that given two *equivalent* circuits C_1 and C_2 from the class, the two distribution of obfuscations $i\mathcal{O}(C_1)$ and $i\mathcal{O}(C_2)$ should be computationally indistinguishable. We note that if the circuit class $\mathcal{C}$ has efficiently computable canonical forms, then the computation of that canonical form would already be an indistinguishability obfuscator [BGI$^+$12, GR07]. Since their introduction in 2001, it has been an open problem to construct indistinguishability obfuscators for any natural circuit class that is not known to admit efficiently computable canonical forms. In particular, the central open question regarding indistinguishability obfuscation has been:

Do there exist indistinguishability obfuscators for all polynomial-size circuits?

It is important to note that unlike simulation-based definitions of obfuscation, it is not immediately clear how useful indistinguishability obfuscators would be. Perhaps the strongest philosophical justification for indistinguishability obfuscators comes from the work of Goldwasser and Rothblum [GR07], who showed that (efficiently computable) indistinguishability obfuscators achieve the notion of *Best-Possible Obfuscation* [GR07]: Informally, a best-possible obfuscator guarantees that its output hides as much about the input circuit as any circuit of a certain size.

Nevertheless, very few applications of indistinguishability obfuscators were described in the literature, in fact the only application that we are aware of is to removing software watermarks [BGI$^+$12]. The main contributions of this work are to **(1)** construct indistinguishability obfuscators for all circuits, and **(2)** show how to use indistinguishability obfuscators to solve the central open problem in the area of Functional Encryption.

Functional Encryption. In Functional Encryption, ciphertexts encrypt inputs x and keys are issued for strings y. The striking feature of this system is that using the key SK_y to decrypt a ciphertext $\mathrm{CT}_x = \mathsf{Enc}(x)$, yields the value $F(x, y)$ but does not reveal anything else about x. Furthermore, *any arbitrary* collusion of key holders relative to many strings y_i does not yield any more information about x beyond what is "naturally revealed" by the functionality (i.e. $F(x, y_i)$ for all i).

Functional Encryption started with the notion of attribute-based encryption (ABE) [SW05, GPSW06] and evolved over time [BW07, KSW08]. The term functional encryption first appeared in [SW08], and formal definitions and constructions were subsequently given in [BSW11, O'N10]. Earlier influences include Identity-Based Encryption (IBE) [Sha85, BF01, Coc01] and searching on encrypted data [SWP00, BCOP04].

The general notion of functional encryption subsumes the notion of ABE: In ABE, encrypted messages consist of a secret payload m and a *public* attribute string x, where the secret key SK_y permits the recovery of the payload m if and only if $F(x, y) = 1$. While both ABE and FE require full collusion resilience, the critical difference between ABE and general FE is that ABE does not ensure the secrecy of the attribute string x. Recent breakthrough works [GVW13, GGH+13b, GGSW13] show how to achieve ABE for arbitrary circuits F, however these constructions do not provide any secrecy for the x input.

In contrast to ABE, existing FE schemes prior to our work are much more limited in power, with the state of the art roughly limited to the inner-product construction of Katz et al. [KSW08] and its adaptation to lattice-based cryptography by Agrawal et al. [AFV11]. In these systems ciphertexts and keys are associated with vectors $\mathbf{x}, \mathbf{y}$, and on decryption we only learn if the dot product $\mathbf{x} \cdot \mathbf{y}$ is zero. While there are interesting applications of this basic functionality, it is a far cry from the goal of realizing functional encryption for general functions, or even a large general class of circuits. There are also constructions that achieve only limited-collusion notions of security (and also several impossibility results for strong simulation-based security notions). We discuss these in Section 1.10 below. The central open question regarding functional encryption has been:

Does there exist a functional encryption scheme supporting all polynomial-size circuits?

1.1 Our Results

Our first result is a candidate construction of an indistinguishability obfuscator $i\mathcal{O}$ for all polynomial-size, log-depth circuits (i.e. $\mathbf{NC}^1$). The security of our construction relies on a new algebraic hardness assumption. We provide evidence for the plausibility of our assumption by proving it to be true in a specific generic model that seems to capture most "natural attacks" on this type of construction.

Our construction and assumption are staged in a framework that we call *Multilinear Jigsaw Puzzles.* Multilinear Jigsaw Puzzles are a simplified variant of multilinear maps, in which only the party who generated the system parameters can "encode elements in the exponent" of the different groups. These Multilinear Jigsaw Puzzles can be constructed by *eliminating* some components from the GGH (asymmetric) "approximate multilinear maps" [GGH13a]. Most notably, we eliminate the elements of the GGH setup that give rise to the most potent cryptanalytic attacks, including the "weak discrete log" attack [GGH13a]. We can also instantiate Multilinear Jigsaw Puzzles by eliminating some parts from the recent work of [CLT13] over the integers.

After constructing indistinguishability obfuscators for $\mathbf{NC}^1$ from Multilinear Jigsaw Puzzles, we use these indistinguishability obfuscators in a black-box manner to obtain the following results:

- Using indistinguishability obfuscator for $\mathbf{NC}^1$ together with any (leveled) fully homomorphic encryption (FHE) scheme with decryption in $\mathbf{NC}^1$ (e.g. [Gen09b, BV11, BGV12, Bra12, GSW13]), we show how to obtain an indistinguishability obfuscator for all polynomial-size circuits. The essential new idea here is to use a "two-key" encryption technique, similar in spirit to Naor-Yung [NY90].

- Using indistinguishability obfuscator for polynomial-size circuits, together with injective one-way functions, public-key encryption, and a novel variant of Sahai's simulation-sound non-interactive zero knowledge [Sah99] proofs, we show how to obtain functional encryption schemes supporting all polynomial-size circuits.

  Our construction achieves selective indistinguishability security, which can be amplified to full indistinguishability security using complexity leveraging. Finally, using the recent transformation of De Caro et al [CJO+13] one can obtain meaningful simulation-based security for functional encryption for all circuits.

  Our construction furthermore achieves *succinct* ciphertexts: The size of the ciphertexts in our scheme does not depend on potential secret key circuit sizes or even depth (nor do they depend on any parameter sizes needed for Multilinear Jigsaw Puzzles).

We emphasize that these results use our indistinguishability obfuscator for $\mathbf{NC}^1$ as a black box, and do not need to make further use of Multilinear Jigsaw Puzzles. We are able to obtain these results through a novel use of indistinguishability obfuscators, by making an intuitive connection between the use of indistinguishability obfuscation and witness indistinguishable proofs. (More on this below.)

1.2 Multilinear Jigsaw Puzzles

For our construction we use a variant of multilinear maps that offers only a *strict subset* of the functionality, and we term this variant Multilinear Jigsaw Puzzles. These are similar to (the asymmetric version of) the GGH multilinear encoding schemes [GGH13a], except that in our setting only the party that generated the system parameters is able to encode elements. Below is an intuitive description of Multilinear Jigsaw Puzzles using multiplicative notation for groups, a more formal treatment can be found in Section 2. Roughly speaking, an instance of a Multilinear Jigsaw Puzzle scheme includes a multilinear map system over groups of prime order p:

$$e : G_1 \times G_2 \times \cdots \times G_k \to G_T.$$

The system parameters allows a user to perform group and multilinear operations, but the parameters need not include canonical generators for the different groups $g_i \in G_i$ and $g_T \in G_T$. A *valid multilinear form* over such system is anything that can be computed using the group operation in the separate groups and the multilinear map. (For example, for $k = 3$ with variables $x_i \in G_1$, $y_i \in G_2$, $z_i \in G_3$ and $w_i \in G_T$, the expression $w_1 \cdot e(x_1, y_3, z_1 z_2)^2 \cdot e(x_2^3 x_4,\ y_1^2, z_2 z_5^3)^5 \cdot w_3^2$ is a valid form.)

In Multilinear Jigsaw Puzzle we view group elements as the puzzle pieces. The intuitive analogy to jigsaw puzzles is that these group elements can only be combined in very structured ways – like jigsaw puzzle pieces, different puzzle pieces either "fit" together or, if they do not "fit", then they cannot be combined in any meaningful way. We view a valid multilinear form in these elements as a suggested solution to this jigsaw puzzle: a valid multilinear form suggests ways to interlock the pieces together. A solution is correct if evaluating the multilinear form on the given elements yields the the unit in the target group $g_T^0 \in G_T$.

Thus, a Multilinear Jigsaw Puzzle scheme consists of just two algorithms, the Jigsaw generator that generates the system parameters and some group elements, and the Jigsaw verifier that checks whether a given solution is correct for these elements. In more detail, the Jigsaw generator outputs some system parameters prms and $k + 1$ nonempty sets of elements $S_i = \{x_1^{(i)}, \ldots, x_{n_i}^{(i)}\} \subset G_i$ for $i = 1, \ldots, k, T$. The Jigsaw verifier takes as input $(\mathsf{prms}, S_1, \ldots, S_k, S_T, \Pi)$, where Π and is a valid multilinear form in these elements, and it outputs "yes" if Π evaluates to the unit element in G_T on the given elements and "no" otherwise. Note that the Multilinear Jigsaw Puzzle scheme itself *does not specify which particular multilinear form the Jigsaw Verifier should apply*, this choice will typically come from the application.

The hardness assumptions that we make typically say that two output distributions of the Jigsaw generator are computationally indistinguishable. For instance, an (asymmetric) analog of the BDDH assumption for $k = 2$ could be that $(\{g_1, g_1^a, g_1^b\}, \{g_2, g_2^c\}, \{g_T, g_T^{abc}\})$ is indistinguishable from $(\{g_1, g_1^a, g_1^b\}, \{g_2, g_2^c\}, \{g_T, g_T^r\})$, where g_1, g_2, g_T are generators of G_1, G_2, and G_T, respectively, that satisfy $e(g_1, g_2) = g_T$.

1.3 Indistinguishability Obfuscation for $\mathbf{NC}^1$

We now sketch some elements of our candidate construction of indistinguishability obfuscator for $\mathbf{NC}^1$ using Multilinear Jigsaw Puzzles. A more detailed high-level overview of our construction can be found in Section 3, and the construction itself is described in Section 4.

Barrington's celebrated theorem [Bar86] shows that any $\mathbf{NC}^1$ circuit C can be transformed into an oblivious matrix branching program: That is, it can be transformed into a collection of k square matrices $M_1^0, M_2^0, \ldots M_k^0$ and another k square matrices $M_1^1, M_2^1, \ldots M_k^1$. The computation of the original circuit on an input x of length ℓ can then be reduced to a matrix product as follows, where $f : [k] \to [\ell]$ is a public function that is fixed in advance:

$$C(x) = 0 \quad \text{iff} \quad \prod_{i=1}^{k} M_i^{x_{f(i)}} = I$$

Our starting intuition is that this kind of evaluation is very well-suited for the setting of multilinear maps: The entries of the matrices M_i^0 and M_i^1 can be given in the group G_i: for instance, if the $(1,1)$ entry of M_i^0 is α, we can encode this as g_i^α. Then the multilinear map will allow us to compute the final product in G_T. We remark that our actual construction allows us to multiply k matrices together, not just k elements.

However, simply giving these matrices, even when encoded in the exponent, is insecure. Instead, we apply a combination of Kilian's matrix product randomization technique [Kil88] together with novel "multiplicative" randomization techniques that we develop. This yields a construction of an indistinguishability obfuscator for $\mathbf{NC}^1$ and a corresponding assumption; we justify our assumption by showing that it holds in a generic matrix model.

1.4 How to Use Indistinguishability Obfuscation

Now that we have constructed an indistinguishability obfuscator, we are faced with the question: what good is an indistinguishability obfuscator? The definition of indistinguishability obfuscation does not make clear what, *if anything*, an indistinguishability obfuscator actually hides about a circuit. In particular, if the circuit being obfuscated was already in an obvious canonical form, then we know that the indistinguishability obfuscator would not need to hide anything. We observe here an analogy to *witness indistinguishable proofs* [FS90]: if a statement being proven only has a unique witness, then a witness-indistinguishable proof does not need to hide the witness. The way witness indistinguishability can be used is by explicitly constructing statements that can have multiple witnesses. Similarly, we will use indistinguishability obfuscation by constructing circuits that inherently have multiple equivalent forms. We use this analogy to build our main application of functional encryption.

Warmup: Witness Encryption for NP. As a warmup to see how indistinguishability obfuscation can be applied, we start by considering a cryptographic notion which implicitly *already* considers circuits that inherently have multiple equivalent forms. Recall the notion of Witness Encryption for NP recently introduced in [GGSW13]: Given an NP Language L, a witness encryption scheme for L is an encryption scheme that takes as input an instance x and a message bit b, and outputs a ciphertext c. If $x \in L$ and w is a valid witness for x, then a decryptor can use w to decrypt c and recover b. However, if $x \notin L$, then an encryption of 0 should be computationally indistinguishable from an encryption of 1.

We now observe that Indistinguishability Obfuscation for $\mathbf{NC}^1$ implies Witness Encryption for an NP-Complete language such as $L =$ SATISFIABILITY. Consider the "point filter function" function [GK05] $F_{x,b}(w)$, defined as follows: if w is a valid witness for x, then $F_{x,b}(w) = b$. If w is an invalid witness for x, then $F_{x,b}(w) = \perp$. Note that for SATISFIABILITY, it is immediate that $F_{x,b}$ is in NC^1.

Now consider an Indistinguishability Obfuscation of $F_{x,b}$. This obfuscated circuit is a valid witness encryption for x, b. Correctness of decryption is immediate. To see why secrecy holds, note that if $x \notin L$, then both $F_{x,0}$ and $F_{x,1}$ are constant functions that always output $\perp$. Thus by the definition of Indistinguishability Obfuscation, the obfuscations of $F_{x,0}$ and $F_{x,1}$ are computationally indistinguishable.

1.5 Indistinguishability Obfuscation for all polynomial-size Circuits

Next, we show how to use indistinguishability obfuscation for $\mathbf{NC}^1$ and (leveled) fully homomorphic encryption (FHE) with decryption in $\mathbf{NC}^1$ to obtain indistinguishability obfuscation for all polynomial-size circuits. The main idea is to adapt the two-key paradigm [NY90] to work using indistinguishability obfuscators instead of witness-indistinguishable proofs. This construction is described in Section 5.

To obfuscate a circuit C, we choose and publish two FHE keys PK_0 and PK_1. We also give out encryptions of the circuit C under both FHE public keys, yielding ciphertexts c_0 and c_1. Finally, the obfuscation also has the indistinguishability obfuscation of a certain $\mathbf{NC}^1$ circuit C_{Dec0} that we will describe below.

The evaluator, who holds an input x, is told to use the FHE evaluation algorithm with x and *both* ciphertexts c_0 and c_1 to yield encryptions of $C(x)$ under both PK_0 and PK_1: let us call these encryptions e_0 and e_1 respectively. The evaluator also keeps track of all the intermediate bit values encountered during evaluation, which can be seen as a "proof" π that it used x to perform the evaluation correctly on both c_0 and c_1. The evaluator then feeds e_0, e_1, x, π into the obfuscated circuit $i\mathcal{O}(C_{Dec0})$. This circuit C_{Dec1} first checks the proof π to make sure that e_0 and e_1 were correctly computed – note that this can be done easily in $\mathbf{NC}^1$ since the evaluator has included all intermediate bit values encountered during evaluation, so the circuit merely needs to check that each appropriate triple of intermediate bit values respects the AND, OR, or NAND operations that were used during the evaluation process. If the proof checks out, then the circuit decrypts e_0 using SK_0, and outputs this decryption, which should be $C(x)$.

The key insight is that there is another $\mathbf{NC}^1$ circuit C_{Dec1} that is equivalent to C_{Dec0}, which simply decrypts e_1 using SK_1 instead. Because of the proof π that must be provided, both of these circuits always behave identically on all inputs. Furthermore, when using C_{Dec0}, we note that SK_1 is never used anywhere, and therefore the semantic security of the FHE scheme using PK_1 is maintained even given C_{Dec0}. Thus, by alternatively applying the semantic security of the FHE scheme and switching back and forth between C_{Dec0} and C_{Dec1} using the indistinguishability obfuscation property, we can prove that the obfuscation of any two equivalent circuits C and C' are computationally indistinguishable.

1.6 Functional Encryption for all circuits

We next sketch our main application of building functional encryption for all circuits. This construction is described in Section 6. The starting point idea for our solution is to pick a standard public-key encryption key pair $(\mathrm{PK}, \mathrm{SK})$ in the setup phase of the Functional Encryption. An encryption of a value x will simply be an encryption of x using the public key PK. The secret key sk_C corresponding to a circuit C be an obfuscation of a program that uses SK to decrypt x, then computes and output $C(x)$. While this solution would work if our obfuscator achieved the black-box obfuscation definition, there is no reason to believe that an indistinguishability obfuscator would necessarily hide SK.

Recall that the indistinguishability security definition for functional encryption requires the adversary to declare two inputs x_0 and x_1, with the promise that all secret keys SK_C that she will ask for will satisfy $C(x_0) = C(x_1)$. Then the security definition requires that the adversary will not be able to distinguish an encryption of x_0 from an encryption of x_1.

A natural first step would be to have two public keys PK_0 and PK_1, and require the encryption of x to consist of encryptions of x under both keys. However, in this case, unlike above, the receiver cannot generate a proof on his own that both ciphertexts encrypt the same message to provide to the obfuscated decryption circuit. Thus, we must require the encryptor to generate such a proof, which must hide x. A natural solution is to have the encryptor generate a non-interactive zero-knowledge (NIZK) proof that the ciphertexts both encrypt the same input. The obfuscated circuit would first check this proof, and if it checks out, it would use one of the two secret keys SK_0 or SK_1 for decryption and evaluation, confident that the output would be the same no matter which secret key it uses. When we are proving that the adversary cannot distinguish an encryption of x_0 from an encryption of x_1, we could move to an intermediate hybrid experiment where the NIZK proof is simulated, and the two ciphertexts are actually to x_0 and x_1 respectively.

Here we encounter the key new problem that we must tackle: when simulating a NIZK proof, we need to change the common reference string, and in all known NIZK systems, with respect to a simulated common reference string, valid proofs of *false* statements exist. Even with the notion of simulation soundness [Sah99], which was devised precisely to deal with this problem, valid proofs of false statements exist, even if they cannot be found by efficient adversaries. However, the notion of indistinguishability obfuscation requires that circuits be equivalent for *all inputs*, even inputs to the circuit that contain valid proofs for false statements. We address this problem by introducing the notion of *statistically simulation sound* NIZK, which requires that *except with respect to one particular statement to be simulated*, all other valid proofs that exist must be only for true statements[1]. Statistical simulation soundness can be achieved in a standard way using witness indistinguishable proofs. Once we have it, then a similar two-key alternation argument to the one given above allows us to establish security of our functional encryption scheme. We note that full collusion-resistance follows from a standard hybrid argument using the indistinguishability obfuscation definition for each secret key given out, one at a time.

Once we achieve indistinguishability security for functional encryption, this can be upgraded to natural simulation-based definitions of security using the work of De Caro et al. [CJO+13]. Furthermore, we note that our construction enjoys the property that ciphertexts are succinct; indeed their size depends only on

[1]Because this statement must be fixed in advance, the resulting security proof is for selective security. Selective security can be "upgraded" to full security using a standard complexity leveraging argument, by assuming subexponential security of the underlying cryptographic primitives that we use.

the public-key encryption scheme and NIZK being used, and does not depend on the underlying details of the obfuscation mechanism in any way (and therefore it does not depend on the parameters needed for our Multilinear Jigsaw Puzzles). We also place no a-priori bound on the circuit size of the circuits used for secret key generation.

1.6.1 Ciphertext Succinctness and Applications

One thing we emphasize is that ciphertexts in our scheme are very succinct in that their size depends only upon the message size and security parameter. In fact, if the right combination of public key encryption and NIZK is used the ciphertext size and encryption time can be considered small in a practical sense. Prior solutions for (the weaker) primitives of ABE for circuits [GVW13, GGH$^+$13b] and single use functional encryption [GKP$^+$13] achieved ciphertexts sizes and encryption times that were proportional to the maximum depth of the function to be evaluated. (It should be noted that these solutions rely on weaker assumptions.)

Following Parno et. al. [PRV12] we get a delegatable computation scheme with where the client's work is independent of the circuit size. Following Goldwasser et. al. [GKP$^+$13] we get a reusable Yao garbled circuits where the work to reuse a garbled circuit is proportional to the input size and independent of the size and depth of the circuit.

Finally, we get an efficient Private Linear Broadcast Encryption (PLBE) scheme as described by Boneh, Sahai, and Waters [BSW06], which they show implies a secure traitor tracing [CFN94] system. This final application requires collusion resistance.

1.7 An application to Restricted-Use Software

We have already given evidence that indistinguishability obfuscation has significant applications to cryptography. Here, we discuss an application of indistinguishability obfuscation to a question that does not deal with encryption or authentication, but rather *restricted-use software:*

Software developers will often want to release a demo or restricted use version of their software that limits the features that are available in a full version. In some cases a commercial software developer will do this to demonstrate their product; in other cases the developer will want to make multiple tiers of a product with different price points. In other domains, the software might be given to a partner that is only partially trusted and the developer only wants to release the features needed for the task.

Ideally, a developer could create a downgraded version of software simply by starting with the full version and then turning off certain features at the interface level — requiring minimal additional effort. However, if this is all that is done, it could be easy for an attacker to bypass these controls and gain access to the full version or the code behind it. The other alternative is for a software development team to carefully excise all unused functionality from the core of the software. Removing functionality can become a very time consuming task that could itself lead to the introduction of software bugs. In addition, in many applications it might be unclear what can and cannot remain for a restricted use version.

One immediate solution is for a developer to restrict the use at the interface level and then release an obfuscated version of the program. For this application indistinguishability obfuscation suffices, since by definition a version restricted in the interface is indistinguishable from an obfuscated program with equivalent behavior that has its smarts removed at the start.

1.8 Stronger notions of obfuscation

In this work, we give new constructions for secure obfuscation. While indistinguishability obfuscation is the technical focus of our work, we note that we know of no actual attacks on our obfuscation scheme as a black-box obfuscator *except for* the attacks that arise on specific circuit classes that are known to be unobfuscatable. This leads to the intriguing possibility that our construction (or future alternative constructions) could achieve black-box obfuscation for a large class of circuits, that somehow exclude the problematic examples that have been identified. Indeed, as observed by Goldwasser and Rothblum [GR07], indistinguishability obfuscation *must* yield virtual black-box obfuscation *if* such a virtual black-box obfuscation is possible for the function being obfuscated. In particular, while much study still needs to be done, we currently have no reason not to conjecture that our obfuscation mechanism suffices for such applications as:

- **Secure Patching of Software:** If a new malware vulnerability is found in software, there is a risk that releasing a software patch will allow attackers to become aware of a vulnerability before the patch has a chance to fully circulate among users. Obfuscation offers a solution concept: an initial patch can be released in obfuscated form, and then transitioned to a more efficient un-obfuscated patch once large-scale adoption has occurred for the initial patch. Here, the assumption would be that the obfuscated patch would hide where the vulnerability in the software was (at least as well as the original vulnerable software did).

- **"Intellectual Property" Protection:** If a new algorithm is developed for solving a problem, and there is a worry that available legal protections will not suffice for preventing reverse-engineering the algorithm, then obfuscation offers a natural solution: Here, the general required security property seems very close to black-box obfuscation, but if only certain parts of the algorithm is novel, weaker yet sufficient obfuscation security guarantees may be formalizable in some contexts.

1.9 Future Directions in Obfuscation

Our work opens up several new possibilities in obfuscation, below we discuss some of them.

- **The Power of Indistinguishability Obfuscation:** While a black box obfuscator immediately results in several turnkey applications, our work shows that indistinguishability obfuscation combined with adroit use of other cryptographic primitives can give rise to powerful functionalities. Seeing that indistinguishability obfuscation can be much more powerful than it initially appears, an important line of research is to see how far we can push the limits of what it gives us.

- **Indistinguishability Obfuscation from Weaker Assumptions:** The argument of security candidate for Indistinguishability Obfuscation clearly has a significant heuristic component. In future research, we can hope to gain better understanding of these heuristics and eventually achieve constructions under more standard assumptions. A major open problem is to obtain an unconditional proof for our scheme (or a variant of it) in a less idealized generic model, such as the (plain) generic multilinear map model, as opposed to the generic colored matrix model that we use. The eventual goal of this line of research may be to get an Indistinguishability Obfuscator provably secure under the Learning with Error (LWE) assumption (with the aid of complexity leveraging), perhaps using techniques similar to those developed in the context of FHE.

- **Studying the Possibility of Black Box Obfuscation:** As mentioned earlier we know of no actual attacks on our obfuscation scheme as a black-box obfuscator except for attacks that arise on specific circuit classes that are known to be unobfuscatable. A clear line of research is to better understand the potential for our construction to serve as a black box obfuscator, by identifying amenable function classes. One thrust is cryptanalysis, namely finding non-generic attacks when using our scheme, perhaps even against "natural" functionalities. A second direction is to attempt to gain some evidence of the viability for black box security of our construction in some heuristic model such as the generic group model. Of course, such a study would not be limited to our construction and we expect that other interesting candidates or variants of our construction would emerge in the near future.

- **Improving the Practical Efficiency:** While our current obfuscation construction runs in polynomial-time, it is likely too inefficient for most practical problems. An important objective is to improve the efficiency for use in practical applications.

- **Reasoning about Obfuscation of Learnable Programs in Practice:** A final important task is to begin to close the gap between the theoretical definitions for obfuscation and informal expectations that often arise in practice. Large software development projects can consume millions of dollars and hundreds of person years. A company that invested such an effort might reasonably expect that a "good" obfuscator would provide some real and tangible protection. For example, the company might hope that releasing an obfuscated version of their software would not be any "worse than" providing remote access to a server running the software.

One issue is that despite the substantial investment, the obfuscated program may be learnable in that its behavior can be completely determined by a (large) polynomial number of input queries. If this is the case, then simply giving out the program in the clear would technically count as valid obfuscation from the cryptographic definition even though this strategy would be very disappointing in a real sense.

We would like to be able to bridge this gap between theory and practice and figure out ways to rigorously define and argue about meaningful obfuscation notions for learnable programs. Getting a better understanding of this gap appears critical for using obfuscation in non-cryptographic applications. We emphasize that an important initial step is to simply understand this problem better.

1.10 Other Related Work

Some recent work has focused on a collusion-bounded form of functional encryption [SS10, GVW12, GKP$^+$13], where security is only ensured so long as an attacker does not acquire more than some predetermined number of keys. This concept is very similar to the manner in which one-time signatures relax the general notion of signatures. In these settings, collusion-bounded FE has been achieved for general circuits [SS10, GVW12], and in the case of single-key FE this has been done with quite succinct ciphertexts [GKP$^+$13]. Certain collusion-bounded functional encryption schemes with special properties suffice for several interesting applications such as delegated computation and most notably reusable garbled circuits [GKP$^+$13]. However, in the general use scenario envisioned for FE [BSW11], unbounded collusion resistance is essential, as a single user or a collection of users would be holding multiple secret keys. This is the focus of our work.

Recent work demonstrated that certain strong simulation-based formulations of the goal of functional encryption are impossible to meet [BSW11, AGVW12]. Our work focuses on realizing the indistinguishability based definition of FE, however recent work has shows that indistinguishability security for general circuits can be used to achieve meaningful simulation-based security as well [CJO$^+$13].

2 Preliminaries

In this section we will start by giving our definition of indistinguishability obfuscation ($i\mathcal{O}$) and the notions of Multilinear Jigsaw Puzzles and Barrington's branching program that will be needed in the realization of our obfuscation candidate.

2.1 Indistinguishability Obfuscators

Definition 1 (Indistinguishability Obfuscator ($i\mathcal{O}$))**.** A uniform PPT machine $i\mathcal{O}$ is called an *indistinguishability obfuscator* for a circuit class $\{\mathcal{C}_\lambda\}$ if the following conditions are satisfied:

- For all security parameters $\lambda \in \mathbb{N}$, for all $C \in \mathcal{C}_\lambda$, for all inputs x, we have that

$$\Pr[C'(x) = C(x) : C' \leftarrow i\mathcal{O}(\lambda, C)] = 1$$

- For any (not necessarily uniform) PPT distinguisher D, there exists a negligible function α such that the following holds: For all security parameters $\lambda \in \mathbb{N}$, for all pairs of circuits $C_0, C_1 \in \mathcal{C}_\lambda$, we have that if $C_0(x) = C_1(x)$ for all inputs x, then

$$\Big|\Pr\big[D(i\mathcal{O}(\lambda, C_0)) = 1\big] - \Pr\big[D(i\mathcal{O}(\lambda, C_1)) = 1\big]\Big| \leq \alpha(\lambda)$$

Definition 2 (Indistinguishability Obfuscator for NC^1)**.** A uniform PPT machine $i\mathcal{O}$ is called an *indistinguishability obfuscator* for NC^1 if for all constants $c \in \mathbb{N}$, the following holds: Let $\mathcal{C}_\lambda$ be the class of circuits of depth at most $c \log \lambda$ and size at most λ. Then $i\mathcal{O}(c, \cdot, \cdot)$ is an indistinguishability obfuscator for the class $\{\mathcal{C}_\lambda\}$.

Definition 3 (Indistinguishability Obfuscator for $P/poly$)**.** A uniform PPT machine $i\mathcal{O}$ is called an *indistinguishability obfuscator* for $P/poly$ if the following holds: Let $\mathcal{C}_\lambda$ be the class of circuits of size at most λ. Then $i\mathcal{O}$ is an indistinguishability obfuscator for the class $\{\mathcal{C}_\lambda\}$.

We also define a variant of the indistinguishability obfuscator definition that is suitable for use in uniform security proofs. This definition easily follows from the basic definition above with respect to non-uniform adversaries:

Definition 4 (Family-Indistinguishability Obfuscator ($i\mathcal{O}$))**.** A uniform PPT machine $i\mathcal{O}$ is called an *family-indistinguishability obfuscator* for a circuit class $\{\mathcal{C}_\lambda\}$ if the following conditions are satisfied:

- For all security parameters $\lambda \in \mathbb{N}$, for all $C \in \mathcal{C}_\lambda$, for all inputs x, we have that

$$\Pr[C'(x) = C(x) : C' \leftarrow i\mathcal{O}(\lambda, C)] = 1$$

- For any (not necessarily uniform) PPT adversaries *Samp*, D, there exists a negligible function α such that the following holds: if $\Pr[\forall x, C_0(x) = C_1(x) : (C_0, C_1, \sigma) \leftarrow Samp(1^\lambda)] > 1 - \alpha(\lambda)$, then we have:

$$\begin{aligned} \Big| &\Pr\left[D(\sigma, i\mathcal{O}(\lambda, C_0)) = 1 : (C_0, C_1, \sigma) \leftarrow Samp(1^\lambda)\right] \\ -&\Pr\left[D(\sigma, i\mathcal{O}(\lambda, C_1)) = 1 : (C_0, C_1, \sigma) \leftarrow Samp(1^\lambda)\right]\Big| \leq \alpha(\lambda) \end{aligned}$$

Observe that against non-uniform adversaries, every indistinguishability obfuscator for a circuit class $\mathcal{C}$ must also be a family-indistinguishability obfuscator for the circuit class $\mathcal{C}$. This is because we can non-uniformly fix the coins of *Samp* to produce a concrete (C_0, C_1, σ) that still yields non-negligible advantage for the distinguisher $D(\sigma, \cdot)$.

2.2 Multilinear Jigsaw Puzzles

In this section, we formalize our Multilinear Jigsaw Puzzle framework. The description of the framework does not refer to any specific implementation of multilinear maps; as such, we do not require any familiarity with the GGH multilinear encodings [GGH13a] for this section.

However, readers familiar with the GGH framework should think of Multilinear Jigsaw Puzzles as offering a *strict subset* of the functionality of (asymmetric) GGH multilinear encoding schemes [GGH13a]. The main difference is that only the party who generated the system parameters can encode elements. This simplified framework will suffice for our work (but perhaps not for other applications). Using this simplified variant offers several advantages: Most importantly, as opposed to [GGH13a] we *do not* provide nontrivial encodings of 1 and 0 in the public parameters, which avoids the "weak discrete logarithm" attack from [GGH13a]. For more information of the general GGH framework, we refer the reader to [GGH13a]. Note also that our multilinear puzzles framework can alternatively be implemented using a subset of the recent work on approximate multilinear maps over the integers [CLT13].

In the Multilinear Jigsaw Puzzle framework, there are two entities: the Jigsaw Generator, and the Jigsaw Verifier. Informally, the Jigsaw Generator takes as input a description of the "plaintext elements" that need to be encoded, and uses this to output the actual jigsaw puzzle pieces (which are encoding of those plaintext elements). We call them jigsaw puzzle pieces because they can only be meaningfully combined in very restricted ways (*i.e.* according to group operations and the multilinear map). The Jigsaw Verifier takes as input the jigsaw puzzle pieces and a specific Multilinear Form for combining these pieces. The output of the Jigsaw Verifier is 1 if the algorithm successfully "arranges the jigsaw puzzle" to produce a suitable encoding of 0.

We first formalize the manner in which the plaintext elements are specified. Intuitively we would like these elements to be given to the Jigsaw Generator as input, but in our setting it is the generator itself that choose the plaintext space $\mathbb{Z}_p$ from which these plaintext elements are taken, so we cannot give it the elements as input. Instead, the generator is given a *Jigsaw Specifier*, which is roughly an algorithm that takes p as input and produces as output the elements from $\mathbb{Z}_p$ that the generator should encode.

One helpful fact for reading the definition below, is that in asymmetric k-multilinear encoding schemes, there are "encoding levels" that correspond to subsets of the index set $[k] = \{1, 2, \ldots, k\}$. Hence the Jigsaw Specifier would output not only the plaintext elements but also the "levels" relative to which they need to be encoded.

Definition 5 (Jigsaw Specifier)**.** A *Jigsaw Specifier* is a tuple (k, ℓ, A) where $k, \ell \in \mathbb{Z}^+$ are parameters, and A is a probabilistic circuit with the following behavior: On input a prime number p, A outputs the prime p and an ordered set of ℓ pairs $(S_1, a_1), (S_2, a_2), \ldots, (S_\ell, a_\ell)$, where each $a_i \in \mathbb{Z}_p$ and each $S_i \subseteq [k]$.

For a polynomial $\alpha(\cdot)$, we say that a family of Jigsaw specifiers $\{(k_\lambda, \ell_\lambda, A_\lambda)\}_{\lambda \in \mathbb{Z}^+}$ is an *α-bounded Jigsaw Specifier family* if $k_\lambda, \ell_\lambda, |A_\lambda| < \alpha(\lambda)$.

We next formalize the notion of a Multilinear Form. This is a purely syntactic object, corresponding to an arithmetic circuit that we want to compute, so below we define forms with gates for addition, negation, and subtraction. We also include below special "ignore gates" that allow us to have more potential inputs than the number of real input to the circuit (and then we can choose only a subset of them for the actual computation).

Definition 6 (Multilinear Form)**.** A *Multilinear Form* is a tuple $\mathcal{F} = (k, \ell, \Pi, F)$, where $k, \ell \in \mathbb{Z}^+$ are parameters, and Π is a circuit with ℓ input wires, made out of binary addition $\oplus$ gates, binary multiplication $\otimes$ gates, unary negation $\ominus$ gates, and unary "ignore" $\square$ gates. F is an assignment of an index set $I \subseteq [k]$ to every wire of Π. A multilinear form must satisfy the following constraints:

(a) For every $\oplus$-gate or $\ominus$-gate, all the inputs and outputs of that gate are assigned the same set I;

(b) For every $\otimes$-gate, its two inputs are assigned disjoint sets $I_1, I_2 \subseteq [k]$, and its outputs are assigned the union set $I_1 \cup I_2$;

(c) The out-degree of all $\square$-gates is zero; and

(d) The output wire is assigned the set $[k]$.

A form $\mathcal{F} = (k, \ell, \Pi, F)$ is a-bounded (for some $a \in \mathbb{Z}$) if the size of the circuit Π is at most a.

We note that what makes this form "multilinear" is the requirement that the sets corresponding to the input of a multiplication gate be disjoint. This means that any index $i \in [k]$ can be added only once on any path from input to output, which implies that the output must be a multilinear in the inputs. Multilinear Forms are meant to be evaluated on the output from the Jigsaw Specifier, as defined next.

Definition 7 (Multilinear Evaluation)**.** Let $X = (p, (S_1, a_1), (S_2, a_2), \ldots, (S_\ell, a_\ell))$ be the output of the Jigsaw Specifier (k, ℓ, A) (with each $a_i \in \mathbb{Z}_p$ and each $S_i \subseteq [k]$). We say that a Multilinear Form $\mathcal{F} = (k', \ell', \Pi, F)$ is compatible with X if $k = k'$, $\ell = \ell'$, and the input wires of Π are assigned the sets $S_1, S_2, \ldots, S_\ell$.

If $\mathcal{F}$ is compatible with X then the evaluation of $\mathcal{F}$ on X, denoted $\mathcal{F}(X)$, is the output of the circuit Π on the input $(S_1, a_1), (S_2, a_2), \ldots, (S_\ell, a_\ell)$, defined as follows, where all arithmetic operations are over $\mathbb{Z}_p$:

- For every $\ominus$ gate, if the input is (S, a), then the output of the gate is $(S, -a)$.
- For every $\oplus$ gate with input (S, a_1) and (S, a_2), the output of the gate is $(S, a_1 + a_2)$.
- For every $\otimes$ gate with inputs (S_1, a_1) and (S_2, a_2) where S_1 and S_2 are disjoint, the output of the gate is $(S_1 \cup S_2, a_1 \cdot a_2)$.

All the operations above are performed in $\mathbb{Z}_p$. We say that the multilinear evaluation $\mathcal{F}(X)$ *succeeds* if the final output is $([k], 0)$.

With these definitions, we can now formally define Multilinear Jigsaw Puzzles:

Definition 8 (Multilinear Jigsaw Puzzle)**.** A Multilinear Jigsaw Puzzle scheme consists of two PPT algorithms, $\mathcal{MJP} = (\mathsf{JGen}, \mathsf{JVer})$, as follows:

Jigsaw Generator. The generator itself is specified via a pair of PPT algorithms $\mathsf{JGen} = (\mathsf{InstGen}, \mathsf{Encode})$:

- The randomized instance-generator $\mathsf{InstGen}$ takes as input the security parameter 1^λ and the multilinearity parameter k, and outputs a prime p of size at least 2^λ, public system parameters prms, and a secret state s to pass to the encoding algorithm, $(p, \mathsf{prms}, s) \leftarrow \mathsf{InstGen}(1^\lambda, 1^k)$.
- The (possibly randomized) encoding algorithm takes as input the prime p, the public parameters prms and secret state s, and a pair (S, a) with $S \subseteq [k]$ and $a \in \mathbb{Z}_p$ (which should be part of the output of the Jigsaw Specifier), and outputs an encoding of a relative to S. We denote this encoding by $(S, u) \leftarrow \mathsf{Encode}(p, \mathsf{prms}, s, S, a)$. (We assume for convenience that the index set S is included as part of the encoding.)

In our framework the Jigsaw Generator is given a Jigsaw Specifier (ℓ, k, A) and the security parameter λ, it first runs the instance-generation to get $(p, \mathsf{prms}, s) \leftarrow \mathsf{InstGen}(1^\lambda, 1^k)$, next runs the Jigsaw Specifier on input p to get $(p, (S_1, a_1), \dots, (S_\ell, a_\ell)) \leftarrow A(p)$, and finally encodes all the plaintext elements by running $(S_i, u_i) \leftarrow \mathsf{Encode}(\mathsf{prms}, s, S_i, a_i)$ for all $i = 1, \dots, \ell$.

The public output of the Jigsaw Generator, which we denote by puzzle, consists of the parameters prms, and also all the encodings (S_i, u_i). For notational convenience, however, below we consider an "extended output" that includes also the output of A (namely the plaintext version (S_i, a_i)). Namely, we denote

$$(p, X, \mathsf{puzzle}) \leftarrow \mathsf{JGen}(1^\lambda, k, \ell, A).$$

where $X = (p, (S_1, a_1), (S_2, a_2), \dots, (S_\ell, a_\ell))$ and $\mathsf{puzzle} = (\mathsf{prms}, (S_1, u_1), \dots, (S_\ell, u_\ell))$. We call puzzle the public output and X the private output. For ease of notation, if we want to refer only to the public output with respect to a Jigsaw Specifier (k, ℓ, A), we sometimes abuse notation and write puzzle_A to denote just the public output in the experiment above.

Jigsaw Verifier. The verifier JVer is a PPT algorithm that takes as input the public output of a Jigsaw Generator $\mathsf{puzzle} = (\mathsf{prms}, ((S_1, u_1), \dots, (S_\ell, u_\ell))$, and a Multilinear Form $\mathcal{F} = (k, \ell, \Pi, F)$. It outputs either accept (1) or reject (0).

For a particular generator output (p, X, puzzle) and a form $\mathcal{F}$ compatible with X, we say that the verifier JVer is *correct* relative to $(p, \mathsf{puzzle}, \mathcal{F}, X)$ if either $\mathcal{F}(X) = ([k], 0)$ and $\mathsf{JVer}(\mathsf{puzzle}, \mathcal{F}) = 1$ or $\mathcal{F}(X) \neq ([k], 0)$ and $\mathsf{JVer}(\mathsf{puzzle}, \mathcal{F}) = 0$. Otherwise JVer is *incorrect* relative to $(p, \mathsf{puzzle}, \mathcal{F}, X)$.

We require that with high probability over the randomness of the generator, the verifier will be correct *on all forms*. Specifically, if JVer is deterministic then we require that for any polynomial $\alpha(\cdot)$ and α-bounded Jigsaw Specifier family $\{(k_\lambda, \ell_\lambda, A_\lambda)\}_{\lambda \in \mathbb{Z}^+}$, we have

$$\Pr\left[(p, X, \mathsf{puzzle}) \leftarrow \mathsf{JGen}(1^\lambda, k_\lambda, \ell_\lambda, A_\lambda);\ \begin{array}{l} \exists\ \alpha(\lambda)\text{-bounded form } \mathcal{F} = (k_\lambda, \ell_\lambda, \Pi, F) \\ \text{compatible with } X \text{, s.t. } \mathsf{JVer} \text{ is incorrect} \\ \text{relative to } (p, \mathsf{puzzle}, \mathcal{F}, X) \end{array}\right] \leq negl(\lambda).$$

for some $negl(\cdot)$, a negligible function. [2]

[2]If JVer is randomized then the condition in the probability expression that JVer is incorrect is replaced by the condition that it is incorrect with probability more than $negl(\lambda)$. Our implementation of the Multilinear Jigsaw Puzzle framework will have a deterministic Jigsaw Verifier.

Security in the Multilinear Jigsaw Puzzle framework. The above formalization and guarantees speak to the correctness of the Multilinear Jigsaw Puzzle framework. For capturing security, we first consider what, intuitively, we want the Multilinear Jigsaw Puzzle framework to capture: We want it to be the case that the *only* way to distinguish between two different Jigsaw Puzzles puzzle_A and $\mathsf{puzzle}_{A'}$ is for there to be a particular multilinear form $\mathcal{F}$ that succeeds with noticeably different probabilities when evaluated on A vs. A'. Thus, we will consider distributions over puzzles puzzle_A and $\mathsf{puzzle}_{A'}$ where we do not believe that *any* multilinear form distinguishes between these puzzles, and our assumptions will state that these puzzles are computationally indistinguishable from each other.

Thus, more formally, hardness assumptions in the Multilinear Jigsaw Puzzle framework will require that for two different polynomial-size families of Jigsaw Specifiers $\{(k_\lambda, \ell_\lambda, A_\lambda)\}_{\lambda \in \mathbb{Z}^+}$ and $\{(k_\lambda, \ell_\lambda, A'_\lambda)\}_{\lambda \in \mathbb{Z}^+}$, the public output of the Jigsaw Generator on $(k_\lambda, \ell_\lambda, A_\lambda)$ will be computationally indistinguishable from the public output of the Jigsaw Generator on $(k_\lambda, \ell_\lambda, A'_\lambda)$.

In Appendix A, we describe how to implement the Multilinear Jigsaw Puzzle framework using a strict subset of [GGH13a].

2.3 Branching Programs

For our NC^1 functional encryption scheme we use "oblivious linear branching programs" as the underlying computational model. This is just the notion of branching programs as used in Barrington's theorem [Bar86], with the permutations encoded by permutation matrices. Namely, in this model a branching program consists of a sequence of steps, where in each step we examine one input bit, and depending on its value we choose one of two permutations. We then multiply all these permutations and the output of the computation is one if the resulting permutation is the identity. (Slightly more general, we can specify any pair of permutations π_0, π_1 to represent the outputs 0 and 1.)

Barrington's theorem requires only permutations in S_5, hence in the definition below we restrict ourselves to only this case. Also, below let $[w]$ denote the set $\{1, \dots, w\}$.

Definition 9 (Oblivious Linear Branching Program)**.** Let $A_0, A_1 \in \{0,1\}^{5\times 5}$ be two distinct permutation matrices. An (A_0, A_1) oblivious branching program of length-n for ℓ-bit inputs is a sequence

$$BP = \big((\mathsf{inp}(i), A_{i,0}, A_{i,1})\big)_{i=1}^{n},$$

where the $A_{i,b}$'s are permutation matrices in $\{0,1\}^{5\times 5}$, and $\mathsf{inp}(i) \in [\ell]$ is the input bit position examined in step i. The function computed by this branching program is

$$f_{BP,A_0,A_1}(x) \overset{\text{def}}{=} \begin{cases} 0 & \text{if } \prod_{i=1}^{n} A_{i,x_{\mathsf{inp}(i)}} = A_0 \\ 1 & \text{if } \prod_{i=1}^{n} A_{i,x_{\mathsf{inp}(i)}} = A_1 \\ \mathsf{undef} & \text{otherwise} \end{cases}$$

Theorem 1 ([Bar86])**.** There exist two distinct 5-cycle permutation matrices $A_0, A_1 \in \{0,1\}^{5\times 5}$, such that for any depth-d fan-in-2 Boolean circuit $C(\cdot)$, there exists an (A_0, A_1) oblivious linear branching program of length at most 4^d that computes the same function as the circuit C.

3 Intuition: Indistinguishability Obfuscation Candidate for $\mathbf{NC}^1$

To help explain the role of the different components in our cryptosystem, we describe below a trajectory that begins with Kilian's protocol for oblivious $\mathbf{NC}^1$ circuit evaluation, and modifying it until we end up with our obfuscation candidate for $\mathbf{NC}^1$ circuits.

Starting Point: Barrington and Kilian. Barrington's theorem [Bar86] that bounded-width branching programs can compute $\mathbf{NC}^1$ circuits played a major role in cryptography [GMW87, Kil88, BOGG$^+$88], providing a simple algebraic structure that facilitates structure-preserving randomization (as well as simple and natural defenses to standard attacks). In a nutshell, Barrington showed how to transform an arbitrary fan-in-2, depth-d boolean circuit $C : \{0,1\}^\ell \to \{0,1\}$ into a "width-5 permutation branching program" BP of length $n \le 4^d$. This program is defined by a sequence $BP = \{(\mathsf{inp}(i), A_{i,0}, A_{i,1})\}_{i=1}^n$, where the function $\mathsf{inp} : [n] \to [\ell]$ describes what input bit is examined in the i'th step, and the $A_{i,b}$'s are permutations in S_5 (which we can view as 5×5 permutation matrices). On any particular ℓ-bit input $(b_1 b_2 \dots b_\ell)$, the evaluation of BP consists of computing the product matrix $P = \prod_{i=1}^n A_{i,b_{\mathsf{inp}(i)}}$, outputting one if P is the identity and zero otherwise.

Let us recall Kilian's protocol [Kil88] in which the two players, Alice and Bob, evaluate an $\mathbf{NC}^1$ circuit on their joint input. Let $C(x,y)$ denote that circuit, with Alice's input x and Bob's input y, and $|x|+|y| = \ell$. Invoking Barrington's theorem, we get a length-n branching program BP that computes the same function as C. Alice begins the protocol by choosing n random invertible matrices $\{R_i\}_{i=1}^n$ over $\mathcal{Z}_p$, computing their inverses, and then setting $\tilde{A}_{i,b} = R_{i-1} A_{i,b} R_i^{-1}$ for all $i \in [n], b \in \{0,1\}$ (with index arithmetic modulo n).[3] Call the new program with $\tilde{A}_{i,b}$ the *randomized branching program RBP*. It is clear that RBP evaluates the same function as the original BP. Denoting the joint ℓ-bit input by $\chi = (x|y)$, Alice sends to Bob the matrices corresponding to her input x, $\{\tilde{A}_{i,\chi_{\mathsf{inp}(i)}} : i \in [n], \mathsf{inp}(i) \le |x|\}$. Next they use oblivious transfer protocol by which Bob learns the matrices corresponding to his input y, $\{\tilde{A}_{i,\chi_{\mathsf{inp}(i)}} : i \in [n], \mathsf{inp}(i) > |x|\}$. Now Bob can put the matrices in the proper order to compute the product $P = \prod_{i \in [n]} \tilde{A}_{i,\chi_{\mathsf{inp}(i)}}$, and hence $C(x,y)$. Alice learns nothing about Bob's input, and Kilian shows that Bob does not learn anything about Alice's input since the randomization hides Alice's input perfectly.

Scheme number one: Kilian's protocol as-is. Starting our journey from Kilian to $i\mathcal{O}$, we think of Alice as the obfuscator and Bob as the evaluator. The randomized branching program RBP for a universal circuit $C(x,y)$ in $\mathbf{NC}^1$ is generated during system startup. The x input corresponds to the specific circuit being obfuscated and y the input on which it is evaluated. The obfuscation consists of the matrices corresponding to her input x, $\{\tilde{A}_{i,\chi_{\mathsf{inp}(i)}} : i \in [n], \mathsf{inp}(i) \le |x|\}$ and all the matrixes for the inputs corresponding to y, namely $\{\tilde{A}_{i,b} : i \in [n], b \in 0,1, \mathsf{inp}(i) > |x|\}$. Bob during evaluation chooses the matrices corresponding to his input y, putting together with the matrices corresponding to input x, evaluates the program outputting $C(x,y)$.

Unsurprisingly, this scheme is broken. Informally speaking, however, we can prove that all attacks on this basic scheme fall into three categories:

- **Partial Evaluation Attacks.** A partial evaluation attack occurs when the adversary computes "partial" products $\left(\tilde{A}_{i,\chi_{\mathsf{inp}(i)}} \cdot \tilde{A}_{i+1,\chi_{\mathsf{inp}(i+1)}} \cdots \tilde{A}_{j,\chi_{\mathsf{inp}(j)}}\right)$ and $\left(\tilde{A}_{i,\chi'_{\mathsf{inp}(i)}} \cdot \tilde{A}_{i+1,\chi'_{\mathsf{inp}(i+1)}} \cdots \tilde{A}_{j,\chi'_{\mathsf{inp}(j)}}\right)$, corresponding to two inputs $\chi = (x,y)$ and $\chi' = (x,y')$, and compares these against each other. Note that the outer randomization matrices R_{i-1} and R_j^{-1} will be the same. This would allow the adversary to learn whether two partial computations in the branching program yield the same value, revealing information about the input x.

- **Mixed Input Attacks.** A mixed input attack occurs when the adversary performs the matrix product involved in a correct computation, but doesn't respect the inp function. That is, for different steps in the BP that consider the same input bit y_i, it uses some matrices that correspond to $y_i = 0$ and others that correspond to $y_i = 1$. Again, this could yield information about x beyond just the final output $C(x,y)$.

- **Attacks that do not respect algebraic structure or are not multilinear.** These attacks must either fail to respect the algebraic structure of the matrices over $\mathcal{Z}_p$, or compute non-multilinear algebraic functions over these matrices[4]. Informally speaking, we prove (in a particular idealized generic model), that all multilinear attacks over matrices must fall into one of the two attack categories above.

[3]Kilian uses random permutation matrices for the R_i's, but that aspect of the protocol is not relevant to our discussion.

[4]For example, Ishai [Ish13] pointed out to us that computing matrix inverses, a function of algebraic degree -1, can lead to some non-trivial attacks on this scheme.

Dealing with "other attacks" using Multilinear Jigsaw Puzzles. We begin by addressing the possibility of non-multilinear attacks, i.e., attacks that do things other than evaluating multilinear forms on the given elements. This is precisely where Multilinear Jigsaw Puzzles come in. In our case the Jigsaw Generator will be the obfuscator, the Jigsaw Specifier will give us the plaintext matrices as above, and the Jigsaw Verifier will be used when evaluating the program in order to decide if the resulting matrix is the identity or not. Informally speaking, the Jigsaw Generator will generate a puzzle system with multi-linearity n, where n is the length of the branching program to be obfuscated. Instead of revealing the matrices $\tilde{A}_{i,b}$ in cleartext, the obfuscator encodes each entry of the ith matrices $\tilde{A}_{i,b}$ relative to the singleton index set $\{i\} \subseteq [n]$.[5] Intuitively, this encoding makes it difficult to mount attacks that are not multilinear in the input elements. Roughly speaking, our hardness assumption states that it is infeasible to distinguish the encoding of two equal-length sequences of matrices that have the same behavior in terms of their products.

Dealing with Partial Evaluation Attacks: "Bookends" and Higher Dimensional Matrices. Note that encoding the matrices in the Multilinear Jigsaw Puzzle framework does nothing to prevent Partial Evaluation Attack and Mixed Input Attacks, as those attacks arise even when the adversary respects the matrix product structure.

To address the partial evaluation attack, we would like to add special "bookend" components at the start and end of the computation, without which no partial computation can be compared to any other. To do this, first we embed the matrices $A_{i,b}$ inside higher-dimension matrices of the form

$$D_{i,b} \sim \begin{pmatrix} \$ & & & \\ & \ddots & & \\ & & \$ & \\ & & & A_{i,b} \end{pmatrix}, \quad \tilde{D}_{i,b} \sim R_{i-1} \times \begin{pmatrix} \$ & & & \\ & \ddots & & \\ & & \$ & \\ & & & A_{i,b} \end{pmatrix} \times R_i^{-1},$$

where the \$'s are random coefficients on main diagonal that are *unique* to each matrix, the unspecified entries are 0, and the R_i's are random matrices over Z_p (which will be invertible with overwhelming probability).

Intuitively, encoding the matrices $\tilde{D}_{i,b}$ relative to the singleton insex set $\{i\}$ allows us to evaluate the multilinear form corresponding to their product, thereby recovering an encoding of the matrix $R_0 \cdot P \cdot R_n^{-1}$ (with P the product matrix) relative to the index set $[n]$. But now P is a high-dimension matrix with random random entries along the main diagonal, and the output of the function is encoded in the P's lower-right 5×5 quadrant (depending on whether or not it is the identity matrix). How can we "decode" the final output? To accomplish this, we add the special "bookend" terms that exactly cancel out the random diagonal entries.

Specifically, let the random part of the main diagonal have dimension $2m$ (for m to be specified shortly), so the matrices all have dimension $2m+5$. We generate two $(2m+5)$-vectors $\mathbf{s}$ and $\mathbf{t}$ of the form:

$$\mathbf{s} \sim (0 \ldots 0 \ \$ \ldots \$ \ \text{-}\ \mathbf{s}^* \text{-}\ ), \qquad \mathbf{t} \sim (\$ \ldots \$ \ \ 0 \ldots 0 \ \ \text{-}\ \mathbf{t}^* \text{-}\ ),$$

where the first 2 blocks of each vector have size m, and the last blocks have size 5. We also compute $\tilde{\mathbf{s}} = \mathbf{s}R_0^{-1}$ and $\tilde{\mathbf{t}} = R_n \mathbf{t}$.

Increasing the multilinearity level from n to $n+2$, the obfuscator adds encodings of the two bookend vectors $\hat{\mathbf{s}} = \mathsf{Enc}_{\{1\}}(\tilde{\mathbf{s}})$ and $\hat{\mathbf{t}} = \mathsf{Enc}_{\{n+2\}}(\tilde{\mathbf{t}})$ (where $\mathsf{Enc}_{\{i\}}(\cdot)$ denotes an encoding relative to the singleton $\{i\}$), and also adds to the obfuscation an encoding of the scalar $p^* = \langle \mathbf{s}^*, \mathbf{t}^* \rangle$ at level $[n+2]$. Now the evaluator can compute a level-$[n+2]$ encoding of

$$p = \tilde{\mathbf{s}} \cdot (R_0 P R_n^{-1}) \cdot \tilde{\mathbf{t}}^T = \mathbf{s}^* \cdot A \cdot \mathbf{t}^{*T} \quad \text{and} \quad p' = p - p^*$$

where the first $2m$ dimensions disappear due to the special structure of the $\mathbf{s}$, $\mathbf{t}$ and P. If A is indeed the identity matrix, then $\mathbf{s}^* \cdot A \cdot \mathbf{t}^{*T} = \langle \mathbf{s}^*, \mathbf{t}^* \rangle = p^*$, and so $p' = 0$ which can be verified using the Jigsaw Verifier. If A is some other permutation matrix, equality will not hold with high probability.

[5]In group notations, this is the equivalent of encoding the entries of that matrix in the exponent of the i'th group G_i.

Dealing with Mixed Input Attacks: Multiplicative Bundling. Finally, we must add a method that prevents the adversary from computing "mixed up" products of some matrices that correspond to $y_i = 0$ and other that correspond to some $y_i = 1$ (for the same input bit y_i). To defend against this threat we use a *multiplicative bundling* technique. The idea is simple, to prevent mixing and matching of components, Alice adds additional scalar multiples that help ensure that every "non-faithful" execution leads to a "garbled" output. Namely we choose scalars $\{\alpha_{i,b} : i \in [n], b \in \{0,1\}\}$ at random and set

$$D_{i,b} \sim \begin{pmatrix} \$ & & & \\ & \ddots & & \\ & & \$ & \\ & & & \alpha_{i,b} \cdot A_{i,b} \end{pmatrix}, \quad \tilde{D}_{i,b} \sim R_{i-1} \times \begin{pmatrix} \$ & & & \\ & \ddots & & \\ & & \$ & \\ & & & \alpha_{i,b} \cdot A_{i,b} \end{pmatrix} \times R_i^{-1}.$$

Unfortunately with these multiplicative factors, the encoded value of p^* is no longer sufficient to decode the output. To fix this issue, we need to provide in the public parameters something else that helps us to "cancel out" the $\alpha_{i,b}$'s. For that purpose, we introduce a "parallel" dummy branching program that computes the constant 1 function into our obfuscated program, and decoding the result is done by comparing the results from both programs.

Specifically, at system setup time we also generate a second set of matrices R'_i, vectors $\mathbf{s}'$ and $\mathbf{t}'$ with the same form as $\mathbf{s}$ and $\mathbf{t}$ where $\langle \mathbf{s}', \mathbf{t}' \rangle = \langle \mathbf{s}, \mathbf{t} \rangle$, and choose scalars $\{\alpha'_{i,b} : i \in [n], b \in \{0,1\}\}$ whose product match that of the α's, namely

$$\prod_{\mathsf{inp}(i)=j} \alpha_{i,b} = \prod_{\mathsf{inp}(i)=j} \alpha'_{i,b} \text{ for all } j \in [\ell],\ b \in \{0,1\}.$$

Then we set $\tilde{\mathbf{s}}' = \mathbf{s}' \cdot (R'_0)^{-1}$, $\tilde{\mathbf{t}}' = R'_n \cdot \mathbf{t}'^T$, and

$$D'_{i,b} \sim \begin{pmatrix} \$ & & & \\ & \ddots & & \\ & & \$ & \\ & & & \alpha'_{i,b} \cdot I \end{pmatrix}, \quad \tilde{D}'_{i,b} \sim R'_{i-1} \times \begin{pmatrix} \$ & & & \\ & \ddots & & \\ & & \$ & \\ & & & \alpha'_{i,b} \cdot I \end{pmatrix} \times R_i'^{-1},$$

Note that this has almost exactly the same structure as the "primary" branching program, except all the $A_{i,b}$'s are replaced by the identity matrix (hence this dummy program computes the constant 1 function). Also, this new copy is nearly independent of the primary program, except for the choice of the $\alpha_{i,b}$'s and $\alpha'_{i,b}$'s. The obfuscated program includes the "primary" vectors $\tilde{\mathbf{s}}, \tilde{\mathbf{t}}$ and matrices $\tilde{D}_{i,b}$, as well as the "dummy" vectors $\tilde{\mathbf{s}}', \tilde{\mathbf{t}}'$ and matrices $\tilde{D}'_{i,b}$.

Given all the encoded matrices and vectors, we can decode the output by checking if the two programs compute the same thing. Namely, we compute the difference between the two evaluations

$$\delta \ = \ \hat{\mathbf{s}} \cdot \left(\prod_{i \in [n]} \hat{D}_{i,b_i} \right) \cdot \hat{\mathbf{t}}^T \ - \ \hat{\mathbf{s}}' \cdot \left(\prod_{i \in [n]} \hat{D}'_{i,b_i} \right) \cdot \hat{\mathbf{t}}'^T,$$

and use the Jigsaw verifier to test if $\delta = 0$.

Observe that these random scalars indeed thwart mixed-evaluation attacks: For every input bit y_j, if we choose all the $\alpha_{i,0}$'s and all the $\alpha'_{i,0}$'s for $\mathsf{inp}(i) = j$ then the product matches and we could get $\delta = 0$ (similarly if we choose all the $\alpha_{i,1}$'s and $\alpha'_{i,1}$'s). However any attempt to mix-and-match these matrices will result in two different random products for the primary and dummy programs, which an adversary can only "cancel out" with probability $1/p$.

Additional Safeguards. In the construction above we choose to set the number m of "random dimensions" that are added to the matrices to more than the multi-linearity level of the system, namely set

$m = 2n+5$. We note that we do not have any concrete reason why even setting $m = 1$ is insecure, but we increase m as an additional safeguard against unanticipated attacks. By doing so, we increase the level of randomness present in both the $D_{i,b}$ matrices, and the individual R_i matrices to be well beyond the maximum multilinearity supported by the Multilinear Jigsaw Puzzle. In particular, this means that written as formal polynomials, each entry of R_i^{-1} is a polynomial with a degree exceeding the maximum multilinearity of the system. Intuitively, this gives us more confidence in our assumption because it seems quite implausible that an adversary that is essentially limited to computations that are "close" to multilinear forms would be able to detangle such high degree computations involving so many degrees of freedom in terms of the randomness used. However, again, we are not aware of any attacks on our scheme even without this additional safeguard.

4 Indistinguishability Obfuscation Candidate for $\mathbf{NC}^1$

In this section we describe our indistinguishability obfuscation candidate for $\mathbf{NC}^1$. We begin by presenting a notion of randomized branching programs, somewhat similar to Kilian's [Kil88], then build upon this construction using Multilinear Jigsaw Puzzles to get our indistinguishability obfuscation candidate.

Notations. Consider a length-n oblivious branching program over ℓ input variables,

$$BP = \{(\mathsf{inp}(i), A_{i,0}, A_{i,1}) : i \in [n], \mathsf{inp}(i) \in [\ell], A_{i,b} \in \{0,1\}^{5\times 5}\},$$

where $\mathsf{inp}(i) \in [\ell]$ is the position of the input bit that is examined in step i. We extend this notation naturally to a set of steps $S \subseteq [n]$, namely $\mathsf{inp}(S) = \{\mathsf{inp}(i) : i \in S\} \subseteq [\ell]$. Conversely, for bit position $j \in [\ell]$ we denote by I_j the steps in the branching program BP that examine the j'th input bit, $I_j = \{i \in [n] : \mathsf{inp}(i) = j\}$, and let $I_J = \cup\{I_j : j \in J\}$ for $J \subseteq [\ell]$.

4.1 Randomized Branching Programs

Like Kilian [Kil88], we express the steps of the branching program in terms of matrices, and randomize the i'th matrix by enveloping it by random matrices R_{i-1} and R_i^{-1}. However, in our setting, we also need some additional garbling techniques. Let $\mathbb{Z}_p$ be the ring over which we randomize the branching program, and let $m = 2n+5$. We generate a randomized branching program as follows.

1. Sample random and independent scalars $\{\alpha_{i,0}, \alpha_{i,1}, \alpha'_{i,0}, \alpha'_{i,1} : i \in [n]\}$ in the ring, subject to the constraint that $\prod_{i\in I_j} \alpha_{i,0} = \prod_{i\in I_j} \alpha'_{i,0}$ and $\prod_{i\in I_j} \alpha_{i,1} = \prod_{i\in I_j} \alpha'_{i,1}$ for all $j \in [\ell]$.

2. For every $i \in [n]$, compute two $(2m+5)\times(2m+5)$ block-diagonal matrices $D_{i,0}, D_{i,1}$ where the diagonal entries $1,\ldots,2m$ are chosen at random and the bottom-right 5×5 are the scaled $A_{j,b}$'s, and two more $(2m+5)\times(2m+5)$ matrices $D'_{i,0}, D'_{i,1}$ where the diagonal entries $1,\ldots,2m$ are random and the bottom-right 5×5 are the scaled identity:

$$D_{i,b} \sim \begin{pmatrix} \$ & & & \\ & \ddots & & \\ & & \$ & \\ & & & \alpha_{i,b}A_{i,b} \end{pmatrix}, \quad D'_{i,b} \sim \begin{pmatrix} \$ & & & \\ & \ddots & & \\ & & \$ & \\ & & & \alpha'_{i,b}I \end{pmatrix}, b \in \{0,1\}.$$

3. Choose vectors $\mathbf{s}$ and $\mathbf{t}$, and $\mathbf{s}'$ and $\mathbf{t}'$ of dimension $2m+5$ as follows:
   - The entries $1\ldots,m$ in the $\mathbf{s}$ and $\mathbf{s}'$ vectors are set to zero and entries $m+1,\ldots,2m$ are chosen at random. In the $\mathbf{t}$ and $\mathbf{t}'$ vectors the entries $1,\ldots,m$ are chosen at random and $m+1,\ldots,2m$ are set to zero.

- Choose two pairs of random 5-vectors $\mathbf{s}^*$ and $\mathbf{t}^*$, and $\mathbf{s}'^*$ and $\mathbf{t}'^*$, such that $\langle \mathbf{s}^*, \mathbf{t}^* \rangle = \langle \mathbf{s}'^*, \mathbf{t}'^* \rangle$. The last 5 entries in $\mathbf{s}$ are set to $\mathbf{s}^*$, the last 5 entries in $\mathbf{t}$ are set to $\mathbf{t}^*$. The last 5 entries in $\mathbf{s}'$ are set to $\mathbf{s}'^*$, the last 5 entries in $\mathbf{t}'$ are set to $\mathbf{t}'^*$.

$$\begin{aligned} \mathbf{s} &\sim (0 \dots 0 \ \$ \dots \$ \ \text{ - } \mathbf{s}^* \text{- }), & \mathbf{t} &\sim (\$ \dots \$ \ \ 0 \dots 0 \ \text{ - } \mathbf{t}^* \text{- })^T \\ \mathbf{s}' &\sim (0 \dots 0 \ \$ \dots \$ \ \text{ - } \mathbf{s}'^* \text{- }), & \mathbf{t}' &\sim (\$ \dots \$ \ \ 0 \dots 0 \ \text{ - } \mathbf{t}'^* \text{- })^T. \end{aligned}$$

4. Sample $2(n+1)$ random full-rank $(2m+5) \times (2m+5)$ matrices over the ring, $R_0, R_1, \dots, R_n$ and $R'_0, R'_1, \dots, R'_n$, and compute their inverses.

5. The randomized branching program over $\mathbb{Z}_p$ is the following:

$$\mathcal{RND}_p(BP) = \left\{ \begin{array}{ll} \tilde{\mathbf{s}} = \mathbf{s}R_0^{-1}, \ \tilde{\mathbf{t}} = R_n \mathbf{t}, & \tilde{\mathbf{s}}' = \mathbf{s}'(R'_0)^{-1}, \ \tilde{\mathbf{t}}' = R'_n \mathbf{t}' \\ \left\{ \tilde{D}_{i,b} = R_{i-1} D_{i,b} R_i^{-1} : i \in [n], b \in \{0,1\} \right\}, & \left\{ \tilde{D}'_{i,b} = R'_{i-1} D'_{i,b} (R'_i)^{-1} : i \in [n], b \in \{0,1\} \right\} \end{array} \right\} \quad (1)$$

We remark that this "randomized" program consists in essence of two parallel programs, one embeds the original branching program BP with all the $A_{i,b}$'s and the other embeds a "dummy program" of the same length, consisting only of identity matrices (so it computes the constant function 1). In our construction we use the dummy program for the purpose of equality test: the original program outputs 1 (on a given input) only when it agrees with the dummy program on that input.

4.2 Garbled Branching Programs and Hardness Assumptions

Below we describe a transformation for "garbling BPs with fixed inputs," and define formally our hardness assumption that says that this garbling transformation has good obfuscation properties.

We use Multilinear Jigsaw Puzzles as specified in Section 2.2. Let BP be a length-n linear branching program, computing some function $f : \{0,1\}^\ell \to \{0,1\}$. We apply our Multilinear Jigsaw Puzzle framework to the "Jigsaw specifier" that on input p randomizes the branching program over $\mathbb{Z}_p$ and outputs $\mathcal{RND}_p(BP)$ as above. That is, our branching-program garbling procedure begins by running the instance-generation part of the Jigsaw Generator to get $(p, \mathsf{prms}, s) \leftarrow \mathsf{InstGen}(1^\lambda, n+2)$. We then randomize the branching program over $\mathbb{Z}_p$ to get $\mathcal{RND}_p(BP)$, and use the encoding part of the Jigsaw generator to encode each element of the step-i matrices relative to the singleton index set $\{i+1\}$, each element of the vectors $\tilde{\mathbf{s}}, \tilde{\mathbf{s}}'$ relative to singleton index-set $\{1\}$, and each element of the vectors $\tilde{\mathbf{t}}, \tilde{\mathbf{t}}'$ relative to singleton index-set $\{n+2\}$. We denote the randomized and encoded program, which is the public output of the Jigsaw generator, by

$$\widehat{\mathcal{RND}}_p(BP) = \left\{ \begin{array}{ll} \mathsf{prms}, \quad \hat{\mathbf{s}} = \mathsf{Encode}_{\{1\}}(\tilde{\mathbf{s}}), \ \hat{\mathbf{t}} = \mathsf{Encode}_{\{n+2\}}(\tilde{\mathbf{t}}), & \hat{\mathbf{s}}' = \mathsf{Encode}_{\{1\}}(\tilde{\mathbf{s}}'), \ \hat{\mathbf{t}}' = \mathsf{Encode}_{\{n+2\}}(\tilde{\mathbf{t}}') \\ \left\{ \hat{D}_{i,b} = \mathsf{Encode}_{\{i+1\}}(\tilde{D}_{i,b}) : i \in [n], b \in \{0,1\} \right\}, & \left\{ \hat{D}'_{i,b} = \mathsf{Encode}_{\{i+1\}}(\tilde{D}'_{i,b}) : i \in [n], b \in \{0,1\} \right\} \end{array} \right\}.$$

The corresponding private output of the Jigsaw generator (i.e., plaintext) is $(p, \mathcal{RND}_p(BP))$.

For every input χ to the original branching program, we can choose the corresponding matrices from both the primary and the dummy programs and test whether they yield the same result using only the allowed multi-linearity. In more detail, for every input $\chi \in \{0,1\}^\ell$ we consider the Multilinear Form $\mathcal{F}_\chi$ which is defined as

$$\mathcal{F}_\chi\big(\mathcal{RND}_p(BP)\big) \ = \ \tilde{\mathbf{s}}\big(\prod_i \tilde{D}_{i,\chi_{\mathsf{inp}(i)}}\big)\tilde{\mathbf{t}} \ - \ \tilde{\mathbf{s}}'\big(\prod_i \tilde{D}'_{i,\chi_{\mathsf{inp}(i)}}\big)\tilde{\mathbf{t}}' \bmod p.$$

We observe that (a) if $BP(\chi) = 1$ then $F_\chi(\mathcal{RND}_p(BP)) = 0$ (with probability 1), and (b) if $BP(\chi) = 0$ then $\mathcal{F}_\chi(\mathcal{RND}_p(BP)) \neq 0$ except with probability 1/p. Given $\widehat{\mathcal{RND}}_p(BP)$ and χ, we can use the Jigsaw verifier to check if $F_\chi(\mathcal{RND}_p(BP)) = 0$, thereby learning whp the output of $BP(\chi)$.

4.2.1 Hardness assumptions

Roughly, we assume that if for two different ways of fixing some inputs to the branching program result in the same function on the remaining non-fixed inputs, then it is infeasible to decide which of the two sets of fixed inputs is used in a given garbled program.

Input-fixing. Given $\widehat{\mathcal{RND}}_p(BP)$ as above and a partial assignment for the input bits, $\sigma : J \to \{0,1\}$ (for $J \subset [\ell]$), the Parameter-fixing procedure just removes all the matrices $\tilde{D}_{i,b}$, $\tilde{D}'_{i,b}$ that are not consistent with that partial assignment σ (i.e., where $i \in I_j$ and $b \neq \sigma(\mathsf{inp}(i))$). Thus we have

$$\mathsf{GARBLE}\big(\widehat{\mathcal{RND}}_p(BP), (J,\sigma)\big) \quad = \quad \left\{ \begin{array}{ll} \mathsf{prms},\ \ \hat{\mathbf{s}},\ \hat{\mathbf{t}}, & \hat{\mathbf{s}}',\ \hat{\mathbf{t}}' \\ \big\{\hat{D}_{i,b} : i \in I_J, b = \sigma(\mathsf{inp}(i))\big\}, & \big\{\hat{D}'_{i,b} : i \in I_J, b = \sigma(\mathsf{inp}(i))\big\} \\ \big\{\hat{D}_{i,b} : i \notin I_J, b \in \{0,1\}\big\}, & \big\{\hat{D}'_{i,b} : i \notin I_J, b \in \{0,1\}\big\} \end{array} \right\}.$$

We think of a garbled program relative to the partial assignment (J,σ) as a representation of a function, one with the input restriction from (J,σ). If the underlying program is computing a function F and we have a partial assignment (J,σ), then we denote the resulting function by $F|_\sigma$.

Definition 10 (Functionally Equivalent Assignments)**.** Fix a function $F : \{0,1\}^\ell \to \{0,1\}$ and consider the two partial assignments over the same input variables, (J,σ_0) and (J,σ_1). We say that these assignments are *functionally equivalent relative to* F if $F|_{\sigma_0} = F|_{\sigma_1}$.

Assumption 1 (Equivalent Program Indistinguishability)**.** For any length-n branching program BP computing a function $F : \{0,1\}^\ell \to \{0,1\}$, and any two partial assignments on the same variables (J,σ_0) and (J,σ_1), if these assignments are functionally equivalent relative to F then the two corresponding garbled programs are computationally indistinguishable

$$\mathsf{GARBLE}\big(\ \widehat{\mathcal{RND}}(BP), (J,\sigma_0)\ \big) \ \overset{(c)}{\approx}\ \mathsf{GARBLE}\big(\ \widehat{\mathcal{RND}}(BP), (J,\sigma_1)\ \big).$$

4.3 Our Candidate

Theorem 2. Under the Equivalent Program Indistinguishability assumption (Assumption 1), there exists an efficient $i\mathcal{O}$ for $\mathbf{NC}^1$ circuits.

Proof. Fix a constant γ, and for every value of the security parameter λ let $\mathcal{C}_\lambda$ be the class of circuits of depth $\gamma \log \lambda$ and size at most λ. Let U_λ be a poly-sized universal circuit for this circuit class, where $U_\lambda(C,m) = C(m)$ for all $C \in \mathcal{C}_\lambda$ and $m \in \{0,1\}^n$. Furthermore, all circuits $C \in \mathcal{C}_\lambda$ can be encoded as an $\ell = \ell(\lambda)$ bit string as input to U. Let $UBP_\lambda(C,m)$ be the "universal branching program", which was obtained by applying Barrington's theorem to a universal circuit $U_\lambda(C,m)$ (its existence is guaranteed by [Bar86, Theorem 5]).

Denote by I_C the steps in the program UBP_λ that examine the input bits from the C input, and for each particular circuit c denote by (I_C,σ_c) the partial assignment that fixes the bits of that circuit in the input of UBP_λ. The obfuscator, on input circuit c, simply applies our garbling method from Section 4.2 to UBP_λ, with parameter-fixing (I_C,σ_c). Namely, we have

$$i\mathcal{O}(\lambda, c) = \mathsf{GARBLE}\big(\widehat{\mathcal{RND}}(UBP_\lambda),\ (I_C,\sigma_c)\big).$$

Functionality and polynomial slowdown are obvious, and Assumption 1 directly asserts that for any two circuits c_1, c_2 that compute the same function, $UBP(c_1,\cdot) \equiv UBP(c_2,\cdot)$, we have

$$i\mathcal{O}(\lambda, c_1) = \mathsf{GARBLE}\big(\widehat{\mathcal{RND}}(UBP_\lambda),\ (I_C,\sigma_{c_1})\big) \ \overset{(c)}{\approx}\ \mathsf{GARBLE}\big(\widehat{\mathcal{RND}}(UBP_\lambda),\ (I_C,\sigma_{c_2})\big) = i\mathcal{O}(\lambda, c_2).$$

■

5 Amplifying to Poly-sized Circuit Indistinguishability Obfuscation

In this section we show to achieve Poly-sized Circuit Indistinguishability Obfuscation from an indistinguishability obfuscator, $i\mathcal{O}_{\mathbf{NC}^1}$, for circuits in $\mathbf{NC}^1$.

In addition to BPO, our construction makes use of two primitives: Perfectly Sound Non-Interactive Witness Indistinguishable Proofs and Fully Homomorphic Encryption. We also make use of the fact that any poly-time circuit computation can be verified by a "low=depth" circuit in $\mathbf{NC}^1$. (Background material for these primitives can be found in Appendix B.) We let $(\mathsf{Setup}_{FHE}, \mathsf{Encrypt}_{FHE}, \mathsf{Eval}_{FHE}, \mathsf{Decrypt}_{FHE})$. Furthermore, we assume the decryption algorithm $\mathsf{Decrypt}_{FHE}$ can be realized by a family of circuits in $\mathbf{NC}^1$ and that the system has *perfect correctness*.

5.1 Our Construction

Consider a family of circuit classes $\{\mathcal{C}_\lambda\}$ for $\lambda \in \mathbb{N}$ where both the input size, $n = n(\lambda)$, is a polynomial function of λ and the maximum circuit size, $p(\lambda)$ is also a polynomial function of λ. Let $\{U_\lambda\}$ be a poly-sized universal circuit family for these circuit classes, where $U_\lambda(C, m) = C(m)$ for all $C \in \{\mathcal{C}_\lambda\}$ and $m \in \{0,1\}^n$. Furthermore, all circuits $C \in \{\mathcal{C}_\lambda\}$ can be encoded as an $\ell = \ell(\lambda)$ bit string as input to U.

We show how to build an $i\mathcal{O}$ for such a circuit class given an indistinguishability obfuscator, $i\mathcal{O}_{\mathbf{NC}^1}$, for circuits in $\mathbf{NC}^1$.

Our construction is described by an obfuscate algorithm and an evaluation algorithm.[6]

- $\mathsf{Obfuscate}(1^\lambda, C \in \mathcal{C}_\lambda)$: The Setup_{FE} algorithm takes the security parameter λ and computes

  1. Generate $(\mathsf{PK}^1_{FHE}, \mathsf{SK}^1_{FHE}) \leftarrow \mathsf{Setup}_{FHE}(1^\lambda)$ and $(\mathsf{PK}^2_{FHE}, \mathsf{SK}^2_{FHE}) \leftarrow \mathsf{Setup}_{FHE}(1^\lambda)$. If we are using a *leveled* FHE scheme, the number of levels should be set to be the depth of U_λ.
  2. Generate ciphertexts $g_1 = \mathsf{Encrypt}_{FHE}(\mathsf{PK}^1_{FHE}, C)$ and $g_2 = \mathsf{Encrypt}_{FHE}(\mathsf{PK}^2_{FHE}, C)$. Here we assume that C is encoded in a canonical form as an ℓ bit string for use by the universal circuit $U_\lambda(\cdot,\cdot)$
  3. Generate an NC^1 obfuscation for program $\mathrm{P1}^{(\mathsf{SK}^1_{FHE}, g_1, g_2)}$ as $P = i\mathcal{O}_{\mathbf{NC}^1}(\mathrm{P1}^{(\mathsf{SK}^1_{FHE}, g_1, g_2)})$. (See Figure 1.)
  4. The obfuscation components are output as: $\sigma = (P, \mathsf{PK}^1_{FHE}, \mathsf{PK}^2_{FHE}, g_1, g_2)$.

- $\mathsf{Evaluate}(\sigma = (P, \mathsf{PK}^1_{FHE}, \mathsf{PK}^2_{FHE}, g_1, g_2), m)$ The $\mathsf{Evaluate}$ algorithm takes in the obfuscation output σ and program input m and computes the following.

  1. Compute $e_1 = \mathsf{Eval}_{FHE}(\mathsf{PK}^1_{FHE}, U_\lambda(\cdot, m), g_1)$ and $e_2 = \mathsf{Eval}_{FHE}(\mathsf{PK}^2_{FHE}, U_\lambda(\cdot, m), g_2)$. [7]
  2. Compute a low depth proof ϕ that e_1 and e_2 were computed correctly.
  3. Run $P(m, e_1, e_2, \phi)$ and output the result.

Correctness To verify correctness we first check that the size of the circuits evaluating $\mathrm{P1}^{(\mathsf{SK}^1_{FHE}, g_1, g_2)}$ are in $\mathbf{NC}^1$. In step 1 the program is the verification is in $\mathbf{NC}^1$ since we applied a low depth proof as described in Appendix B.4. The only other step of the program is also in $\mathbf{NC}^1$ since we use an FHE scheme with decryption in $\mathbf{NC}^1$. Since both steps are in $\mathbf{NC}^1$ the entire circuit is in $\mathbf{NC}^1$.

The correctness of FHE encryption means that g_1 will be an encryption of the circuit C. And the correctness of FHE evaluation means that e_1 will be an encryption of $U(C, m) = C(m)$. The correctness of

[6] Technically, we could make do with just a single obfuscation algorithm that outputs a circuit description as is the convention given in Section 2. However, for the exposition of this construction we have the obfuscate algorithm output a cryptographic material that is used by the evaluation algorithm.

[7] The circuit $U_\lambda(\cdot, m)$ is the universal circuit with m hardwired in as an input. This in hardwired circuit takes in an ℓ bit circuit description C as its input and evaluates to $U(C, m)$.

the $i\mathcal{O}_{\mathbf{NC}^1}$ guarantees the obfuscation of the program $\mathrm{P1}^{(\mathsf{SK}^1_{FHE}, g_1, g_2)}$ will be executed faithfully. The check step program itself will pass step one on honest execution and that the result of $C(m)$ will be output.

We describe the two program classes in the two figures below.

P1

Given input (m, e_1, e_2, ϕ), $\mathrm{P1}^{(\mathsf{SK}^1_{FHE}, g_1, g_2)}$ proceeds as follows:

1. Check if ϕ is a valid low-depth proof for the NP-statement:

$$e_1 = \mathsf{Eval}_{FHE}(\mathsf{PK}^1_{FHE}, U_\lambda(\cdot, m), g_1) \quad \bigwedge \quad e_2 = \mathsf{Eval}_{FHE}(\mathsf{PK}^2_{FHE}, U_\lambda(\cdot, m), g_2).$$

2. If the check fails output 0; otherwise, output $\mathsf{Decrypt}_{FHE}(e_1, \mathsf{SK}^1_{FHE})$.

Figure 1:

P2

Given input (m, e_1, e_2, ϕ), $\mathrm{P2}^{(\mathsf{SK}^2_{FHE}, g_1, g_2)}$ proceeds as follows:

1. Check if ϕ is a valid low-depth proof for the NP-statement:

$$e_1 = \mathsf{Eval}_{FHE}(\mathsf{PK}^1_{FHE}, U_\lambda(\cdot, m), g_1) \quad \bigwedge \quad e_2 = \mathsf{Eval}_{FHE}(\mathsf{PK}^2_{FHE}, U_\lambda(\cdot, m), g_2).$$

2. If the check fails output 0; otherwise, output $\mathsf{Decrypt}_{FHE}(e_2, \mathsf{SK}^2_{FHE})$.

Figure 2:

5.2 Proof of Security

We prove that for all $C_0, C_1 \in \mathcal{C}_\lambda$ there can be no poly-time indistinguishability attacker $\mathcal{A}$ that wins the security game from Section 2 with non-negligible advantage.

We organize our proof into a sequence of hybrids. In the first hybrid the challenger obfuscates C_0. We then gradually change the obfuscation in multiple hybrid steps into an obfuscation of C_1. We show that each successive hybrid experiment is indistinguishable from the last, thus showing our obfuscator to have indistinguishability security. The proof hybrid steps themselves primarily weave back and forth in between changing the underlying ciphertexts and the programs that are used in a two-key proof type manner.

Sequence of Hybrids

- Hyb_0: This hybrid corresponds to a honest execution of the Indistinguishability Obfuscation game where C_0 is obfuscated.

- Hyb_1: Same as hybrid Hyb_0 except we now generate $g_1 = \mathsf{Encrypt}_{FHE}(\mathsf{PK}^1_{FHE}, C_0)$ and $g_2 = \mathsf{Encrypt}_{FHE}(\mathsf{PK}^2_{FHE}, C_1)$. Now g_1 and g_2 encrypt different circuits.

- Hyb_2: We still generate $g_1 = \mathsf{Encrypt}_{FHE}(\mathsf{PK}^1_{FHE}, C_0)$ and $g_2 = \mathsf{Encrypt}_{FHE}(\mathsf{PK}^2_{FHE}, C_1)$ as in Hyb_1. Now P is created as $P = i\mathcal{O}_{\mathbf{NC}^1}(\mathrm{P2}^{(\mathsf{SK}^2_{FHE}, g_1, g_2)})$.

- Hyb_3: We now generate $g_1 = \mathsf{Encrypt}_{FHE}(\mathsf{PK}^1_{FHE}, C_1)$ and $g_2 = \mathsf{Encrypt}_{FHE}(\mathsf{PK}^2_{FHE}, C_1)$. The obfuscated program P is still created as $P = i\mathcal{O}_{\mathbf{NC}^1}(\mathrm{P2}^{(\mathsf{SK}^2_{FHE}, g_1, g_2)})$.

- Hyb_4: We still generate $g_1 = \mathsf{Encrypt}_{FHE}(\mathsf{PK}^1_{FHE}, C_0)$ and $g_2 = \mathsf{Encrypt}_{FHE}(\mathsf{PK}^2_{FHE}, C_1)$ as in Hyb_3. Now P is created as $P = i\mathcal{O}_{\mathbf{NC}^1}(\mathrm{P1}^{(\mathsf{SK}^1_{FHE}, g_1, g_2)})$.

Proofs of Hybrid Arguments

Claim 1. If our FHE scheme is IND-CPA secure, then no poly-time attacker can distinguish with non-negligible probability between Hyb_0 and Hyb_1.

Proof. We show that if there is a poly-time attacker $\mathcal{A}$ that has a non-negligible difference in advantage between Hyb_0 and Hyb_1 , then there is a poly-time algorithm $\mathcal{B}$ that breaks the IND-CPA security of our PKE scheme. $\mathcal{B}$ begins by running $\mathcal{A}$ and receiving C_0, C_1.

$\mathcal{B}$ first gets PK^2_{FHE} from the IND-CPA challenger. Next, it gives the IND-CPA challenger C_0, C_1 and gets back a ciphertext g'. It sets $g_2 = g'$. $\mathcal{B}$ then sets $g_1 = \mathsf{Encrypt}_{FHE}(\mathsf{PK}^1_{FHE}, C_0)$ and P is created as $P = i\mathcal{O}_{\mathbf{NC}^1}(\mathrm{P1}^{(\mathsf{SK}^1_{FHE}, g_1, g_2)})$. *Note only the first secret key is needed to create P.*

If the IND-CPA challenger used the first message C_0 , then we are exactly in hybrid Hyb_0; if it chose the second message C_1, then we are in Hyb_1. Therefore if an attacker can distinguish between the two hybrids with non-negligible advantage, it will break the IND-CPA property of the PKE scheme.

■

Claim 2. If the Family-$i\mathcal{O}$ assumption holds for our $\mathbf{NC}^1$ indistinguishability obfuscator, then no poly-time attacker can distinguish between Hyb_1 and Hyb_2.

Proof. We show that if there is a poly-time attacker $\mathcal{A}$ that has a non-negligible difference in advantage between Hyb_0 and Hyb_1, then there is a poly-time algorithm $\mathcal{B}$ that breaks the IND-CPA security of our PKE scheme. $\mathcal{B}$ begins by running $\mathcal{A}$ and receiving C_0, C_1.

$\mathcal{B}$ first generates the two FHE private keys itself, keeping both secret keys. It then creates $g_1 = \mathsf{Encrypt}_{FHE}(\mathsf{PK}^1_{FHE}, C_0)$ and $g_2 = \mathsf{Encrypt}_{FHE}(\mathsf{PK}^2_{FHE}, C_1)$. Next, it submits the circuits $\mathrm{P1}^{(\mathsf{SK}^1_{FHE}, g_1, g_2)}$ and $\mathrm{P2}^{(\mathsf{SK}^2_{FHE}, g_1, g_2)}$ to the indistinguishability obfuscator challenger. It receives back a program P' and sets $P = P'$.

Suppose that output of C_0 is equivalent to the output of C_1 on all inputs. Then both programs $\mathrm{P1}^{(\mathsf{SK}^1_{FHE}, g_1, g_2)}$ and $\mathrm{P2}^{(\mathsf{SK}^2_{FHE}, g_1, g_2)}$ will have the same output on all inputs. Both inputs will halt and output 0 if the check does not pass. If the check does pass, this means that e_1 and e_2 are encryptions of the same message. This is due to the *perfect* correctness of FHE encryption and evaluation together with the fact that $U(C_0, m) = U(C_1, m)$ for all m. Since both of the programs give the same output on all inputs, we are in a valid instance of the assumption.

If the IND-CPA challenger used the first circuit $\mathrm{P1}^{(\mathsf{SK}^1_{FHE}, g_1, g_2)}$, then we are exactly in hybrid Hyb_1; if it chose the second circuit $\mathrm{P2}^{(\mathsf{SK}^2_{FHE}, g_1, g_2)}$, then we are in Hyb_2. Therefore if an attacker can distinguish between the two hybrids with non-negligible advantage, it will break the indistinguishability security of the obfuscator.

■

Claim 3. If our FHE scheme is IND-CPA secure, then no poly-time attacker can distinguish with non-negligible probability between Hyb_0 and Hyb_1.

Proof. We show that if there is a poly-time attacker $\mathcal{A}$ that has a non-negligible difference in advantage between Hyb_2 and Hyb_3 , then there is a poly-time algorithm $\mathcal{B}$ that breaks the IND-CPA security of our PKE scheme. $\mathcal{B}$ begins by running $\mathcal{A}$ and receiving C_0, C_1.

$\mathcal{B}$ first gets PK^1_{FHE} from the IND-CPA challenger. Next, it gives the IND-CPA challenger C_0, C_1 and gets back a ciphertext g'. It sets $g_1 = g'$. $\mathcal{B}$ then sets $g_2 = \mathsf{Encrypt}_{FHE}(\mathsf{PK}^2_{FHE}, C_1)$ and P is created as $P = i\mathcal{O}_{\mathbf{NC}^1}(\mathrm{P2}^{(\mathsf{SK}^2_{FHE}, g_1, g_2)})$. *Note only the second secret key is needed to create P.*

If the IND-CPA challenger used the first message C_0 , then we are exactly in hybrid Hyb_2; if it chose the second message C_1, then we are in Hyb_3. Therefore if an attacker can distinguish between the two hybrids with non-negligible advantage, it will break the IND-CPA property of the PKE scheme.

■

Claim 4. If the Family-$i\mathcal{O}$ assumption holds for our $\mathbf{NC}^1$ indistinguishability obfuscator, then no poly-time attacker can distinguish between Hyb_3 and Hyb_4.

Proof. We show that if there is a poly-time attacker $\mathcal{A}$ that has a non-negligible difference in advantage between Hyb_3 and Hyb_4, then there is a poly-time algorithm $\mathcal{B}$ that breaks the IND-CPA security of our PKE scheme. $\mathcal{B}$ begins by running $\mathcal{A}$ and receiving C_0, C_1.

$\mathcal{B}$ first generates the two FHE private keys itself, keeping both secret keys. It then creates $g_1 = \mathsf{Encrypt}_{FHE}(\mathsf{PK}^1_{FHE}, C_0)$ and $g_2 = \mathsf{Encrypt}_{FHE}(\mathsf{PK}^2_{FHE}, C_1)$. Next, it submits the circuits $\mathrm{P2}^{(\mathsf{SK}^2_{FHE}, g_1, g_2)}$ and $\mathrm{P1}^{(\mathsf{SK}^1_{FHE}, g_1, g_2)}$ to the indistinguishability obfuscator challenger. It receives back a program P' and sets $P = P'$.

Both programs $\mathrm{P1}^{(\mathsf{SK}^1_{FHE}, g_1, g_2)}$ and $\mathrm{P2}^{(\mathsf{SK}^2_{FHE}, g_1, g_2)}$ will have the same output on all inputs. Both inputs will halt and output 0 if the check does not pass. If the check does pass, this means that e_1 and e_2 are encryptions of the same message. This is due to the *perfect* correctness of FHE encryption and evaluation. The FHE evaluation of both programs will evaluate $U(C_1, m)$. Since both of the programs give the same output on all inputs, we are in a valid instance of the assumption.

If the IND-CPA challenger used the first circuit $\mathrm{P2}^{(\mathsf{SK}^2_{FHE}, g_1, g_2)}$, then we are exactly in hybrid Hyb_1; if it chose the second circuit $\mathrm{P1}^{(\mathsf{SK}^1_{FHE}, g_1, g_2)}$, then we are in Hyb_2. Therefore if an attacker can distinguish between the two hybrids with non-negligible advantage, it will break the indistinguishability security of the obfuscator.

■

Theorem 3. Under the assumptions listed above our $i\mathcal{O}$ for poly-sized circuits is secure in the indistinguishability game defined in Section 2.

Proof. Our claims show a succession of five hybrids where no poly-time attacker can distinguish one from the next with non-negligible advantage. The first hybrid Hyb_0 corresponds to obfuscating circuit C_0 and the last hybrid Hyb_4 corresponds to obfuscating the circuit C_1 in the indistinguishability security game. Security in the indistinguishability game follows. ■

6 Functional Encryption

Our syntax for functional encryption roughly follows in the line of Boneh-Sahai-Waters [BSW11] except we specialize our notation for the case where the private key is a function f and the ciphertext input is a message m. This is without loss of generality when f can be any poly-sized circuit and thus includes a universal circuit.

For security we use the indistinguishability notion, which was the first one considered for functional encryption (as well as predicate encryption [BW07, KSW08]). Indistinguishability is a basic notion of security which may be enough for some applications, and furthermore, De Caro et. al. [CJO+13] show how in the random oracle model one can transform a system with indistinguishability secure into one with strong simulation security.

Definition 11 (Functional Encryption)**.** A *functional encryption scheme* for a class of functions $\mathcal{F} = \mathcal{F}(\lambda)$ over message space $\mathcal{M} = \mathcal{M}_\lambda$ consists of four algorithms $\mathcal{FE} = \{\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}\}$:

- $\mathsf{Setup}(\mathbf{1}^\lambda)$ – a polynomial time algorithm that takes the unitary representation of the security parameter λ and outputs a public parameters PP and a master secret key MSK.
- $\mathsf{KeyGen}(\mathrm{MSK}, f)$ – a polynomial time algorithm that takes as input the master secret key MSK and a description of function $f \in \mathcal{F}$ and outputs a corresponding secret key SK_f.
- $\mathsf{Encrypt}(\mathrm{PP}, x)$ – a polynomial time algorithm that takes the public parameters PP and a string x and outputs a ciphertext CT.

- $\mathsf{Decrypt}(\mathrm{SK}_f, \mathrm{CT})$ – a polynomial time algorithm that takes a secret key SK_f and ciphertext encrypting message $m \in \mathcal{M}$ and outputs $f(m)$.

A functional encryption scheme is correct for $\mathcal{F}$ if for all $f \in \mathcal{F}$ and all messages $m \in \mathcal{M}$:

$$\Pr[\ (PK, \mathrm{MSK}) \leftarrow \mathsf{Setup}(\mathbf{1}^\lambda);\ \ \mathsf{Decrypt}(\mathsf{KeyGen}(\mathrm{MSK}, f), \mathsf{Encrypt}(PK, m)) \neq f(m)\ ] = negl(\lambda)$$

Indistinguishability Security for Functional Encryption

We describe indistinguishability security as a multi-phased game between an attacker $\mathcal{A}$ and a challenger.

Setup: The challengers runs $(\mathrm{PP}, \mathrm{MSK}) \leftarrow \mathsf{Setup}(1^\lambda)$ and gives PP to $\mathcal{A}$.

Query 1: $\mathcal{A}$ adaptively submits queries f_i in $\mathcal{F}$ and is given $\mathrm{SK}_{\leftarrow}\mathsf{KeyGen}(\mathrm{MSK}, f)$. This step can be repeated any polynomial number of times by the attacker.

Challenge: $\mathcal{A}$ submits two messages $m_0, m_1 \in \mathcal{M}$ such that $f(m_0) = f_i(m_1)$ for all f queried in the key query phase and is given $\mathsf{Encrypt}(\mathrm{PP}, m_b)$.

Query 2: $\mathcal{A}$ continues to issue key queries as before subject to the same restriction that any f queried must satisfy $f(m_0) = f(m_1)$.

Guess $\mathcal{A}$ eventually outputs a bit b' in $\{0, 1\}$.

The advantage of an algorithm $\mathcal{A}$ in this game is $\mathsf{Adv}_{\mathcal{A}} = \Pr[b' = b] - \frac{1}{2}$.

Definition 12. An functional encryption scheme is indistinguishability secure if for all poly-time $\mathcal{A}$ the function $\mathsf{Adv}_{\mathcal{A}}(\lambda)$ is negligible.

We also consider a weaker *selective* variant of security for indistinguishability. The selective game is equivalent to the adaptive one with the exception that the attacker must declare the challenge messages m_0, m_1 in an Init phase that occurs *before* it sees the public parameters.

Definition 13. An functional encryption scheme is *selectively* indistinguishability secure if for all poly-time $\mathcal{A}$ the advantage of $\mathcal{A}$ is negligible in the selective indistinguishability security game. .

Remarks and Transformations We make a few remarks about the above definitions. First, we observe that any system that is selectively secure can argued to be adaptively secure if we are willing to utilize complexity leveraging and loose a factor of $|\mathcal{M}|$ in the security reduction. Our construction in the next section will use the selective notion.

Second, our definition considers a single challenge ciphertext. However, using the usual hybrid methods it can be shown that any system indistinguishability secure under this definition is secure under a multiple message definition such as that used in [CJO+13].

6.1 Functional Encryption from Indistinguishability Obfuscation

We now give our construction and proof of security of a functional encryption system from an indistinguishability obfuscator ($i\mathcal{O}$). In addition to an $i\mathcal{O}$, our construction uses public key encryption and statistically simulation sound non-interactive zero knowledge proofs (SSS-NIZKs). Background material can be found in Appendix B. We let $(\mathsf{Setup}_{PKE}, \mathsf{Encrypt}_{PKE}, \mathsf{Eval}_{PKE}, \mathsf{Decrypt}_{PKE})$ be the algorithms comprising our (perfectly correct) encryption scheme. Our SSS-NIZK system will consist of algorithms $\mathsf{Setup}_{NIZK}, \mathsf{Prove}_{NIZK}, \mathsf{Verify}_{NIZK}$ and has a simulator Sim.. We build a functional encryption system for messages of length $n = n(\lambda)$. For messages of length n and security parameter λ the ciphertexts of our PKE scheme will be of length $\ell = \ell(\lambda, n)$. The construction is as follows:

- $\mathsf{Setup}_{FE}(1^\lambda)$: The Setup_{FE} algorithm takes the security parameter λ and computes the following.

1. Generate $(\mathsf{PK}^1_{PKE}, \mathsf{SK}^1_{PKE}) \leftarrow \mathsf{Setup}_{PKE}(1^\lambda)$ and $(\mathsf{PK}^2_{PKE}, \mathsf{SK}^2_{PKE}) \leftarrow \mathsf{Setup}_{PKE}(1^\lambda)$.
2. Set CRS $\leftarrow \mathsf{Setup}_{NIZK}$.

It sets the public parameters and master secret key as

$$\mathrm{PP} = \{\mathsf{PK}^1_{PKE}, \mathsf{PK}^2_{PKE}, \mathrm{CRS}\} \quad \text{and} \quad \mathrm{MSK} = \{\mathsf{SK}^1_{PKE}\}.$$

- $\mathsf{KeyGen}_{FE}(\mathrm{MSK}, f)$: Output an obfuscation $\mathcal{O}_{\mathrm{P3}^{(f,\mathsf{SK}^1_{PKE},\mathrm{CRS})}}$ for the program $\mathrm{P3}^{(f,\mathsf{SK}^1_{PKE},\mathrm{CRS})}$ using the circuit size value equal to the value $\max\{|\mathrm{P3}^{(f,\mathsf{SK}^1_{PKE},\mathrm{CRS})}|, |\mathrm{P4}^{(f,\mathsf{SK}^2_{PKE},\mathrm{CRS})}|\}$. We output the secret key SK_f as the obfuscated program.
- $\mathsf{Encrypt}_{FE}(\mathrm{PP}, m \in \{0,1\}^n)$: Output $c := (e_1, e_2, \pi)$, where $e_1 = \mathsf{Encrypt}_{PKE}(\mathsf{PK}^1_{PKE}, m; r_1)$ and $e_2 = \mathsf{Encrypt}_{PKE}(\mathsf{PK}^2_{PKE}, m; r_2)$ and π is a NIZK proof of Equation 2.
- $\mathsf{Decrypt}_{FE}(\mathrm{SK}_f, c = (e_1, e_2, \pi))$: The decryption algorithm runs the obfuscated program SK_f on input (e_1, e_2, π) and outputs the answer.

Correctness Correctness follows immediately from the correctness of the $i\mathcal{O}$, PKE system, SSS-NIZK, and the description of the program template P3.

We describe the two program classes in the figures below.

P3

Given input(e_1, e_2, π), $\mathrm{P3}^{(f,\mathsf{SK}^1_{PKE},\mathrm{CRS})}$ proceeds as follows:

1. Check that π is valid NIZK proof (using the Verify_{NIZK} algorithm and CRS) for the NP-statement

$$\exists m, r_1, r_2 : \Big(e_1 = \mathsf{Encrypt}_{PKE}(\mathsf{PK}^1_{PKE}, m; r_1) \bigwedge e_2 = \mathsf{Encrypt}_{PKE}(\mathsf{PK}^2_{PKE}, m; r_2)\Big) \tag{2}$$

2. If any checks fail output 0; otherwise output $f\big(\mathsf{Decrypt}_{PKE}(\mathsf{SK}^1_{PKE}, e_1)\big)$.

Figure 3:

P4

Given input(e_1, e_2, π), $\mathrm{P4}^{(f,\mathsf{SK}^2_{PKE},\mathrm{CRS})}$ proceeds as follows:

1. Check that π is valid NIZK proof (using the Verify_{NIZK} algorithm and CRS) for the NP-statement

$$\exists m, r_1, r_2 : \Big(e_1 = \mathsf{Encrypt}_{PKE}(\mathsf{PK}^1_{PKE}, m; r_1) \bigwedge e_2 = \mathsf{Encrypt}_{PKE}(\mathsf{PK}^2_{PKE}, m; r_2)\Big)$$

2. If any checks fail output 0; otherwise output $f\big(\mathsf{Decrypt}_{PKE}(\mathsf{SK}^2_{PKE}, e_2)\big)$.

Figure 4:

6.2 Proof of Security

We now prove that no poly-time attacker can break our system if our underlying assumptions hold. We prove selective security for the functional encryption indistinguishability game. Any poly-time attack algorithm $\mathcal{A}$ will make a maximum of $q = q(\lambda)$ private key queries. For simplicity we assume that an algorithm will always make exactly q queries. We denote f_i for $i \in [q]$ to be the i-th function queried in the indistinguishability game. By the rules of the game $f_i(m_0)$ is constrained to be equal to $f_i(m_1)$.

We organize our proof into a sequence of hybrids. In the first hybrid the challenger encrypts m_0. We then gradually change the encryption in multiple hybrid steps (via a two-key type proof) into an encryption of m_1. We show that each successive hybrid experiment is indistinguishable from the last, thus showing our obfuscator to have indistinguishability security.

Sequence of Hybrids

- Hyb_0: This hybrid corresponds to the honest execution the selective indistinguishability game given in Section 6 where the challenger encrypts m_0 in the challenge ciphertext.

- Hyb_1 This hybrid is identical to Hyb_0 with the exception that (CRS, π^*) is simulated as

  $$(\text{CRS}, \pi^*) \leftarrow \mathsf{Sim}(1^\lambda, \exists m, r_1, r_2 : e_1^* = \mathsf{Encrypt}_{PKE}(\mathtt{PK}^1_{PKE}, m; r_1) \bigwedge e_2^* = \mathsf{Encrypt}_{PKE}(\mathtt{PK}^2_{PKE}, m; r_2)).$$

  where e_1^*, e_2^* is part of the challenge ciphertext. Note, that in the selective security game the challenge ciphertext can be given out simultaneously with the public parameters.

- Hyb_2 This hybrid is identical to the last hybrid, Hyb_1, with the exception that the challenge ciphertext is generated as $e_1^* = \mathsf{Encrypt}_{PKE}(\mathtt{PK}^1_{PKE}, m_0; r_1)$ and $e_2^* = \mathsf{Encrypt}_{PKE}(\mathtt{PK}^2_{PKE}, m_1; r_2)$. (The NIZK is still simulated.)

- $\mathsf{Hyb}_{3,i}$ for $i \in [0, q]$: In this set of hybrids we change the form of the secret keys used to decrypt the ciphertext. In $\mathsf{Hyb}_{3,i}$ the first i private keys requested will result in private keys generated as obfuscations of the program $\text{P4}^{(f_i, \mathsf{SK}^2_{PKE}, \text{CRS})}$. The remaining $i+1$ to q private keys are generated using the program $\text{P3}^{(f_i, \mathsf{SK}^1_{PKE}, \text{CRS})}$ as in Hyb_2 . We observe that $\mathsf{Hyb}_{3,0}$ is equivalent to Hyb_2.

- Hyb_4 This hybrid is identical to the hybrid $\mathsf{Hyb}_{3,q}$ with the exception that the challenge ciphertext is generated as $e_1^* = \mathsf{Encrypt}_{PKE}(\mathtt{PK}^1_{PKE}, m_1; r_1)$ and $e_2^* = \mathsf{Encrypt}_{PKE}(\mathtt{PK}^2_{PKE}, m_1; r_2)$. (The NIZK is still simulated and keys all created from $\text{P4}^{(f, \mathsf{SK}^2_{PKE}, \text{CRS})}$.)

- $\mathsf{Hyb}_{5,i}$ for $i \in [0, q]$: The challenge ciphertext and CRS is formed the same as in Hyb_2. In this set of hybrids we change the form of the secret keys used to decrypt the ciphertext. In $\mathsf{Hyb}_{5,i}$ the first i private keys requested will result in private keys generated as obfuscations of the program $\text{P3}^{(f_i, \mathsf{SK}^1_{PKE}, \text{CRS})}$. The remaining $i+1$ to q private keys are generated using the program $\text{P4}^{(f_i, \mathsf{SK}^2_{PKE}, \text{CRS})}$ as in Hyb_4 . We observe that $\mathsf{Hyb}_{5,0}$ is equivalent to Hyb_4.

- Hyb_6: This hybrid is the same as $\mathsf{Hyb}_{5,q}$ with the exception that the CRS is generated from an honest run of the Setup_{NIZK} algorithm and that the NIZK proof component, π^*, of the challenge ciphertext is generated from the witness (r_1, r_2). This corresponds to the security game when message m_1 is encrypted for the challenge ciphertext.

Proofs of Hybrid Arguments

Claim 5. If our SSS-NIZK system is computationally zero knowledge, then no poly-time attacker can distinguish with non-negligible probability between Hyb_0 and Hyb_1.

Proof. We show that if there is a poly-time attacker $\mathcal{A}$ that can distinguish between Hyb_0 and Hyb_1 with non-negligible advantage, then there is a poly-time algorithm $\mathcal{B}$ that breaks the zero knowledge security of our NIZK scheme. $\mathcal{B}$ begins by running $\mathcal{A}$ and receiving m_0, m_1.

$\mathcal{B}$ generates both public keys itself (keeping the first secret key) and encrypts $e_1^* = \mathsf{Encrypt}_{PKE}(\mathtt{PK}^1_{PKE}, m_0; r_1)$ and $e_2^* = \mathsf{Encrypt}_{PKE}$ $(\mathtt{PK}^2_{PKE}, m_0; r_2)$. It then submits to the challenger the statement

$$x' = \exists m, r_1, r_2 : e_1^* = \mathsf{Encrypt}_{PKE}(\mathtt{PK}^1_{PKE}, m; r_1) \bigwedge e_2^* = \mathsf{Encrypt}_{PKE}(\mathtt{PK}^2_{PKE}, m; r_2)$$

as well as the witness (m_0, r_1, r_2). It receives back (CRS', π'). It uses $\mathrm{CRS} = \mathrm{CRS}'$ to make the public parameters and sets the challenge ciphertext as $(e_1^*, e_2^*, \pi^* = \pi')$. The attacker, $\mathcal{A}$, then makes q private key queries to $\mathcal{B}$. All queries for a function f are answered by using SK^1_{PKE} to make $\mathrm{P3}^{(f, \mathsf{SK}^1_{PKE}, \mathrm{CRS})}$.

If the zero knowledge challenger used the honest setup algorithm and prover to generate CRS', π' , then we are exactly in hybrid Hyb_0; if it simulated the proof, then we are in hybrid Hyb_1. Therefore, if an attacker can distinguish between the two hybrids with non-negligible advantage, it will break the zero knowledge property of the NIZK scheme.

■

Claim 6. If our PKE system is IND-CPA secure, then no poly-time attacker can distinguish with non-negligible probability between Hyb_1 and Hyb_2.

Proof. We show that if there is a poly-time attacker $\mathcal{A}$ that can distinguish between Hyb_1 and Hyb_2 with non-negligible advantage, then there is a poly-time algorithm $\mathcal{B}$ that breaks the IND-CPA security of our PKE scheme. $\mathcal{B}$ begins by running $\mathcal{A}$ and receiving m_0, m_1.

$\mathcal{B}$ first generates the first public key itself (keeping the secret key) and encrypts $e_1^* = \mathsf{Encrypt}_{PKE}(\mathsf{PK}^1_{PKE}, m_0; r_1)$. It then receives a public key from the challenger and designates it as PK^2_{PKE}. Next, it submits m_0, m_1 to the challenger and receives back e'. It sets $e_2^* = e'$. Finally, it uses the simulation algorithm to get:

$$(\mathrm{CRS}, \pi^*) = \mathsf{Sim}(1^\lambda, \exists m, r_1, r_2 : e_1^* = \mathsf{Encrypt}_{PKE}(\mathsf{PK}^1_{PKE}, m; r_1) \bigwedge e_2^* = \mathsf{Encrypt}_{PKE}(\mathsf{PK}^2_{PKE}, m; r_2)).$$

The parameters and challenge ciphertext are now given out as $\mathrm{PP} = \{\mathsf{PK}^1_{PKE}, \mathsf{PK}^2_{PKE}, \mathrm{CRS}\}$ and (e_1^*, e_2^*, π^*). The attacker, $\mathcal{A}$, then makes q private key queries to $\mathcal{B}$. All queries for a function f are answered by using SK^1_{PKE} to make $\mathrm{P3}^{(f, \mathsf{SK}^1_{PKE}, \mathrm{CRS})}$.

If the IND-CPA challenger gave an encryption of m_0, then we are exactly in hybrid Hyb_0; if it gave e' as an encryption of m_1, then we are in hybrid Hyb_1. Therefore, if an attacker can distinguish between the two hybrids with non-negligible advantage, it will break the IND-CPA property of the PKE scheme.

■

Claim 7. If the Family-$i\mathcal{O}$ assumption holds for our indistinguishability obfuscator, then no poly-time attacker can distinguish between $\mathsf{Hyb}_{3,i}$ and $\mathsf{Hyb}_{3,i+1}$ for $i \in [0, q-1]$.

Proof. We show that if there is a poly-time attacker $\mathcal{A}$ that that can distinguish between $\mathsf{Hyb}_{3,i}$ and $\mathsf{Hyb}_{3,i+1}$ with non-negligible advantage, then there is a poly-time algorithm $\mathcal{B}$ that breaks the Family-$i\mathcal{O}$ security of our obfuscation scheme. $\mathcal{B}$ begins by running $\mathcal{A}$ and receiving m_0, m_1.

$\mathcal{B}$ generates both public keys (keeping both secret keys) and encrypts $e_1^* = \mathsf{Encrypt}_{PKE}(\mathsf{PK}^1_{PKE}, m_0; r_1)$ and $e_2^* = \mathsf{Encrypt}_{PKE}\ (\mathsf{PK}^2_{PKE}, m_1; r_2)$. it uses the simulation algorithm to get:

$$(\mathrm{CRS}, \pi^*) = \mathsf{Sim}(1^\lambda, \exists m, r_1, r_2 : e_1^* = \mathsf{Encrypt}_{PKE}(\mathsf{PK}^1_{PKE}, m; r_1) \bigwedge e_2^* = \mathsf{Encrypt}_{PKE}(\mathsf{PK}^2_{PKE}, m; r_2)).$$

The parameters and challenge ciphertext are now given out as $\mathrm{PP} = \{\mathsf{PK}^1_{PKE}, \mathsf{PK}^2_{PKE}, \mathrm{CRS}\}$ and (e_1^*, e_2^*, π^*).

The attacker, $\mathcal{A}$, then makes q private key queries to $\mathcal{B}$. For $j \le i$ the j-th private key is created as an obfuscation of the program $\mathrm{P4}^{(f_j, \mathsf{SK}^2_{PKE}, \mathrm{CRS})}$. For $j > i+1$ the j-th private key is created as an obfuscation of the program $\mathrm{P3}^{(f_j, \mathsf{SK}^1_{PKE}, \mathrm{CRS})}$. For the $i+1$-th private key query $\mathcal{B}$ submits $C_0 = \mathrm{P3}^{(f_{i+1}, \mathsf{SK}^1_{PKE}, \mathrm{CRS})}$ and $C_1 = \mathrm{P4}^{(f_{i+1}, \mathsf{SK}^2_{PKE}, \mathrm{CRS})}$ to the Family-$i\mathcal{O}$ challenger. and receives back an obfuscated circuit C', which $\mathcal{B}$ gives as the $i+1$-th private key.

We now check that we are using a valid instance of the Family-$i\mathcal{O}$ assumption in showing that both programs have the same output on each input. We break these down by cases on the input. We first consider inputs (e_1, e_2, π) where e_1, e_2 are valid encryptions of the same message and π is a proof of this that passes Verify_{NIZK}. For these inputs both circuits will proceed to the second step where they decrypt the same message,m, no matter what key they use and compute the same function f_{i+1}. Thus the output is the same on all inputs of this class. The second sets of inputs we consider are where the verification check of equation 2 in step 1 does not pass. In this case, both circuits output 0. Finally, we can consider cases where the verification check passes, but e_1, e_2 are *not* valid encryptions of the same message. Due to the statistical simulation soundness property of the SSS-NIZK this can only happen if $e_1 = e_1^*$ and $e_2 = e_2^*$. In this case decrypting e_1^* gives m_0 and decrypting $e_2^* = m_1$. However, the first circuit outputs $f_{i+1}(m_0)$ which is must be equal to $f_{i+1}(m_1)$ (by the rules of the game), which is the output of the second circuit. Since the two circuits have the same output on all inputs, this constitutes a valid instance of the assumption.

If the Family-$i\mathcal{O}$ challenger chose the first program, then we are exactly in hybrid $\mathsf{Hyb}_{3,i}$; if it chose the second, then we are in $\mathsf{Hyb}_{3,i+1}$. Therefore, if an attacker can distinguish between the two hybrids with non-negligible advantage, it will break the Family-$i\mathcal{O}$ assumption.

■

Claim 8. If our PKE system is IND-CPA secure the, then no poly-time attacker can distinguish with non-negligible probability between $\mathsf{Hyb}_{3,q}$ and Hyb_4.

The proof of this claim follows in a directly analogous way to that of Claim 6.

Claim 9. If the Family-$i\mathcal{O}$ assumption holds for our indistinguishability obfuscator, then no poly-time attacker can distinguish between $\mathsf{Hyb}_{5,i}$ and $\mathsf{Hyb}_{5,i+1}$ for $i \in [0, q-1]$.

The proof of this claim follows in a directly analogous way to that of Claim 7.

Claim 10. If our SSS-NIZK system is computationally zero knowledge, then no poly-time attacker can distinguish with non-negligible probability between Hyb_0 and Hyb_1.

The proof of this claim follows in a directly analogous way to that of Claim 5.

Theorem 4. Under the assumptions listed above our FE system is selectively secure in the indistinguishability game of Section 6.

Proof. Our claims show a succession of a polynomial number of hybrid experiments where no poly-time attacker can distinguish one from the next with non-negligible advantage. The first hybrid Hyb_0 corresponds to encrypting a message m_0 in the selective functional encryption indistinguishability game and the last hybrid Hyb_6 corresponds to encrypting the message m_1. Security in the indistinguishability game follows.

■

Acknowledgements

We are grateful to Boaz Barak and Yuval Ishai for illuminating conversations.

References

[AFV11] Shweta Agrawal, David Mandell Freeman, and Vinod Vaikuntanathan. Functional encryption for inner product predicates from learning with errors. In *Proceedings of the 17th international conference on The Theory and Application of Cryptology and Information Security*, ASIACRYPT'11, 2011.

[AGVW12] Shweta Agrawal, Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption: New perspectives and lower bounds. Cryptology ePrint Archive, Report 2012/468, 2012.

[Bar86] D A Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc1. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, STOC '86, 1986.

[BCOP04] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *EUROCRYPT*, pages 506–522, 2004.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, 2001.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *STOC*, pages 103–112, 1988.

[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 *CRYPTO*, pages 1–18, 2001.

[BGI$^+$12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6, 2012.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, 2012.

[BOGG$^+$88] Michael Ben-Or, Oded Goldreich, Shafi Goldwasser, Johan Håstad, Joe Kilian, Silvio Micali, and Phillip Rogaway. Everything provable is provable in zero-knowledge. In *CRYPTO*, pages 37–56, 1988.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *CRYPTO*, pages 868–886, 2012.

[BSW06] Dan Boneh, Amit Sahai, and Brent Waters. Fully collusion resistant traitor tracing with short ciphertexts and private keys. In *EUROCRYPT*, pages 573–592, 2006.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: definitions and challenges. In *Proceedings of the 8th conference on Theory of cryptography*, TCC'11, pages 253–273, Berlin, Heidelberg, 2011. Springer-Verlag.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *CRYPTO*, 2011.

[BW07] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *Proceedings of the 4th conference on Theory of cryptography*, TCC'07, pages 535–554, Berlin, Heidelberg, 2007. Springer-Verlag.

[CFN94] Benny Chor, Amos Fiat, and Moni Naor. Tracing traitors. In *CRYPTO*, pages 257–270, 1994.

[CJO$^+$13] Angelo De Caro, Vincenzo Iovino Abhishek Jain, Adam O'Neill, Omer Paneth, and Giuseppe Persiano. On the achievability of simulation-based security for functional encryption. In *CRYPTO*, 2013.

[CLT13] Jean-Sebastien Coron, Tancrede Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. Cryptology ePrint Archive, Report 2013/183, 2013.

[CMNT11] Jean-Sébastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. In *CRYPTO*, pages 487–504, 2011.

[Coc01] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In *IMA Int. Conf.*, pages 360–363, 2001.

[FLS99] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs under general assumptions. *SIAM Journal of Computing*, 29(1):1–28, 1999.

[FS90] Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *STOC*, pages 416–426, 1990.

[Gen09a] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. `crypto.stanford.edu/craig`.

[Gen09b] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *EUROCRYPT*, 2013.

[GGH$^+$13b] Sanjam Garg, Craig Gentry, Shai Halevi, Amit Sahai, and Brent Waters. Attribute-based encryption for circuits from multilinear maps. Cryptology ePrint Archive, Report 2013/128, 2013. `http://eprint.iacr.org/`.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *STOC*, 2013.

[GK05] Shafi Goldwasser and Yael Tauman Kalai. On the impossibility of obfuscation with auxiliary input. In *Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium on*, pages 553–562. IEEE, 2005.

[GKP$^+$13] Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Succinct functional encryption and applications: Reusable garbled circuits and beyond. In *STOC*, 2013.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In Alfred V. Aho, editor, *STOC*, pages 218–229. ACM, 1987.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for np. In *Proceedings of Eurocrypt 2006, volume 4004 of LNCS*, pages 339–358. Springer, 2006.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, CCS '06, 2006.

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

[GR07] Shafi Goldwasser and Guy N. Rothblum. On best-possible obfuscation. In *Proceedings of the 4th conference on Theory of cryptography*, TCC'07, pages 194–213, 2007.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO*, 2013.

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

[GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In *STOC*, 2013.

[Ish13] Yuval Ishai. Personal communication, 2013.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In Janos Simon, editor, *STOC*, pages 20–31. ACM, 1988.

[KSW08] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *Proceedings of the theory and applications of cryptographic techniques 27th annual international conference on Advances in cryptology*, EUROCRYPT'08, 2008.

[LATV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *STOC*, pages 1219–1234, 2012.

[Nao91] Moni Naor. Bit commitment using pseudorandomness. *J. Cryptology*, 4(2):151–158, 1991.

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *STOC*, pages 427–437, 1990.

[O'N10] Adam O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010.

[PRV12] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *TCC*, pages 422–439, 2012.

[Sah99] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *FOCS*, pages 543–553, 1999.

[Sha85] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 47–53, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[SS10] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 463–472, New York, NY, USA, 2010. ACM.

[SS11] Damien Stehlé and Ron Steinfeld. Making ntru as secure as worst-case problems over ideal lattices. In *EUROCRYPT*, pages 27–47, 2011.

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.

[SW08] Amit Sahai and Brent Waters. Slides on functional encryption. PowerPoint presentation, 2008. `http://www.cs.utexas.edu/~bwaters/presentations/files/functional.ppt`.

[SWP00] Dawn Xiaoding Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*, pages 44–55. IEEE, 2000.

[vDGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *EUROCRYPT*, pages 24–43, 2010.

A Multilinear Jigsaw Puzzle implementation details

One instantiation of Multilinear Jigsaw Puzzles is obtained by taking the GGH multilinear encoding schemes and eliminating those aspects that are no longer needed, namely the encodings of 0,1 in the public parameters. (We note that we can also use the recent multilinear maps over the integers of Coron et al. [CLT13] as a basis for our Multilinear Jigsaw Puzzles.) We provide the GGH-based instantiation here.

Jigsaw generator. On input (k, λ, Π), the Jigsaw generator proceeds as follows.

Instance Generator. The Jigsaw generator first chooses a large random prime q, which will be the overall modulus used. She will use a dimension parameter m which will be an appropriate (large enough) power of two. Further work will be done in the ring $R = \mathcal{Z}[X]/(X^m+1)$, and the ring $R_q = \mathcal{Z}_q[X]/(X^m+1) = R/qR$. The size of these parameters will be discussed at the end of this section.

The first sequence of operations that the Jigsaw generator performs are to set up the approximate multilinear maps. She begins by choosing several *secret* polynomials:

- First, she chooses a "small" random polynomial $g \in R$ such that $|R/(g)|$ is a (large) prime p. Here, "small" refers to each coefficient in g being a small integer – we will describe exactly what "small" should mean when we later discuss parameter selection. This polynomial g must also satisfy a technical condition ensuring that g^{-1} when viewed as residing in $\mathbb{Q}[X]/(X^m + 1)$, is sufficiently small. We omit the proofs that these conditions can be satisfied with noticeable probability by a random choice of small g; these are found in [GGH13a].
- Then, she chooses k random polynomials $z_1, z_2, \dots, z_k \in R_q$. These polynomials are chosen uniformly, and therefore are very unlikely to be "small."

The Jigsaw Generator will identify the "plaintext space" with $\mathcal{Z}_p$. In other words, in the analogy to (true) multilinear maps, the underlying groups will be of prime order p. Furthermore, each "level" of the multilinear map will be associated with a subset of the set $[k] = \{1, 2, \dots, k\}$. Again, to make the analogy complete, we will think of having groups G_S, for every subset $S \subseteq [k]$, and there will be a bilinear map from $G_S \times G_{S'} \to G_{S\cup S'}$ for every disjoint pair of sets $S, S' \subset [1, k]$.

Encoding. Next, the Jigsaw Generator is ready to create the *puzzle pieces*, which are encodings of various elements of $\mathcal{Z}_p$ at different levels. To create an encoding $\mathsf{Encode}_S(a)$ of an element $a \in \mathcal{Z}_p$ at the level $S \subset [1, k]$, the Puzzle Maker does the following:

- First, she reduces a modulo (g) to create a small polynomial $\hat{a} \equiv a \pmod{g}$ in R.
- Then, she computes in R_q the following:

$$\mathsf{Encode}_S(a) = \frac{\hat{a} + e \cdot g}{\prod_{i \in S} z_i}$$

  where e is chosen so that $\hat{a} + e \cdot g$ is discrete Gaussian centered at the origin, and so that the resulting e is "small." Once again, we will discuss exactly what "small" means here when we discuss settings of parameters. We think of e as being a crucial noise term for achieving security. (We refer the reader to [GGH13a] for details on a [GPV08] based more sophisticated sampling procedure.)

The various puzzle pieces generated by the Jigsaw Generator will be given to the Jigsaw Verifier.

Zero Test Generation. The final step of the Jigsaw Generator is to create a "zero testing" parameter that will allow the Jigsaw Verifier to know when he has solved the puzzle by creating a nontrivial encoding of 0 at the highest level $\{1, 2, \dots, k\}$. This will be used to allow the Jigsaw verifier to check if two elements create at the highest level are equal to each other, by subtracting and then using the zero test.

The Jigsaw Generator now creates the zero-testing element by performing the following computation in R_q:

$$\mathbf{p}_{zt} = \frac{h \cdot \prod_{i=1}^{k} z_i}{g}$$

Here, h is a random "mid-size" polynomial chosen from a discrete Gaussian in R, whose coefficients are of size roughly $q^{2/3}$. Intuitively, since an encoding of 0 at level $\{1, 2, \dots, k\}$ is $e \cdot g / \prod_i z_i$, by multiplying it by $\mathbf{p}_{zt}$, one is left with $h \cdot e$ which will be small. By testing if this product is small, the Puzzle Solver will know whether it has successfully created an encoding of 0 at the final level. The proof of this fact is omitted and can be found in [GGH13a].

When she is done, she sends $(\mathsf{prms}, \mathbf{p}_{zt})$ and the various puzzle pieces (a.k.a. encodings) she has created to the Jigsaw Verifier. The Jigsaw Generator plays no further role.

Jigsaw verifier. The verifier is given the system parameters $\mathsf{prms} = (k, m, q, p, \mathbf{p}_{zt})$, a valid form Π, and inputs for Π $(I_1, u_1), (I_2, u_2), \ldots, (I_n, u_n)$. If the output wire is not associated with $[k]$ then the verifier outputs 0. If it is associated with $[k]$ then the verifier just evaluates Π on these inputs using R_q operations, then use the zero-test parameter to check if the output is an encoding of zero. Namely every $, \ominus, \oplus, \otimes$ gate in the form Π is replaced by the operations $-, +, \times$ in R_q, respectively, applied to the input elements to this gate (explained below). Denote the element at the output wire by u, the properties of the encoding scheme ensure that it is of the form

$$u = \frac{a^*}{\prod_{i \in [k]} z_i} \in R_q,$$

where $\|a^*\| \ll \sqrt{q}$, and $a^* \in (g)$ if and only if $u \in S^{(0)}_{[k]}$. The verifier multiplies the output encoding by the zero-test parameter to get $w = u \cdot \mathbf{p}_{zt} \in R_q$, and output 1 if $\|w\| \ll q$ and 0 otherwise.

Now we give more details on the the algorithm used by the Jigsaw Verifier.

Addition at same level. Given two encodings $\alpha = \mathsf{Encode}_S(a)$ and $\beta = \mathsf{Encode}_S(b)$ at the same level S, observe the following computation over R_q:

$$\alpha + \beta = \frac{(\hat{a} + \hat{b}) + e' \cdot g}{\prod_{i \in S} z_i}$$

We will choose parameters so that any polynomial number of additions will keep the term e' from growing too large. Thus, we see that addition yields another valid encoding at the same level.

Multiplication, jumping levels. Given two encodings $\alpha = \mathsf{Encode}_S(a)$ and $\beta = \mathsf{Encode}_T(b)$ at disjoint level sets $S, T \subset [1, k]$, observe the following computation over R_q:

$$\alpha \cdot \beta = \frac{(\hat{a} \cdot \hat{b}) + e' \cdot g}{\prod_{i \in S \cup T} z_i}$$

We will choose parameters so that the term e' will not grow too large as long as the level sets S and T are disjoint. Thus, we see that multiplication yields another valid encoding at a different level.

The encodings are designed so that only these operations are "meaningful." Other operations, such as addition across different levels, or division of encodings, will produce "junk." Similarly, multiplication so that the final level contains any element with multiplicity more than one will produce an encoding that essentially cannot be distinguished from a random. See [GGH13a] for further details.

Finally, the Jigsaw verifier will yield some concrete encoded value u at the highest level $[1, k]$ at which point it will then invoke the zero testing procedure:

Zero-testing. The procedure $\mathsf{isZero}(\mathsf{prms}, \mathbf{p}_{zt}, u)$ just multiplies $v = u \cdot \mathbf{p}_{zt}$ in R_q, and tests if v is small enough (e.g., if it has canonical embedding of Euclidean norm smaller than $q^{7/8}$).

If the zero test passes, the Jigsaw Verifier outputs 1, otherwise it outputs 0.

Setting Parameters. If we think of the dimensionality m as being an effective security parameter, then to achieve the level of security and functionality that we need, we can set $k = m^{\delta}$, and q of size approximately $2^{O(m^{\epsilon})}$, where δ and ϵ are suitably chosen constants $0 < \delta < \epsilon < 1$. The definition of "small" for the size of g and the noise term e above will be that each coefficient should be smaller than $2^{O(m^{\delta})}$.

B Additional Background

In this section we first give background on three cryptographic primitives used in Section 6.1: Non-Interactive and Perfectly Binding Commitments, Statistical Simulation-Sound Non-Interactive Zero-Knowledge Proofs, and Fully Homomorphic Encryption. In addition, we describe how any proofs that are verifiable in by a family of polynomial sized ciruits can be transformed into a proof that is verifiable by a family of circuits belonging to $\mathbf{NC}^1$.

B.1 Non-Interactive and Perfectly Binding Commitments

We let $\mathsf{Com}(\cdot;\cdot)$ denote the commitment function of a non-interactive commitment scheme. Com takes as input a bit $b \in \{0,1\}$ and randomness $r \in \{0,1\}^\lambda$ and outputs a commitment $c \leftarrow \mathsf{Com}(b;r)$. Commitment schemes must satisfy two properties: hiding and binding.

Computational Hiding. Hiding means that no computationally bounded adversary can distinguish as to which bit is locked in the commitment. Let $\mathcal{A}$ be any non-uniform adversary running in time $\mathsf{poly}(\lambda)$. We say that the commitment scheme is computationally hiding if:

$$\Pr\left[b = b' \;\middle|\; \begin{array}{l} b \leftarrow \{0,1\}; \\ c = \mathsf{Com}(b;r); b' \leftarrow \mathcal{A}(c) \end{array} \right] = \mathrm{negl}(\lambda) \;.$$

Perfectly Binding. Intuitively speaking, binding requires that no (even unbounded) adversary can open the commitment in two different ways. Here, we define the strongest variant known as *perfectly binding*. Formally we require that, there does not exist values (r_0, r_1) such that $\mathsf{Com}(0;r_0) = \mathsf{Com}(1;r_1)$.

For simplicity of exposition, in the sequel, we will assume that random coins are an implicit input to the commitment function. We will also naturally extend the commitment function Com to commit to strings instead of just bits.

For simplicity of exposition, in the presentation of our results in this manuscript, we use a non-interactive perfectly binding string commitment which can be based on any 1-to-1 one-way function. It is easy to see that all the constructions in this work can in fact be built using Naor's [Nao91] 2-round commitment scheme which can be based on any one-way function.

B.2 Statistical Simulation-Sound Non-Interactive Zero-Knowledge Proofs

Let R be an efficiently computable binary relation. For pairs $(x,w) \in R$ we call x the statement and w the witness. Let L be the language consisting of statements in R.

A non-interactive proof system [BFM88, FLS99, GOS06] for a relation R consists of a common reference string generation algorithm K, a prover P and a verifier V. We require that they all be probabilistic polynomial time algorithms, *i.e.*, we are looking at *efficient prover* proofs. The common reference string generation algorithm produces a common reference string σ of length $\Omega(\lambda)$. The prover takes as input (σ, x, w) and produces a proof π. The verifier takes as input (σ, x, π) and outputs 1 if the proof is acceptable and 0 if rejecting the proof. We call (K, P, V) a non-interactive proof system for R if it has the completeness and statistical-soundness properties described below.

Perfect completeness. A proof system is complete if an honest prover with a valid witness can convince an honest verifier. Formally we have

$$\Pr\Big[\sigma \leftarrow K(1^\lambda) : \exists (x,\pi) : x \notin L : V(\sigma, x, \pi) = 1\Big] = 1.$$

Statistical soundness. A proof system is sound if it is infeasible to convince an honest verifier when the statement is false. For all (even unbounded) adversaries $\mathcal{A}$ we have

$$\Pr\Big[\sigma \leftarrow K(1^\lambda); (x,\pi) \leftarrow \mathcal{A}(\sigma) : V(\sigma, x, \pi) = 1 : x \notin L\Big] = \mathsf{negl}(\lambda).$$

Computational zero-knowledge [FLS99]. A proof system is computational zero-knowledge if the proofs do not reveal any information about the witnesses to a bounded adversary. We say a non-interactive proof (K, P, V) is computational zero-knowledge if there exists a polynomial time simulator $S = (S_1, S_2)$, where S_1 returns a simulated common reference string σ together with a simulation trapdoor τ that enables S_2 to simulate proofs without access to the witness. For all non-uniform polynomial time adversaries $\mathcal{A}$ we have for all $x \in L$

$$\Pr\Big[\sigma \leftarrow K(1^\lambda); \pi \leftarrow P(\sigma, x, w) : \mathcal{A}(x, \sigma, \pi) = 1\Big] \approx \Pr\Big[(\sigma, \tau) \leftarrow S_1(1^\lambda, x); \pi \leftarrow S_2(\sigma, \tau, x) : \mathcal{A}(x, \sigma, \pi) = 1\Big].$$

Statistical Simulation-Soundness (SSS). A proof system is said to be statistically simulation sound if it is infeasible to convince an honest verifier of a false statement even when the adversary itself is provided with a simulated proof. For all statements x and all (even unbounded) adversaries $\mathcal{A}$ we have

$$\Pr\Big[(\sigma, \tau) \leftarrow S_1(1^\lambda, x); \pi \leftarrow S_2(\sigma, \tau, x) : \exists (x', \pi') : x' \neq x : V(\sigma, x', \pi') = 1 : x' \notin L\Big] = \mathsf{negl}(\lambda).$$

Realizing statistical simulation soundness. Constructions of non-interactive zero-knowledge (NIZK) proof system [BFM88, FLS99, GOS06] are well known. Now we will describe how a NIZK proof system can be used to realize a SSS NIZK proof system. Formally, using a NIZK proof system (K, P, V) and a non-interactive commitment scheme $\mathsf{Com}(\cdot;\cdot)$ we will construct a NIZK proof (K', P', V') that is also statistically simulation-sound. Let ℓ be an upper bound on the length of the statements proven and let 0^ℓ represent a special statement that is never proven.

- $K'(1^\lambda)$: Generate $\sigma \leftarrow K(1^\lambda)$ and $c \leftarrow \mathsf{Com}(0^\ell; r)$ where r are the random coins and output the common random string as $\Sigma = (\sigma, c)$.
- $P'(\Sigma = (\sigma, c), x, w)$: Generate the proof π' as $P(\sigma, x', w)$ where x' is the following statement:

$$\exists w, r \text{ such that } (x, w) \in R \bigvee c = \mathsf{Com}(x; r). \tag{3}$$

- $V'(\Sigma, x, \pi')$: The verifier V' outputs $V(\sigma, x', \pi')$ where x' is the statement as in Equation 3.

Correctness. The correctness of the scheme (K', P', V') follows directly from the correctness of (K, P, V).

Zero-Knowledge. We will start by describing our simulator $S' = (S'_1, S'_2)$ assuming the simulator $S = (S_1, S_2)$ for (K, P, V).

$S'_1(1^\lambda, x)$ generates the common random string $\Sigma = (\sigma, c)$ where $\sigma \leftarrow K(1^\lambda)$ and $c \leftarrow \mathsf{Com}(x; r)$.

S_2 generates the proof for x' (from Equation 3) using the randomness r. More specifically, it just outputs $P(\sigma, x', r)$.

The zero-knowledge property of our simulator follows from the following simple hybrid argument.

- Hyb_0: This hybrid corresponds to the honest generation of the proof.
- Hyb_1: We start generating c as a commitment to x rather that 0^ℓ.

  Indistinguishability between hybrids Hyb_0 and Hyb_1 follows from the computational hiding property of the commitment scheme.
- Hyb_2: We start using the randomness used in generation of the commitment c for generating the proof rather than the real witness. In other words, we use the simulation strategy.

  Indistinguishability between hybrids Hyb_1 and Hyb_2 follows from the zero-knowledge property of (K, P, V).

Statistical Simulation-Soundness. SSS follows directly from the statistical soundness of the (K, P, V) proof system and the fact that the only false statement for which an acceptable proof exists is x because the commitment c is perfectly binding.

B.3 Fully Homomorphic Encryption

Our definitions here follow [BGV12]. We start with a generic definition of homomorphic encryption that captures both fully and non-fully homomorphic schemes. A homomorphic encryption scheme HE is a tuple of PPT algorithms $(\mathsf{HE.KeyGen}, \mathsf{HE.Enc}, \mathsf{HE.Dec}, \mathsf{HE.Eval})$. The message space R_M of HE is some ring and our computational model will be arithmetic circuits over this ring (with addition and multiplication gates). $\mathsf{HE.KeyGen}$ takes the security parameter (and possibly other parameters of the scheme output by a Setup procedure) and outputs a secret key sk and a public key pk. $\mathsf{HE.Enc}$ takes the public key pk a message μ and outputs a ciphertext c that encrypts μ. $\mathsf{HE.Dec}$ takes the secret key sk and a ciphertext c and outputs a message μ. $\mathsf{HE.Eval}$ takes the public key pk, an arithmetic circuit f over M, and ciphertexts $c_1, \ldots, c_\ell$, where ℓ is the number of inputs to f, and outputs a ciphertext c_f.

Definition 14. We say that a homomorphic encryption *correctly evaluates* a circuit family $\mathcal{F}$ if for all $f \in \mathcal{F}$ and for all $\mu_1, \ldots, \mu_\ell \in R_M$ it holds that if sk, pk were properly generated by $\mathsf{HE.KeyGen}$ with security parameter λ, and if $c_i = \mathsf{HE.Enc}_{\mathsf{pk}}(\mu_i)$ for all i, and $c_f = \mathsf{HE.Eval}_{\mathsf{pk}}(f, c_1, \ldots, c_\ell)$, then

$$\Pr[\mathsf{HE.Dec}_{\mathsf{sk}}(c_f) \neq f(\mu_1, \ldots, \mu_\ell)] = \mathsf{negl}(\lambda) \ ,$$

where the probability is taken over all the randomness in the experiment.

We say that the scheme *compactly evaluates* the family if in addition the running time of the decryption circuit only depends on λ and not on its input.

We use standard semantic security (security under chosen plaintext attack) as our security notion.

Definition 15. A homomorphic scheme is secure if any polynomial time adversary that first gets a properly generated pk, then specifies $\mu_0, \mu_1 \in R_M$ and finally gets $\mathsf{HE.Enc}_{\mathsf{pk}}(\mu_b)$ for a random b, cannot guess the value of b with probability $> 1/2 + \mathsf{negl}(\lambda)$.

In a *leveled* fully homomorphic scheme, the parameters of the scheme depend (polynomially) on the depth of the circuits that the scheme is capable of evaluating. In a *pure* fully homomorphic scheme, the parameters do not have this dependence on the depth.

Definition 16 (Leveled FHE [Gen09a])**.** We say that a family of homomorphic encryption schemes $\{\mathsf{HE}^{(L)} : L \in \mathcal{Z}^+\}$ is leveled fully homomorphic if, for all $L \in \mathcal{Z}^+$, they all use the same decryption circuit, $\mathsf{HE}^{(L)}$ compactly evaluates all circuits of depth at most L, and the computational complexity of $\mathsf{HE}^{(L)}$'s algorithms is polynomial (the same polynomial for all L) in the security parameter, L, and (in the case of $\mathsf{HE}^{(L)}.\mathsf{Eval}$) the size of the circuit.

Definition 17. A *pure* FHE scheme is a leveled FHE where the complexity of $\mathsf{HE}^{(L)}$'s algorithms is independent of L, except insofar as the complexity of $\mathsf{HE}^{(L)}.\mathsf{Eval}$ depends on the size of the circuit.

Often decryption can be decomposed into a two step process, $\mathsf{DecPreprocess}$ and Dec, where $\mathsf{DecPreprocess}$ is a completely public algorithm that does not use sk or any secret information. Conceptually, one may prefer to view $\mathsf{DecPreprocess}$ as more properly belonging to Eval. Definition 16 refers only to the circuit for Dec, and it is fine if $\mathsf{DecPreprocess}$ depends on L.

Gentry's bootstrapping theorem allows one to transform schemes that enjoy some level of homomorphism (bootstrappable schemes) into FHE schemes. Let us first define what a bootstrappable scheme is:

Definition 18. We say that a homomorphic encryption scheme HE is bootstrappable if HE compactly evaluates all circuits of depth at most $(D + 1)$, where D is the depth of HE's decryption circuit, and the computational complexity of HE's algorithms is polynomial in the security parameter and (in the case of the evaluation algorithm) the size of the circuit.

In Definition 18, the depth of the decryption circuit is measured when the secret key is treated as the input, and the decrypted ciphertext is treated as constant.

We can now state the bootstrapping theorem:

Theorem 5 (Bootstrapping [Gen09b, Gen09a])**.** If there exists a bootstrappable encryption scheme HE, then there also exists a leveled FHE scheme $\{\mathsf{HE}^{(L)}\}$ with related security. There also exists a pure FHE scheme FHE with related security with an additional assumption of circular security (that semantic security holds even if the attacker is provided with a (bit-wise) encryption of sk under pk). The decryption circuit of $\{\mathsf{HE}^{(L)}\}$ and FHE is the same as the decryption circuit of HE.

Gentry also showed that semantically secure bootstrappable encryption schemes exist, assuming the hardness of certain lattice problems. More recently, bootstrappable encryption schemes have been constructed based on the approximate-gcd problem [vDGHV10, CMNT11], the learning with errors (LWE) problem [Bra12], the ring learning with errors (RLWE) problem [BV11], and the NTRU problem [SS11, LATV12].

Theorem 5 establishes that there are FHE schemes whose decryption circuit is the same as the decryption circuit of an underlying bootstrappable encryption scheme, which has size polynomial in the security parameter. However, we will require something stronger – namely, that the decryption circuit has depth only logarithmic in the security parameter (is in NC^1). Fortunately, essentially all known bootstrappable encryption schemes already have this property, although it has not often been proven explicitly. However, Brakerski, Gentry and Vaikuntanathan (BGV) [BGV12] did prove such a claim explicitly, as it was essential to their basing security on LWE for $\lambda^{O(\log \lambda)}$ approximation factors. Specifically, they provide the following lemma regarding their decryption circuit. (Note that the modulus q in their system can be made $\mathsf{poly}(\lambda)$ through a technique called "modulus reduction".)

Lemma 1 ([BGV12])**.** Let $n = O(\lambda)$, $q = \mathsf{poly}(\lambda)$ and let the ciphertext $\mathbf{c} \in \mathcal{Z}_q^n$. Consider the function $D(\mathbf{s}) = [[\langle \mathbf{c}, \mathbf{s} \rangle]_q]_2$, for input secret key $\mathbf{s} \in \mathcal{Z}_q^n$, where the coefficients of $\mathbf{s}$ are represented using standard binary representation. Then $D(\cdot)$ can be computed by a circuit of size $\tilde{O}(\lambda)$ and depth $O(\log \lambda)$.

Beginning with BGV, there are also leveled FHE schemes that do not use bootstrapping. In all such schemes that we know of, Dec is in NC^1.

B.4 Low Depth Proofs

Let R be an efficiently computable binary relation. For pairs $(x, w) \in R$ we call x the statement and w the witness. Let L be the language consisting of statements in R. As defined in Section B.2, a non-interactive proof with perfect completeness and perfect soundness for a relation R consists of an (efficient) prover P and a verifier V such that:

Perfect completeness. A proof system is perfectly complete if an honest prover with a valid witness can *always* convince an honest verifier. For all $(x, w) \in R$ we have

$$\Pr\Big[\pi \leftarrow P(x, w) : V(x, \pi) = 1\Big] = 1.$$

Perfect soundness. A proof system is perfectly sound if it is infeasible to convince an honest verifier when the statement is false. For all $x \notin L$ and all (even unbounded) adversaries $\mathcal{A}$ we have

$$\Pr\Big[\pi \leftarrow \mathcal{A}(x) : V(x, \pi) = 1\Big] = 0.$$

Furthermore we say that a non-interactive proof is *low-depth*, if the verifier V can be implemented in $\mathbf{NC}^1$.

We next sketch a very simple construction of a low-depth non-interactive proof. The prover P executes the NP-verification circuit on the witness and generates the proof as the sequential concatenation (in some specified order) of the bit values assigned to the individual wires of the circuit. The verifier V proceeds by checking consistency of the values assigned to the internal wires of the circuit for each gate. In particular for each gate in the NP-verification circuit the verifier checks if the wire vales provided in the proof represent a correct evaluation of the gate. Since the verification corresponding to each gate can be done independent of every other gate and in constant depth, we have that V itself is constant depth.

C Security of our Assumption

In this section we provide evidence for our assumption, demonstrating that our hardness assumption (Assumption 1) does not fall prey to a class of "simple attacks." Below we define a generic model that we term the *generic colored matrix model*, capturing attacks where the adversary only multiplies matrices in the correct order, and prove that Assumption 1 holds (unconditionally) in this model. Later in Section C.4 we discuss why we think that the generic colored matrix model captures the "most natural" attacks against our scheme.

C.1 Generic Colored Matrix Model

Roughly, the generic colored matrix model considers attacks in which the adversary is provided with matrices in some specific order, and is only allowed to add/multiply them in ways that respect this order (and to check for equality). The order is specified by assigning each matrix a left color and a right color, permitting only addition of matrices of matching colors, and only multiplication of matrices where the right color of one matches the left color of the other.

In more detail, a matrix in this model is associated with left and right colors LC, RC and an opaque handle h. The colors are just arbitrary strings, and below we use simple indexes $1, 2, 3, \ldots$ for the handles, i.e. the first matrix that the adversary sees has handle 1, the second has handle 2, etc. The adversary in this model is given the prime number p that defines the field $\mathbb{Z}_p$, and for every matrix $A \in \mathbb{Z}_p^{m\times n}$ it is given the tuple $((m, LC), (n, RC), h)$ but not the matrix A itself. The adversary is interacting with a stateful representation oracle that keeps track of the correspondence between matrices and their handles and let the adversary perform generic computation on these matrices. Given the prime p, the representation oracle chooses an initial set of ℓ colored matrices and assign to them the handles $1, \ldots, \ell$, inserts into its database the records $\{(A_i, (m_i, LC_i), (n_i, RC_i), i)\}_{i=1}^{\ell}$, sends to the adversary the prime p and representations $\{((m_i, LC_i), (n_i, RC_i), i)\}_{i=1}^{\ell}$ (without the matrices themselves), and then processes queries from the adversary as follows:

Addition. When the adversary makes a query $\mathsf{add}(h, h')$, the representation oracle looks up in the database the records corresponding to the two handles h, h'. If such records exist, let us denote them by $(A, (m, LC), (n, RC), h)$ and $(A', (m', LC'), (n', RC'), h')$. If they have the same dimensions and colors — $(m, LC) = (m', LC')$ and $(n, RC) = (n', RC')$ — then the representation oracle computes their modular sum $A^+ = A + A' \bmod p$. If the database already contains the matrix A^+ with dimensions and colors $(m, LC), (n, RC)$ then the oracle returns the handle of that matrix. Otherwise the oracle assigns to this matrix the next available handle index h^+, inserts into the database a record for $(A^+, (m, LC), (n, RC), h^+)$, and returns h^+ to the adversary.

Multiplication by constant. When the adversary makes a query $\mathsf{ConstMul}(h, a)$ (with h a handle and $a \in \mathbb{Z}_p$), the representation oracle looks up in the database the record corresponding to h. If it exists, let us denote it by $(A, (m, LC), (n, RC), h)$, and the representation oracle computes the modular product $A' = a \cdot A \bmod p$. If the database already contains the matrix A' with dimensions and colors $(m, LC), (n, RC)$ then the oracle returns the handle of that matrix. Otherwise the oracle assigns to this matrix the next available handle index h', inserts into the database a record for $(A', (m, LC), (n, RC), h')$, and returns h' to the adversary.

Multiplication. When the adversary makes a query $\mathsf{mult}(h, h')$, the representation oracle looks up in the database the records corresponding to the two handles h, h'. If such records exist, let us denote them by $(A, (m, LC), (n, RC), h)$ and $(A, , (m', LC'), (n', RC'), h')$. If they have matching dimensions and colors — namely $(n, RC) = (m', LC')$ — then the representation oracle computes their modular product $A^\times = A \times A' \in \mathbb{Z}_p^{m \times n'}$. If the database already contains the matrix $A^\times$ with dimensions and colors $(m, LC), (n', RC')$ then the oracle returns the handle of that matrix. Otherwise the oracle assigns to this matrix the next available handle index $h^\times$, inserts into the database a record for $(A^\times, (m, LC), (n', RC'), h^\times)$, and returns $h^\times$ to the adversary.

C.2 Equivalent Program Indistinguishability in the Colored Matrix Model

We now proceed to show that Assumption 1 holds (unconditionally) in the generic Colored Matrix Model. Specifically, an adversary that gets t handles from the representation oracle has at most $O(t^2/p)$ advantage in distinguishing any two equivalent programs. Recall the structure of the randomized branching programs in our construction, as explained in Equation 1:

$$\mathcal{RND}(BP) = \left\{ \begin{array}{ll} \tilde{\mathbf{s}} = \mathbf{s}R_0^{-1},\ \tilde{\mathbf{t}} = R_n\mathbf{t}, & \tilde{\mathbf{s}}' = \mathbf{s}'(R_0')^{-1},\ \tilde{\mathbf{t}}' = R_n'\mathbf{t}' \\ \big\{\tilde{D}_{i,b} = R_{i-1}D_{i,b}R_i^{-1} : i \in [n], b \in \{0,1\}\big\}, & \big\{\tilde{D}'_{i,b} = R'_{i-1}D'_{i,b}(R'_i)^{-1} : i \in [n], b \in \{0,1\}\big\} \end{array} \right\}.$$

Our analysis in this generic model examines only adversary strategies that cancel out the randomizing matrices R_i, R_i', which means that these matrices must be multiplied in order. Hence, in our model we assign to the matrices $\tilde{D}_{i,b}$ left color i and right-color-$(i+1)$, and similarly the matrices $\tilde{D}'_{i,b}$ are assigned the distinct left color i' and right-color-$(i+1)'$. The row vectors $\tilde{\mathbf{s}}$ and $\tilde{\mathbf{s}'}$ both are assigned the left color 0, and they are assigned the right colors 1 and $1'$, respectively. Similarly, the column vectors $\tilde{\mathbf{t}}$, $\tilde{\mathbf{t}}'$ are assigned left colors $(n+1)$ and $(n+1)'$, respectively, and they are both assigned the right color $(n+2)$.

The representation oracle in our model, on input a prime p, branching program BP, and a partial input assignment (J, σ), chooses the matrices and vectors as above (using the procedure described in Section 4.1), then discards the matrices that are incompatible with the partial assignment (J, σ), and interacts with the adversary using the remaining matrices. Denoting $N = 2m + 5$, the initial database of the representation oracle consists of the following matrices and their representation:

$$DB\big(p, BP, (J, \sigma)\big) = \left\{ \begin{array}{ll} (\tilde{\mathbf{s}}, (1, LC = 0), (N, RC = 1), h = 1), & (\tilde{\mathbf{s}}', (1, LC = 0), (N, RC = 1'), h = 2), \\ (\tilde{\mathbf{t}}, (N, LC = n+1), (1, RC = n+2), h = 3), & (\tilde{\mathbf{t}}', (N, LC = (n+1)'), (1, RC = n+2), h = 4), \\ \multicolumn{2}{l}{\big\{(\tilde{D}_{i,b}, (N, LC = i), (N, RC = i+1), h = ..)\ :\ i \in I_J, b = \sigma(\mathsf{inp}(i))\big\},} \\ \multicolumn{2}{l}{\big\{(\tilde{D}_{i,b}, (N, LC = i), (N, RC = i+1), h = ..)\ :\ i \notin I_J, b \in \{0,1\}\big\},} \\ \multicolumn{2}{l}{\big\{(\tilde{D}'_{i,b}, (N, LC = i'), (N, RC = (i+1)'), h = ..)\ :\ i \in I_J, b = \sigma(\mathsf{inp}(i))\big\},} \\ \multicolumn{2}{l}{\big\{(\tilde{D}'_{i,b}, (N, LC = i'), (N, RC = (i+1)'), h = ..)\ :\ i \notin I_J, b \in \{0,1\}\big\},} \end{array} \right\}$$

(where the handles are just all the indexes $1, 2, \ldots, |DB|$). The adversary is given as input the representation without the matrices themselves, and then it interacts with the representation oracle as above. We denote the total number of handles that the representation oracle sends to the adversary by T (these include the initial $|DB|$ handles that are given to the adversary as input, and all the handles that are sent in reply to the adversary queries). The view of the adversary A in this interaction is denoted by $\mathsf{view}_A(p, BP, (J, \sigma), T)$. Staged in this model, our hardness assumption can now be stated as the following theorem:

Theorem 6. For any prime p, branching program BP and two functionally-equivalent partial input assignments (J, σ) and (J, σ'), and any generic-colored-matrix-model adversary A that receives at most T handles from the representation oracle, the statistical distance between the views $\mathsf{view}_A(p, BP, (J, \sigma), T)$ and $\mathsf{view}_A(p, BP, (J, \sigma'), T)$ is bounded below $O(nT^2/p)$.

Note that within the generic colored matrix model the above claim is unconditional.

C.3 Security Proof

To prove Theorem 6 we describe an information theoretic simulation strategy that can simulate the queries of the adversary without knowledge of σ, but just with oracle access to the function $BP|_\sigma$. Since the simulation relies just on $BP|_\sigma$ which are identical over the two choices of σ, σ', the simulation in the two settings must be identical, and since it is a good simulation in both cases the theorem follows.

Non-commutative formal polynomials. We introduce some useful notation for describing our simulation strategy. The simulation strategy must keep track of the computation that the adversary is performing over the representations that are given to it, and it does that using *non-commutative formal polynomials* (NCFPs). A non-commutative formal polynomial over the set of variables $V = \{v_1, v_2, \dots v_t\}$ is of the form $\sum_i c_i \cdot v_{i_1} \times v_{i_2} \times \dots v_{i_{t_i}}$, where $c_i \in \mathbb{Z}_p$ is a constant and $v_{i_1} \times v_{i_2} \times \dots v_{i_{t_i}}$ is the i^{th} monomial. The addition, subtraction and multiplication of any two NCFPs is also a NCFP.

Let $\mathbf{s}, \mathbf{t}, \mathbf{s}', \mathbf{t}', D_{i,b}, D'_{i,b}$ denote the variables corresponding to the representations that are given to the adversary at initialization, then any handle that the adversary gets during the interaction must correspond to a NCFP in these variables.

Simulation. The simulator gets as input the prime p, the length n of the branching program BP, and the set J of fixed input bits (but not the actual input assignment σ or σ'), and it has access to the function $BP|_\sigma = BP_{\sigma'}$. The simulator maintains a database $\mathcal{P}$ of NCFPs in the variables above, each stored in canonical sum-of-monomials form where the monomials are in lexicographic order, along with their representations. Initially the database consists of only the single-variable terms corresponding to $\mathbf{s}, \mathbf{t}, \mathbf{s}', \mathbf{t}', D_{i,b}, D'_{i,b}$, and the adversary is given the corresponding representation. Thereafter, the simulator interacts with the adversary, playing the role of the representation oracle as follows:

Addition. When the adversary makes a query $\mathsf{add}(h, h')$, the simulator checks its database to find the records corresponding to these two handles. If these records exist, we denote them by $(P, (LC, m), (n, RC), h)$ and $(P', (LC', m'), (n', RC'), h')$, where P, P' are NCFPs. If both records exist and they satisfy $(LC, m) = (LC', m'), (n, RC) = (n', RC')$, then the simulator computes the NCFP $P^+ = P + P'$ with scalar arithmetic modulo p. (Otherwise the simulator returns $\perp$). The simulator next checks if a record for $(P^+, (LC, m), (n, RC), h^+)$ exists in the database, and otherwise it inserts such record to the database, using the next available index for h^+. Finally the simulator returns h^+ to the adversary.

Multiplication by constant. When the adversary makes a query $\mathsf{ConstMul}(h, a)$ (with h a handle and $a \in \mathbb{Z}_p$), the simulator looks up in the database the record corresponding to h. If it exists, let us denote it by $(P, (m, LC), (n, RC), h)$, and the representation oracle computes the NCFP $P' = a \cdot P$. If the database does not already have a record for $(P', (LC, m), (n, RC), h')$ then the simulator inserts one, using the next available handle index h'. The simulator returns h' to the adversary.

Multiplication. When the adversary makes a query $\mathsf{mult}(h, h')$, the representation oracle looks up in the database the records corresponding to the two handles h, h'. If such records exist, let us denote them by $(P, (m, LC), (n, RC), h)$ and $(P', (m', LC'), (n', RC'), h')$. If they have matching dimensions and colors — namely $(n, RC) = (m', LC')$ — then the representation oracle computes their product $P^\times = P \times P'$. (If they do not exists or are not matching them the simulator returns $\perp$.)

If $LC = 0$ and $RC' = n + 2$ (which means that this is a "1×1 matrix" corresponding to products that includes the bookend vectors), then the simulator performs an additional step to ensure consistency. Note that because of the color consistency requirements, all of the monomials in $P^\times$ must be of the form either $c \cdot \tilde{\mathbf{s}} \times \prod_{i=1}^n \tilde{D}_{i,b_i} \times \tilde{\mathbf{t}}$ or $c \cdot \tilde{\mathbf{s}}' \times \prod_{i=1}^n \tilde{D}'_{i,b_i} \times \tilde{\mathbf{t}}'$ (for some bits $b_1, b_2, \dots, b_n$), but cannot have a mix of primed and non-primed variables. Moreover, for all $i \in I_J$ we must have $b_i = \sigma(\mathsf{inp}(i))$ (since for these steps we only give out one matrix in the primal program and one in the dummy program, corresponding to the bit $\sigma(\mathsf{inp}(i))$).

Considering the sequence of bits $\mathbf{b} = b_1, b_2, \ldots, b_n$, we say that this it is *consistent* with the program BP if there exists an input χ such that $b_i = \chi_{\mathsf{inp}(i)}$ for all i. (Moreover in this this case we sometime say that $\mathbf{b}$ is consistent with χ.) In other words, if we have two steps i, i' with $\mathsf{inp}(i) = \mathsf{inp}(i')$ but $b_i \neq b_{i'}$ then we say that $\mathbf{b}$ is *inconsistent* with the program BP, and otherwise it is consistent. The same terminology is extended to the monomials, a monomial $c \cdot \tilde{\mathbf{s}} \times \prod_{i=1}^{n} \tilde{D}_{i,b_i} \times \tilde{\mathbf{t}}$ or $c \cdot \tilde{\mathbf{s}}' \times \prod_{i=1}^{n} \tilde{D}'_{i,b_i} \times \tilde{\mathbf{t}}'$ is consistent (with χ) if $b_1, b_2, \ldots, b_n$ is consistent (with χ), and inconsistent otherwise.

If $LC = 0$ and $RC' = n + 2$, then the simulator goes over all the monomials in $P^{\times}$. If it finds any monomial that is (a) made up of non-primed variables ($c \cdot \tilde{\mathbf{s}} \times \prod_{i=1}^{n} \tilde{D}_{i,b_i} \times \tilde{\mathbf{t}}$), and (b) is consistent with some χ such that $BP(\chi) = 1$, then the simulator replaces that monomial by its primed counterpart $c \cdot \tilde{\mathbf{s}}' \times \prod_{i=1}^{n} \tilde{D}'_{i,b_i} \times \tilde{\mathbf{t}}'$. (If there is already a monomial for these primed variables, then it adds the two constants.) Note that the simulator can check the two conditions (a) and (b) above, since it knows the branching program BP and it has access to an oracle for $BP|_{\sigma}$,

Denote the resulting NCFP after all the substitutions (if any) by P^*. (If $LC \neq 0$ or $RC' \neq n + 2$ then no substitutions are made and we have $P^* = P^{\times}$). If the database does not already have a record for $(P^*, (LC, m), (n, RC'), h^*)$ then the simulator inserts one, using the next available handle index h^*. The simulator returns h^* to the adversary.

Intuition: Note that a consistent monomial made out of non-primed variables corresponds to the evaluation of the primary program on some input χ, and the counterpart monomial of primed variables corresponds to the evaluation of the dummy program on the same input. By construction, if $BP(\chi) = 1$ then these two evaluations yield the same element of $\mathbb{Z}_p$, even though as NCFPs they correspond to different terms. In the simulation we therefore identify these two terms, and only keep one of them. (The choice to keep the dummy term is arbitrary, we just as well could have kept the primal term instead.)

We note that simulator above uses super-polynomial time (and space), but this is irrelevant since the statement we are proving is an unconditional probability statement.

C.3.1 Proof that the simulation works

Now we need to argue that the view of the adversary A when interacting with the simulation strategy above is close to its view when interacting with the representation oracle, regardless of whether the execution uses (J, σ) or (J, σ'). It is clear by construction that whenever the simulator returns an existing handle from its database (i.e., the latest query returns the same NCFP as a previous one), then also the representation oracle must return the same existing handle (since also the actual matrix must be the same in this case). It is left to show that when the simulator returns a new handle (corresponding to a new NCFP) then whp also the representation oracle would have returned a new handle (corresponding to a new actual matrix modulo p).

For the arguments below, recall again that the coloring constraints restrict what monomials we could have in the NCFPs in $\mathcal{P}$. Specifically the variables can only appear in a specific order. Specifically, $\mathbf{s}$ can only be followed by $D_{1,0}$ or $D_{1,1}$. Similarly for each $i \in [n-1]$, $D_{i,0}$ and $D_{i,0}$ can only be followed by $D_{i+1,0}$ or $D_{i+1,1}$. Finally, $D_{n,0}$ and $D_{n,1}$ can only be followed by $\mathbf{t}$. Similar constraints are also true for the primed variables. This implies that primed and non-primed variables cannot be multiplied, and they can only be added for terms that include the bookends $\tilde{\mathbf{s}}$ and $\tilde{\mathbf{t}}$ (or $\tilde{\mathbf{s}}'$ and $\tilde{\mathbf{t}}'$). We partition the argument to two cases: one for queries that return a 1×1 matrix with colors $LC = 0, RC = n + 2$, and the other for all the other queries. We begin with the latter case.

NCFPs with $LC \neq 0$ or $RC \neq n + 2$. This case corresponds roughly to the "partial evaluation attacks" from Section 3. Consider an adversary query that returns a matrix with left color $LC \neq 0$ (i.e. i or i' for $i > 0$) or right color $RC \neq n + 2$ (i.e., $RC = j$ or $RC = j'$ for $j < n + 2$). Below we provide the details for the non-primed case $LC > 0$ and $RC < n + 2$, treatment of the other cases is analogous.

Recall that we consider here a query that results in a new formal polynomial, different than the formal polynomials in all previous queries, we denote the previous NCFPs that the simulator has in its database before that query by $P_1, P_2, \dots, P_r$, and the new NCFP generated by the query is denoted P. Our goal is to show that with high probability over the choices of the representation oracle, the actual matrix that result is also different from the matrices in all the previous queries. This would imply that in both the simulation and the real execution, the adversary will get a new handle in reply to this query.

For the case $LC = i > 0$ and $RC = j < n+2$, the coloring constraints imply that all the variables in P must be non-primed, and also that P does not include the variables $\tilde{\mathbf{s}}$ or $\tilde{\mathbf{t}}$. Thus P describes an $N \times N$ matrix of the form $c \cdot R_{i-1} D R_j^{-1}$, where D is mostly diagonal except in the bottom 5×5 quadrant and the R's are invertible matrices. Fix any one of the previous NCFPs P_k and we have two sub-cases:

- P and P_k are colored differently. This is the easy case, by definition of the model the handle that is returned to the adversary for this query (in both the simulation and the real execution) is different than what was returned for P_k.
- P and P_k have the same left and right colors. Hence also P_k is of the form $c_k \cdot R_{i-1} D_k R_j^{-1}$. Note that the scalars $c, c_k \in \mathbb{Z}_p$ are completely determined by the queries of the adversary, and are independent of the random choices of the representation oracle in the real execution, and the matrices R_{i-1} and R_j are the same for P and P_k. It follows that the actual matrices in the real execution are equal if and only if $c \cdot D = c_k \cdot D_k \pmod{p}$.

  Note now that the matrices $c \cdot D$ and $c_k \cdot D_k$ can themselves be expressed as formal multilinear polynomials over $\mathbb{Z}_p$ in the variables $D_{i,b}$ and these polynomials have degree at most n and they must differ as formal polynomials (since P and P_k differ). Moreover, since the $D_{i,b}$'s are all diagonal (except the bottom-right 5×5 quadrant), then the top-left entries in $c \cdot D$ and $c_k \cdot D_k$ can be expressed as the same formal polynomials in the top-left entries of the $D_{i,b}$'s.

  Since in our construction all these top-left entries are chosen independently at random in $\mathbb{Z}_p$, we conclude that the top-left entry in the matrix $c \cdot D - c_k \cdot D_k \bmod p$ is obtained as the evaluation of a non-zero multilinear polynomial of degree at most n, in variables that are chosen uniformly in $\mathbb{Z}_p$. The Schwartz-Zippel lemma implies that this entry is non-zero, except with probability bounded below n/p. Hence the probability that the actual matrices corresponding to P, P_k are equal is also bounded below n/p.

All the other sub-cases are treated mostly the same: The non-primed case $LC > 0$ and $RC = n+2$ is identical to above, except that here we consider the first entry in the column vectors $c \cdot D\mathbf{t}$ and $c_k \cdot D_k \mathbf{t}$. The non-primed case $LC = 0$ and $RC < n+2$ is similar, but we consider the $m+1$'st entry in the row vectors $c \cdot \mathbf{s}D$ and $c_k \cdot \mathbf{s}D_k$ (note that the 1'st entry is identically zero by construction). Finally, treatment of the primed variables is identical to their non-primed counterparts.

NCFPs with $LC = 0$ and $RC = n+2$. This case corresponds roughly to the "mixed input attacks" from Section 3. In this case the formal polynomial P correspond to a 1×1 matrix, and all the terms in P are either of the form $X = c \cdot \tilde{\mathbf{s}} \big(\prod_{i=1}^{n} \tilde{D}_{i,b_i} \big) \tilde{\mathbf{t}}$ or $X = c \cdot \tilde{\mathbf{s}}' \big(\prod_{i=1}^{n} \tilde{D}'_{i,b_i} \big) \tilde{\mathbf{t}}'$ (for a scalar $c \in \mathbb{Z}_p$ and bits b_i).

Recall that the structure of $\tilde{\mathbf{s}}$ and $\tilde{\mathbf{t}}$ cancels out all the random diagonal entries in the $\tilde{D}_{i,b}$'s and $\tilde{D}'_{i,b}$'s except the bottom-right 5×5 quadrant. Hence the actual 1×1 matrix corresponding to every such term therefore depends only on the permutation matrix in the bottom 5×5 quadrant, the random scalars α_{i,b_i} (or α'_{i,b_i}), and the vectors $\mathbf{s}^*, \mathbf{t}^*$ (or $\mathbf{s}^{*\prime}, \mathbf{t}^{*\prime}$). Specifically, a term of primed variables corresponds to $\big(\prod_{i \in [n]} \alpha'_{i,b_i} \big) \cdot \rho$ where $\rho = \langle \mathbf{s}^{*\prime}, \mathbf{t}^{*\prime} \rangle$, and a term of non-primed variables corresponds to $\big(\prod_{i \in [n]} \alpha_{i,b_i} \big) \cdot \mathbf{s}^* \big(\prod_{i \in [n]} A_{i,b_i} \big) \mathbf{t}^*$.

Below we refer to the set of steps I_j in the program BP (which are the steps that examine the input bit χ_j) as the j'th *block of steps*. A monomial X as above is *consistent with the j'th block* if for all $i \in I_j$ we have $b_i = 0$ or for all $i \in I_j$ we have $b_i = 1$. Otherwise, if for some steps $i \in I_j$ we have $b_i = 0$ and for others $b_i = 1$, we call this monomial *inconsistent with the j'th block*. Clearly, a monomial is consistent with some input $\chi \in \{0,1\}^\ell$ if and only if it is consistent with all the blocks $j = 1, \dots, \ell$.

Next recall that the $\alpha_{i,b}$'s and $\alpha'_{i,b}$'s were chosen at random subject to the constraint that for every $j \in [\ell]$ and $b \in \{0,1\}$ we have $\prod_{i \in I_j} \alpha_{i,b} = \prod_{i \in I_j} \alpha'_{i,b}$. Below it will be convenient to consider the following procedure for choosing these scalars: Denoting $k = |I_j|$ and $I_j = \{i_1, \ldots, i_k\}$, for $b \in \{0,1\}$ we draw $2k$ random scalars $\gamma_{j,b} \in \mathbb{Z}_p$ for $j = 1, \ldots, 2k$, and then set $\alpha_{i_j,b} = \gamma_{2j-1,b} \cdot \gamma_{2j,b}$ and $\alpha'_{i_j,b} = \gamma_{2j,b} \cdot \gamma_{2j+1,b}$ (with index arithmetic modulo $2k$). It is easy to see that this process indeed generates the uniform distribution subject to the constraint above.

With this procedure, for every concrete choice of the vectors $\mathbf{s}^*, \mathbf{t}^*, \mathbf{s}^{*\prime}$ and $\mathbf{t}^{*\prime}$, we can express the scalar corresponding to each NCFP monomial X as a formal monomial in the γ's. Below we prove that for any two distinct NCFPs $P \neq P'$, it holds with high probability (over the choice of the vectors) that the corresponding formal polynomials in the γ's are also be distinct. Applying the Schwartz-Zippel lemma again, we conclude that the scalars that correspond to P, P_k are different whp. A key lemma that we use is the following:

Lemma 2 (Straddling Lemma)**.** Fix an integer k, and consider drawing $2k$ random scalars $\gamma_j \leftarrow \mathbb{Z}_q$ for $j = 1, \ldots, 2k$, and setting $\alpha_i = \gamma_{2i-1} \cdot \gamma_{2i}$ and $\alpha'_i = \gamma_{2i} \cdot \gamma_{2i+1}$ (with index arithmetic modulo $2k$).

Also fix two subsets $I, I' \subseteq [k]$ with $\emptyset \subsetneq I \subsetneq [k]$, then we have that $\prod_{i\in I} \alpha_i$ and $\prod_{i\in I'} \alpha'_i$ are distinct formal multilinear monomials in the variables $\gamma_1, \gamma_2 \ldots \gamma_{2k}$.

Proof. It is clear that the two products are indeed multilinear monomials in the γ's, it remains to show that they are distinct. Two case arise, either $I = I'$ or $I \neq I'$:

- If $I = I'$, then since I is a non empty set strictly contained in I, there must exists an index $j \in I$ such that $j - 1 \bmod k \notin I = I'$. This implies that $\prod_{i\in I} \alpha_i$ contains γ_{2j-1} but $\prod_{i\in I'} \alpha'_i$ does not.
- $I \neq I'$: In this case there exists j such that either $j \in I \setminus I'$, or $j \in I' \setminus I$. In the first case we get that $\prod_{i\in I} \alpha_i$ contains γ_{2j} but $\prod_{i\in I'} \alpha'_i$ does not, and in the second case $\prod_{i\in I'} \alpha'_i$ contains γ_{2j} but $\prod_{i\in I'} \delta_i$ does not.

■

Armed with Lemma 2, we can now finally present our analysis. Let P, P' be two fixed distinct NCFPs in the database of the simulator at the end of the simulation, both with colors $LC = 0, RC = n + 2$, and we prove that the scalar corresponding to $\Delta = P - P'$ in the real execution of the representation oracle is nonzero except with probability at most $2n/p$. Since Δ is a non-zero NCFP, then let X be a monomial in Δ. We analyze two cases, where X is either a consistent monomial or an inconsistent one.

Inconsistent Monomials. For an inconsistent monomial, assume that it is made up of non-primed variables $X = c \cdot \tilde{\mathbf{s}}\big(\prod_{i=1}^{n} \tilde{D}_{i,b_i}\big)\tilde{\mathbf{t}}$ (the case for primed variables is symmetric). As argued above, the scalar corresponding to this term can also be described as

$$c \cdot \big(\prod_{i\in[n]} \alpha_{i,b_i}\big) \cdot \mathbf{s}^*\Pi\mathbf{t}^* = c \cdot \big(\prod_{i\in[n]} \gamma_{2i-1,b_i}\gamma_{2i,b_i}\big) \cdot \mathbf{s}^*\Psi\mathbf{t}^*$$

(for some permutation matrix Ψ). For every concrete choice of $\mathbf{s}^*\Psi\mathbf{t}^*$ this gives a single monomial in the γ's with coefficient which is nonzero except with probability $O(1/p)$.

We claim that no other monomial in the NCFP Δ corresponds to the same product of γ's. It is easy to see that no other monomial of non-primed variables in Δ can correspond to the same γ's, since every other non-primed monomial must correspond to some $b'_i \neq b_i$ (for some $i \in [n]$) and thus include $\alpha_{i,b'_i} = \gamma_{2i-1,b'_i}\gamma_{2i,b'_i}$ rather than $\alpha_{i,b_i} = \gamma_{2i-1,b_i}\gamma_{2i,b_i}$. To see that also no primed monomial can correspond to the same product, we use Lemma 2 above. Since X is an inconsistent terms then in particular it is inconsistent with some block j. This means that the product of $\tilde{D}$'s contains both $\tilde{D}_{i,0}$'s for some $i \in I_j$ and $\tilde{D}_{i',1}$'s for some other $i' \in I_j$. Hence the corresponding product of the α's contains some of the $\alpha_{i,1}$'s for $i \in I_j$ but not all of them. We can therefore apply Lemma 2 using the set of steps where we use $\alpha_{i,1}$ as our non-empty proper subset I. The lemma then tells us that no product of the $\alpha'_{i,b}$'s can yield the same product of γ's, which is what we needed to show.

Since no other term in Δ has the same monomial in the γ's, it means in particular that the standard formal polynomial in the γ's must be nonzero whenever $\mathbf{s}^*\Psi\mathbf{t}^* \neq 0$, since nothing can cancel out this monomial.

Consistent Monomials. Assume that Δ consists only of consistent monomials, let X be one of these monomials, and let χ be the input that X is consistent with. Observe that if $BP(\chi) = 1$ then it means that X is made up of primed variables (since non-primed monomials that are consistent with accepted inputs are replaced by the simulator with their primed counterparts). In this case we again argue that no other term in Δ that corresponds to the same product of γ's as X, since every other (primed or non-primed) monomial must have some $\alpha_{i,1-b_i}$ in at least one step where X has α_{i,b_i}.

It remains to examine the case where X is consistent with an input χ such that $BP(\chi) = 0$. In this case Δ can perhaps contain another term corresponding to the the same product of γ's, i.e. one primed monomial $X' = c' \cdot \tilde{\mathbf{s}}' \big(\prod_{i=1}^{n} \tilde{D}'_{i,\chi_{\mathsf{inp}(i)}} \big) \tilde{\mathbf{t}}'$ and one non-primed $X = c \cdot \tilde{\mathbf{s}} \big(\prod_{i=1}^{n} \tilde{D}_{i,\chi_{\mathsf{inp}(i)}} \big) \tilde{\mathbf{t}}$. We assume w.l.o.g. that both terms exist with at least one of c, c' nonzero. Denoting $b_i = \chi_{\mathsf{inp}(i)}$, the (unique) term in Δ with the product of all the γ's corresponding to X, X' is therefore

$$\Big(\prod_{i \in [n]} \gamma_{2i-1,b_i} \cdot \gamma_{2i,b_i} \Big) \cdot \big(c' \cdot \langle \mathbf{s}^{*\prime}, \mathbf{t}^{*\prime} \rangle \ + \ c \cdot \mathbf{s}^* \, \Pi \, \mathbf{t}^* \big)$$

(where Π is the non-identity permutation matrix for output zero of BP). Since Π is a permutation matrix different from the identity and since at least one of c, c' is nonzero, then the expression $\big(c' \cdot \langle \mathbf{s}^{*\prime}, \mathbf{t}^{*\prime} \rangle + c \cdot \mathbf{s}^* \, \Pi \, \mathbf{t}^* \big)$ is nonzero except with probability $O(1/p)$. To be precise, for every permutation $\Pi \neq I$ and every $c, c' \in \mathbb{Z}_p$ not both zero, we have

$$1/p \ < \ \Pr \big[c' \cdot \langle \mathbf{s}^{*\prime}, \mathbf{t}^{*\prime} \rangle + c \cdot \mathbf{s}^* \, \Pi \, \mathbf{t}^* = 0 \big] \ < \ 2/p,$$

where the probability is taken over the choice of the vectors $\mathbf{s}^*, \mathbf{t}^*, \mathbf{s}^{*\prime}, \mathbf{t}^{*\prime}$ such that $\langle \mathbf{s}^*, \mathbf{t}^* \rangle = \langle \mathbf{s}^{*\prime}, \mathbf{t}^{*\prime} \rangle$. We again conclude that except with small probability over the choice of the vectors, the formal polynomial in the γ's must be nonzero.

Completing the proof. We have shown that for every nonzero NCFP Δ, it holds with probability at least $1 - O(1/p)$ (over the choice of the vectors $\mathbf{s}^*$, $\mathbf{t}^*$, $\mathbf{s}^{*\prime}$, and $\mathbf{t}^{*\prime}$) that the corresponding formal polynomial in the γ's is also nonzero. Since the latter formal polynomial has degree $2n$, is follows from the Schwartz-Zippel lemma that the actual scalar corresponding to Δ is nonzero with probability at least $1 - O(n/p)$. We thus conclude that for any pair of distinct NCFPs that the simulator has in its database, the representation oracle will have distinct matrices except with probability no more than $O(n/p)$. As there are T such NCFPs, the probability that any pair of them causes a collision is at most $\binom{T}{2} \cdot O(n/p) = O(T^2 n/p)$. And as long as there are no collisions, the view of the adversary in the simulated execution is identical to its view in the real one. This competes the proof of Theorem 6. ■

C.4 Justification for the model

We now describe our intuition for why attacks under the generic colored matrix model constitute "the most natural class of attacks" against our obfuscation scheme. Our results in this model should be viewed as ensuring that our scheme cannot be "broken by linear-algebra." We note, however, that at present we are not aware of any plausible attack on the scheme that does not fit in this model (except of course lattice-reduction attacks on the underlying GGH encoding scheme).

Recall that in our scheme we give out GGH-type encoding of the elements of the matrices $\tilde{D}_{i,b}$ and vectors $\tilde{\mathbf{s}}, \tilde{\mathbf{t}}$, which are computed as $\tilde{\mathbf{s}} = \mathbf{S}R_0^{-1}$, $\tilde{D}_{i,b} = R_{i-1} D_{i,b} R_i^{-1}$, and $\tilde{\mathbf{t}} = R_n \mathbf{t}$, for randomly-chosen R_i's. The elements of $\tilde{D}_{i,b}$ are encoded relative to a GGH-denominator z_i, the vectors $\tilde{\mathbf{s}}, \tilde{\mathbf{t}}$ are encoded relative to z_0, z_{n+1}, respectively, and the zero-test parameter is given relative to a product of all the z_i's. Regarding plausible attacks on the scheme, we note the following points:

- Using the zero-test parameter to cancel out the z_i's require an element which is encoded relative to the product of all of them, and obtaining such element from the given public parameters boils down essentially to computing a multilinear function with each term having exactly one element encoded relative to each z_i. Note that the presence of noise in the numerator seems to effectively destroy the utility of performing any higher degree computations, such as building a term which is degree-two in each z_i in the denominator, and then attempting to compute a "square root".

- Trying to mix and match elements from different matrices/vectors, or to multiply these matrices out of order, makes it seemingly impossible to eliminate the random R_i's.
- At the same time, it seems that GGH-specific attacks such as the "Weak discrete logarithm attack" from [GGH13a] are not applicable to our scheme, since this scheme does not provide encoding of known constants (such as zero or one) that are needed in such attacks. Although a nontrivial encoding of zero relative to the product of all the z_i's is obtained as part of the computation, it is not clear how an attacker can compute a "lower level encoding" of zero (relative to just part of the z_i's) as required those attacks.

Given these difficulties in devising attacks that do not respect the matrix structure or their order, we therefore devised the generic colored matrix model to isolate and analyze the attacks that do respect this structure.