Universal Obfuscation and Witness Encryption: Boosting Correctness and Combining Security

Prabhanjan Ananth* Aayush Jain† Moni Naor‡ Amit Sahai§ Eylon Yogev‡

Abstract

Over the last few years a new breed of cryptographic primitives has arisen: on one hand they have previously unimagined utility and on the other hand they are not based on simple to state and tried out assumptions. With the on-going study of these primitives, we are left with several different candidate constructions each based on a different, not easy to express, mathematical assumptions, where some even turn out to be insecure.

A *combiner* for a cryptographic primitive takes several candidate constructions of the primitive and outputs one construction that is as good as any of the input constructions. Furthermore, this combiner must be efficient: the resulting construction should remain polynomial-time even when combining polynomially many candidate. Combiners are especially important for a primitive where there are several competing constructions whose security is hard to evaluate, as is the case for indistinguishability obfuscation (IO) and witness encryption (WE).

One place where the need for combiners appears is in design of a *universal construction*, where one wishes to find "one construction to rule them all": an explicit construction that is secure if *any* construction of the primitive exists.

In a recent paper, Goldwasser and Kalai posed as a challenge finding universal constructions for indistinguishability obfuscation and witness encryption. In this work we resolve this issue: we construct universal schemes for IO, and for witness encryption, and also resolve the existence of combiners for these primitives along the way. For IO, our universal construction and combiners can be built based on *either* assuming DDH, or assuming LWE, with security against subexponential adversaries. For witness encryption, we need only one-way functions secure against polynomial time adversaries.

*Center for Encrypted Functionalities, Computer Science Department, UCLA. Email: prabhanjan@cs.ucla.edu. This work was partially supported by grant #360584 from the Simons Foundation and the grants listed under Amit Sahai.

†Center for Encrypted Functionalities, Computer Science Department, UCLA. Email: aayush@cs.ucla.edu.

‡Weizmann Institute of Science, Israel. Email: {moni.naor,eylon.yogev}@weizmann.ac.il. Supported in part by a grant from the I-CORE Program of the Planning and Budgeting Committee, the Israel Science Foundation, BSF and the Israeli Ministry of Science and Technology. Moni Naor is the incumbent of the Judith Kleeman Professorial Chair.

§University of California Los Angeles and Center for Encrypted Functionalities. Email: sahai@cs.ucla.edu. Research supported in part from a DARPA/ARL SAFEWARE award, NSF Frontier Award 1413955, NSF grants 1228984, 1136174, 1118096, and 1065276, a Xerox Faculty Research Award, a Google Faculty Research Award, an equipment grant from Intel, and an Okawa Foundation Research Grant. This material is based upon work supported by the Defense Advanced Research Projects Agency through the ARL under Contract W911NF-15-C-0205. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, or the U.S. Government.

Contents

1 Introduction

We live in a golden, but dangerous, age for cryptography. New primitives are proposed along with candidate constructions that achieve things that were previously in the realm of science fiction. Two such notable examples are *indistinguishability obfuscation*[1] (IO), and *witness encryption*[2] (WE). However, at the same time, we are seeing a steady stream of new attacks on assumptions that are underlie, or at least are closely related to, these new candidates. With this proliferation of constructions and assumptions comes the question: how do we evaluate these various assumptions, which constructions do we choose and how do we actually use them?

What is better: one candidate construction of indistinguishability obfuscation (IO) or two such candidate constructions? What about a polynomial-sized family of candidates? The usual approach should be "the more the merrier", but how do we use these several candidates to actually obfuscate? The relevant notion is that of a *combiner*: it takes several candidates for a primitive and produces one instance of the primitive so that if any of the original ones is a secure construction then the result is a secure primitive. Furthermore, this combiner must be efficient: the resulting construction should remain polynomial-time. Another issue is what do we assume about the insecure constructions. Are they at least correct, i.e. do they maintain the functionality, or can they be arbitrarily faulty? We are interested in a combiner that adds very little complexity to the basic underlying schemes and assumes as little as possible regarding the insecure schemes, i.e. they may be completely dysfunctional. Furthermore, we would like the assumptions underlying our combiner to be as minimal and standard as possible.

One candidate to rule them all (theoretically speaking). In fact, we can even go further: A closely related issue to the existence of combiners is that of a *universal construction* of a primitive: a concrete construction of the primitive that is secure if *any* secure construction exists. In the context of candidate constructions, a universal IO candidate would change the game considerably between attacker and defender: Currently, each IO candidate is based on specific mathematical techniques, and a cryptanalysis of each candidate can be done by finding specific weaknesses in the underlying mathematics. With a universal IO candidate, the only way to give a cryptanalysis of this candidate would be to prove that no secure IO scheme exists. To the best of our knowledge, no plausible approaches have been proposed for obtaining such a proof. Thus, a universal IO scheme would vastly raise the bar on what an attacker must do.

Furthermore, intriguingly, we note that IO exists if **P**=**NP**. In contrast to other objects in cryptography, IO by itself does not imply hardness. This raises the possibility of a future non-constructive existence proof for IO, even without needing to resolve **P** vs **NP**. If we have a universal IO scheme, then any such non-constructive proof would be made explicit: the universal IO scheme would be guaranteed to be secure.

Indeed, in a recent opinion paper regarding assumptions Goldwasser and Kalai [GK16] wrote:

> *We pose the open problem of finding a universal instantiations for other generic assumptions, in particular for IO obfuscation, witness encryption, or 2-message delegation for NP.*

[1] Indistinguishability obfuscation is the ability to scramble a program so that it is not possible to decide what was the source code out of two semantically equivalent options.

[2] Witness encryption is a method for encrypting a message relative to a string x and language L so that anyone with a witness w that $x \in L$ can decrypt but if $x \notin L$ then no information about the message is leaked.

In this work we resolve two out of those three primitives, namely IO and witness encryption, for security against subexponential adversaries for IO, and polynomial adversaries for witness encryption. Our universal constructions also resolve the existence of combiners for these primitives along the way. For IO, our universal construction and combiners can be built based on *either* assuming DDH, or assuming LWE, with security against subexponential adversaries. For witness encryption, we need only one-way functions secure against polynomial adversaries.

The status of iO schemes or – are we dead yet? The state of the art of iO is in flux. There is a steady stream of proposals for constructions and a similar stream of attacks on various aspects of the constructions. In order to clarify the state of the art in Appendix A we provide a detailed explanation of the constructions, the attacks and what implications they have (a summary is provided in Figure 13). As of now (March 2016) there is no argument or attack known that implies that all iO schemes or primitives used by them are broken.

Brief history of combiners and universal cryptographic primitives. The notion of a combiner and its connection to universal construction were formalized by Harnik et al. [HKN+05] (see also Herzberg [Her05]). An early instance of a combiner for encryption is that of Asmuth and Blakely [CA81]. A famous example of a universal construction (and the source of the name) is that of one-functions due to Levin [Lev87] (for details see Goldreich [Gol01, §2.4.1]).

1.1 Our Results

Our first result is a construction of an IO combiner. We give two separate constructions, one using LWE, and other using DDH. Thus, we can build IO combiners from two quite different assumptions.

Theorem 1 (Informal)**.** *Under the hardness of Learning with Errors (LWE) and IO secure against sub-exponential adversaries, there exist an IO combiner.*

Theorem 2 (Informal)**.** *Under the hardness of Decisional Diffie-Hellman (DDH) and IO secure against sub-exponential adversaries, there exist an IO combiner.*

We show how to adapt the LWE-based IO combiner to obtain a universal IO scheme.

Theorem 3 (Informal)**.** *Under the hardness of Learning with Errors (LWE) against sub-exponential adversaries and the existence of IO secure against sub-exponential adversaries, there exists a universal IO scheme.*

For witness encryption, we have similar results, under assumptions widely believed to be weaker. We prove the following theorem.

Theorem 4 (Informal)**.** *If one-way functions exist, then there exist a secure witness encryption combiner.*

Again, we extend this and get a universal witness encryption scheme.

Theorem 5 (Informal)**.** *If one-way functions and witness encryption exist, then there is a universal witness encryption scheme.*

Theorem 5 assumes the existence of one-way functions. Notice that if $\mathsf{P} = \mathsf{NP}$ then WE exist, however, one-way functions do not. Thus, in most cryptographic application one-way functions are used as an additional assumption. Nevertheless, we can make a stronger statement: If there exist

any hard-on-average language in NP then there is a universal WE scheme. In [KMN+14] it was shown that the existence of witness encryption and a hard-on-average language in NP implies the existence of one-way functions. By combing this with Levin's universal one-way function [Lev87] we obtain our result.

In Section 9 we present the constructions of universal secret sharing for NP and universal witness PRFs. Both these constructions assume only one-way functions.

2 Techniques

We present the technical challenges and describe how we overcome them.

2.1 Universal Obfuscation

A natural starting point is to revisit the construction of universal one-way functions [Lev87] – constructions of other known universal cryptographic primitives [HKN+05] have the same flavor. An explicit function f is said to be a universal one-way function if the mere existence of any one-way function implies that f is one-way.

The universal one-way function f_{univ} on input $x = y_1||\ldots||y_\ell$, where $|x| = \ell^2$, executes as follows[3]:

1. Interpret the integer $i \in \{1, \ldots, \ell\}$ as a Turing machine M_i. This interpretation is quite standard in the computational complexity literature[4].
2. Output $M_1(y_1)||\cdots||M_\ell(y_\ell)$.

To argue security, we exploit the fact that there exists a secure one-way function represented by Turing machine M_{owf}. Let ℓ_0 be an integer that can be interpreted as M_{owf}. We argue that it is hard to invert $M_{\mathsf{owf}}(x)$, where x has length at least ℓ_0^2 and is drawn uniformly at random. To see why, notice that in Step 1, M_{owf} will be included in the enumeration. From the security of M_{owf} it follows that it is hard to invert $M_{\mathsf{owf}}(y_{\ell_0})$, where y_{ℓ_0} is the ℓ_0^{th} block of x. This translates to the un-invertibility of $f_{\mathsf{univ}}(x)$. This proves that f_{univ} is one-way[5].

Let us try to emulate the same approach to obtain universal indistinguishability obfuscation. On input circuit C, first enumerate the Turing machines $M_1, \ldots, M_\ell$, where ℓ here is the size of the circuit C. We interpret M_i's as indistinguishability obfuscators. It is not clear how to implement the second step in the context of obfuscation – unlike one-way functions we cannot naïvely break the circuit into blocks and individually obfuscate each block. We need a mechanism to jointly obfuscate a circuit using multiple obfuscators $M_1, \ldots, M_\ell$ such that the security of the joint obfuscation is guaranteed as long as one of the obfuscators is secure. This is where indistinguishability obfuscation combiners come in. Designing combiners for indistinguishability obfuscation involves a whole new set of challenges and we deal with them in a separate section (Section 2.2). For now, we assume we have such combiners at our disposal.

[3] If x can not be expressed of this form then suitably truncate x till it is of this form.

[4] This fact was used to prove the famous Gödel's incompleteness theorem [Göd31].

[5] Note that the definition of one-way function only requires un-invertibility to hold for sufficiently long inputs. This requirement is satisfied by f_{univ} as its un-invertibility holds for inputs of lengths greater than ℓ_0^2.

Warmup Attempt. Using combiners for IO, we propose the following approach to achieve universal obfuscation. The universal obfuscator IO_{univ} on input circuit C executes the following steps:

1. Interpret the integer $i \in \{1, \ldots, \ell\}$ as a Turing machine M_i.
2. Obfuscate C by applying the IO combiner on the machines $M_1, \ldots, M_\ell$. Output the result $\overline{C}$ of the IO combiner.

Unlike the case of one-way functions, in addition to security we need to argue correctness of the above scheme. An obfuscator M_i is said to be correct if the obfuscated circuit $M_i(C)$ is equivalent to C (or agrees on most inputs) and this should be true *for every circuit* C. This in turn depends on the correctness of obfuscators $M_1, \ldots, M_\ell$. But we don't have any guarantee on the correctness of $M_1, \ldots, M_\ell$.

Test-and-Discard. We handle this by first checking for every i whether the obfuscator M_i is correct. This is infeasible in general. However, we test the correctness of M_i only on the particular circuit obfuscated by M_i during the execution of the universal obfuscation. In more detail, suppose we execute IO_{univ} on circuit C and during the execution of the IO combiner, let $[\mathbf{C}]_i$ (derived from C) be the circuit that we obfuscate using machine M_i. Then we test whether $M_i([\mathbf{C}]_i)$ agrees with M_i on significant fraction of inputs. This can be done by picking inputs at random and testing whether both circuits (obfuscated and un-obfuscated) agree on these inputs. If M_i fails the test, it is discarded. If it passes the test, then M_i cannot be used directly since $M_i([\mathbf{C}]_i)$ could agree with $[\mathbf{C}]_i$ on $(1 - 1/\mathrm{poly})$-fraction of inputs and yet it could pass the test with non-negligible probability. So we need to reduce the error probability of $M_i([\mathbf{C}]_i)$ to negligible before it is ready to be used.

Correctness Amplification. A first thought would be to use the recent work that shows an elegant correctness amplification for IO by Bitansky-Vaikuntanathan [BV16]. In particular, they show how to transform an obfuscator that is correct on at least $(1/2 + 1/\mathrm{poly})$-fraction of inputs into one that is correct on all inputs. At first glance this seems to be "just what the doctor ordered", there is, however, one catch here: their transformation is guaranteed to work if the obfuscator is correct *for every circuit* C on at least $(1/2 + 1/\mathrm{poly})$-fraction of inputs. However, we are only ensured that it is approximately correct on *only one* circuit! Nonetheless we show how to realize correctness amplification with respect to a single circuit and ensure that $M_i([\mathbf{C}]_i)$ does not agree with $[\mathbf{C}]_i$ on only negligible fraction of inputs. Once we perform the error amplification, the obfuscator M_i will be used in the IO combiner. In the end, the result of the IO combiner will be an obfuscated circuit $\overline{C}$; the correctness guarantees of $M_i([\mathbf{C}]_i)$, for every i, translate to the corresponding correctness guarantee of $\overline{C}$.

Handling Selective Abort Obfuscators. We now move on to security. For two equivalent circuits C_0, C_1, we need to argue that their obfuscations are computationally indistinguishable. To do this, we need to rely on the security of IO combiner. The security of IO combiner requires that as long as one of the machines M_i is a secure obfuscator[6] then the joint obfuscation of C_0 using $M_1, \ldots, M_\ell$ is indistinguishable from the joint obfuscation of C_1 using the same candidates. The fact that same candidates are used is crucial here since the final obfuscated circuit could potentially reveal the description of the obfuscators combined.

[6] Just as in the case of one-way functions, for sufficiently large circuits C, one of the enumerated machines will be a secure obfuscator.

However, there is no such guarantee offered in our case! Recall that we have a 'test-and-discard' phase where we potentially throw out some obfuscators. It might be the case that a particular candidate M_{mal} is correct only on circuits derived from C_0 but fails on circuits derived from C_1. We call such obfuscators *selective abort obfuscators*. Clearly, selective abort obfuscators can lead to a complete break of security. In fact, if there are ℓ obfuscators used then potentially $\ell - 1$ of them could be of selective abort type. To protect against these adversarial obfuscators we ensure that the distribution of the ℓ derived circuits is computationally independent from the circuit to obfuscate.

Issue of runtime. While the above ideas ensure correctness and security, we haven't yet shown that our scheme is efficient. In fact it could potentially be the case our scheme never halts on some inputs[7]. This could happen since we have no a priori knowledge on the runtime of the obfuscators considered. We propose a naïve solution to this problem: we assume the knowledge of an upper bound on the runtime of the actually secure obfuscator. In some sense, the assumption of time bound might be inherent – without this we are required to predict a bound on the runtime of a Turing machine and we know in general this is an undecidable problem.

2.2 Combiners for Indistinguishability Obfuscation

We now focus our attention in constructing an IO combiner. Recall, in the setting of IO combiner we are given multiple IO candidates[8] with all of them satisfying correctness but with only one of them being secure. We then need to combine all of them to produce a joint obfuscator that is secure.

This scenario is reminiscent of a concept we are quite familiar with: Secure Multi-Party Computation (MPC). In the secure multi-party computation setting, there are multiple parties with individual inputs and the goal of all these parties is to jointly compute a functionality. The privacy requirement states that the inputs of the honest parties are hidden other than what can be leaked by the output.

Indeed, MPC provides a natural template to solve the problem of building an IO combiner: Let $\Pi_1, \ldots, \Pi_n$ be the IO candidates and let C be the circuit to be obfuscated.

- Secret share the circuit C into n shares $s_1, \ldots, s_n$.
- Take any n-party MPC protocol for the functionality $\mathcal{F}$ that can tolerate all-but-one malicious adversaries [GMW87]. The n-input functionality $\mathcal{F}$ takes as input $((s_1, x_1), (s_2, x_2), \ldots, (s_n, x_n))$; reconstructs C from the shares and outputs $C(x)$ only if $x = x_1 = \cdots = x_n$.
- Obfuscate the "code" (or algorithmic description) of the i^{th} party using Π_i.
- The joint obfuscation of all the parties is the final obfuscated circuit!

To evaluate on an input x, perform the MPC protocol on the obfuscated parties with (s_i, x) being the input of the i^{th} party.

[7] This is not a problem for the case of one-way functions because of a well established result that given any one-way function that runs in arbitrary polynomial time we can transform it into a different one-way function that takes quadratic time.

[8] IO candidates are just indistinguishability obfuscation schemes. The scheme of [BGK+14] is an example of an IO candidate, scheme of [PST14] is another example and so on.

Could the above approach lead to a secure IO combiner? The hope is that the security of MPC can be used to argue that one of the shares (corresponding to the honest party) is hidden which then translates to the hiding of C.

However, we face some fundamental challenges in our attempt to realize the above template, and in particular we will not be able to just invoke general solutions like [GMW87], and we will need to leverage more specialized cryptographic objects.

Challenge #1: Single-Input versus Multi-Inputs security. Recall that in the context of MPC, we argue the security only for a *particular set of inputs* (one for every party) in one session. In particular, a fresh session needs to be executed to compute the functionality on a different set of inputs. However, obfuscation is *re-usable* – it enables multiple evaluations of the obfuscated circuit. The obfuscated circuit should hide the original circuit independent of the number of times the obfuscated circuit is evaluated. On the other hand, take the classical Yao's garbled circuits [Yao86], used in two party secure computation, for example. Suppose we are provided with the ability to evaluate the garbled circuit on two different inputs then the security completely breaks down.

Challenge #2: Power of the Adversary. Suppose we start with an arbitrary multi-round MPC protocol. In the world of IO combiners, this corresponds to executing a candidate multiple times during the evaluation of a single input. While the party in the MPC protocol can maintain state in between executions, a candidate does not have the same luxury since it is *stateless*. This enables the adversarial evaluator to launch so called *resetting attacks*: during the evaluation of the IO combiner on a single input x, a secure candidate could first be executed on transcripts consistent with x and later executed on transcripts consistent with a different input x'. Since, the secure candidate cannot maintain state, it is possible that it cannot recognize such a malicious execution. We need to devise additional mechanisms to prevent such attacks.

Challenge #3: Virtual Black Box Obfuscation versus IO. The above two challenges exist even if we had started off with virtual black box (VBB) obfuscation. Dealing with indistinguishability obfuscation as opposed to VBB presents us with fresh challenges. Indeed, in MPC, we take for granted that an honest party hides its input from the adversary. However, if we obfuscate the parties using IO, it is not clear whether the relevant input – the share of C – is hidden at all. Arguing this requires importing IO-friendly tools (for instance, [SW14]) studied in the recent literature and making it compatible with the tools of MPC that we want to use.

We will see next how to address the above challenges.

2.2.1 Our Approach

We present two different approaches to construct IO combiners. The first solution, in addition to existence of IO, assumes the hardness of Decisional Diffie Hellman. The second solution assumes additionally the hardness of learning with errors. Common to both these solutions is a technique of [CLTV15] that we'll call the *partition-programming technique*. We give a brief overview of this technique below.

Partition-Programming Technique: Consider a randomized algorithm $P(\cdot,\cdot)$ that takes as input secret sk, public instance $x \in \{0,1\}^\lambda$ and produces a distribution $\mathcal{D}_x$. Suppose there exists a

simulator Sim that on input x outputs a distribution $\mathcal{D}_x^*$ such that the distributions $\mathcal{D}_x$ and $\mathcal{D}_x^*$ are statistically close.

Lets say we are given obfuscation of $P(sk, \cdot)$ (sk is hardwired in the program), we show how to use the partition-programming technique to *remove the secret sk*. We proceed in 2^λ hybrids: In the i^{th} hybrid, we have a hybrid obfuscated program that on input x, executes $P(sk, x)$ if $x \leq i$ but otherwise it executes $\mathsf{Sim}(x)$. Now, the indistinguishability of i^{th} hybrid and $(i+1)^{th}$ hybrid can be argued directly from the security of IO: here we are using the fact that the simulated distribution and the real distribution are statistically close. In the $(2^{\lambda+1})^{th}$ hybrid, we have a program that only uses Sim, on every input, to generate the output distribution. Thus, we have removed the secret sk from the program.

This technique will come in handy when we address Challenge #1. We will see below how this technique will be used in both the solutions.

DDH-Based Solution. We begin by tackling Challenge #2. We noted that using interactive MPC solutions are bound to result in resetting attacks. Hence, we restrict our attention to non-interactive solutions. We need to determine our communication pattern between the candidates. In particular, we consider the *"line"* communication pattern: Suppose there are n candidates $\Pi_1, \ldots, \Pi_n$ and let C be the circuit to be obfuscated. For this discussion, we use the same notation Π_i to also refer to the circuit obfuscated by the candidate Π_i. The first obfuscated circuit Π_1 produces an output that will be input to Π_2 and so on. In the end, Π_n will receive the input from Π_{n-1} and the output of Π_n will determine the final output.

Lets examine how to achieve a solution in the above communication model, by first considering a naïve approach: Π_1 has hardwired into it an encryption $\mathsf{Enc}(pk, C)$ of circuit C to be obfuscated. It receives an input x, it performs a part of the computation and sends the result to the next candidate Π_2 who performs another part of the computation, sends it to Π_3 and so on. In the end, the last candidate Π_n has the secret key sk to decrypt the output. This is clearly insecure because if both Π_1 and Π_n are broken then using sk and $\mathsf{Enc}(pk, C)$ we can recover the circuit C. This suggests the use of a re-encryption scheme. A re-encryption scheme is associated with public keys $pk_1, \ldots, pk_{n+1}$ and corresponding re-encryption keys $rk_{1\to 2}, \ldots, rk_{n\to(n+1)}$. The first candidate Π_1 will have hardwired into it $\mathsf{Enc}(pk_1, C)$ and the i^{th} candidate has hardwired into it the re-encryption key $rk_{i\to i+1}$. Thus, the i^{th} candidate performs part of the computation, re-encrypts with respect to pk_{i+1} using its re-encryption key $rk_{i\to i+1}$. We provide the secret key sk_{n+1}, corresponding to public key pk_{n+1}, as part of the obfuscated circuit. Using this, the evaluator can decrypt the output and produce the answer. Intuitively, as long as one candidate hides one secret key, the circuit C should be safe.

The natural next step is to figure out how to implement the "computation" itself: one direction would be to consider re-encryption schemes that are homomorphic with respect to arbitrary computations. However, we currently do not know of the existence of such schemes based on DDH (for LWE-based solutions, see below). We note that [ACG+14] faced similar hurdles while designing DDH-based multi-server delegation schemes. They employed the use of re-randomizable garbled circuits to implement the "computation" aspect of the above approach. A re-randomizable garbling scheme is a garbling scheme which is accompanied by a re-randomization algorithm that takes as input garbled circuit-input wire keys pair (GC, w_x) and outputs (GC^r, w_x^r).

Following along the lines of the approach of [ACG+14], we propose the following solution template:

1. First we compute the garbled circuit-wire keys pair (GC^1, w^1) of circuit C corresponding to the re-randomizable garbled circuits scheme. Here, w^1 comprises of keys associated to bits 0 and 1 with respect to every position. Π_1 has hardwired into it, $\mathsf{Enc}(pk_1, (\mathsf{GC}^1, w^1))$.

2. Π_1 takes as input x and produces $\mathsf{Enc}(pk_2, (\mathsf{GC}^2, w_x^2))$, where (GC^2, w_x^2) is obtained by first re-randomizing (GC^1, w^1) and then choosing the wire keys corresponding to x. This process is enabled using the re-encryption key $rk_{1\to 2}$. In addition, we require that the re-encryption process allows for homomorphic operations – in particular, it should allow for homomorphism of re-randomization operation of the garbling schemes.

3. The i^{th} candidate takes as input $\mathsf{Enc}(pk_i, (\mathsf{GC}^i, w_x^i))$; homomorphically re-randomizes the garbled circuit while simultaneously re-encrypting the ciphertext to obtain $\mathsf{Enc}(pk_{i+1}, (\mathsf{GC}^{i+1}, w_x^{i+1}))$.

4. In the end, the n^{th} candidate Π_n outputs $\mathsf{Enc}(pk_{n+1}, (\mathsf{GC}^{n+1}, w_x^{n+1}))$. Using the secret key sk_n, we can decrypt the output $(\mathsf{GC}^{n+1}, w_x^{n+1})$. We then evaluate the garbled circuit GC^{n+1} using the wire keys w_x^{n+1} to recover the output.

We employ a specific re-randomizable garbled circuits by [GHV10] and homomorphic re-encryption scheme by [BBS98], where both these primitives can be based on DDH. The above template does not immediately work since an adversarial evaluator could feed in incorrect inputs to the secure candidate. While [ACG+14] used non-interactive zero knowledge proofs (NIZKs) to resolve this issue, we need to employ "IO-friendly" proofs such as statistically-sound NIZKs [SW14, BP15]. Refer to Section 5.2 for the formal construction.

Security: To argue security, we need to rely on the security of re-encryption schemes in addition to the security guarantees of the other schemes. The security property of a re-encryption scheme states that given re-encryption keys $\{rk_{i\to i+1}\}_{i\in[n]}\backslash\{rk_{\mathbf{i}\to\mathbf{i}+1}\}$ and a secret key sk_{n+1}, it is computationally hard to distinguish $\mathsf{Enc}(pk_1, m_0)$ from $\mathsf{Enc}(pk_1, m_1)$.

To argue the security of universal obfuscator, we have to get rid of the re-encryption key corresponding to the secure candidate – indeed, in the case of [ACG+14] the re-encryption key corresponding to the honest party is removed in the security proof. In our scenario, however, this can only be implemented if we hardwire all possible outputs inside the code of the secure candidate. Clearly, this is not possible since there are exponentially many outputs. This is where we will use the *partition-programming technique* to remove the re-encryption key. To apply the technique, we argue that the re-encrypted ciphertexts are statistically close to freshly generated ciphertexts (which will be our simulated distribution) and this property holds for the particular instantiation of [BBS98] we are considering.

LWE-Based Solution. We give an alternate construction based on the learning with errors (LWE) assumption. One potential approach is to take the above solution and replace the DDH-based primitives with LWE-based primitives. Namely, we replace re-randomizable garbled circuits and re-encryption schemes with fully homomorphic encryption schemes. While we believe this is a viable approach, it turns out we can give an arguably more elegant solution by using the notion of *multi-key fully homomorphic encryption* [LTV12, CM15, MW16]. A multi-key FHE allows for generating individual public key-secret key pairs $\{pk_i, sk_i\}$ such that they can be later combined to obtain a joint public key $\mathbf{pk}$. To be more precise, given a ciphertext with respect to pk_i, there

is an "Expand" operation that transforms it into a ciphertext with respect to a joint public key **pk**. Once this done, the resulting ciphertext can be homomorphically evaluated just like any FHE scheme. The resulting ciphertexts can then be individually decrypted using sk_i's to obtain partial decryptions. Finally, there is a mechanism to combine the partial decryptions to obtain the final output.

Before we outline the solution below, we first fix the communication model. We consider a *"star"* interaction network: suppose there are n candidates $\Pi_1, \ldots, \Pi_n$. Each candidate Π_i is executed on the same input x. The joint outputs of all these candidates are then combined to obtain the final output. We propose the solution template based on multi-key FHE below.

1. We first secret share C into different shares $s_1, \ldots, s_n$.
2. Generate public key-secret key pairs $\{pk_i, sk_i\}$ for all $i \in [n]$. Encrypt s_i with respect to pk_i to obtain the ciphertext CT_i.
3. "Expand" every ciphertext CT_i into another ciphertext $\widehat{\mathsf{CT}_i}$ with respect to the joint public key **pk** which is a function of $(pk_1, \ldots, pk_n)$.
4. Every candidate Π_i has hardwired into it the secret key sk_i and ciphertext $\widehat{\mathsf{CT}_i}$. It takes as input x and first homomorphically evaluates the universal circuit U_x on $\widehat{\mathsf{CT}_i}$ to obtain an encryption of $C(x)$, namely $\widehat{\mathsf{CT}_i^{C(x)}}$, with respect to **pk**. Finally, using sk_i it outputs the partial decryption of $\widehat{\mathsf{CT}_i^{C(x)}}$.
5. The different partial decryptions output by the candidates are later combined to obtain the final output.

Security: We rely on the semantic security of the MFHE scheme to argue the security of the obfuscator. The security notion of multi-key FHE intuitively guarantees that the semantic security on ciphertext CT_i can be argued as long as the adversary never gets the secret key sk_i for some $i \in [n]$. A naïve approach is to remove the secret key $sk_{\mathbf{i}}$ from the secure candidate $\Pi_{\mathbf{i}}$. A similar issue that we encountered in the case of DDH-based solution arises here as well – we need to hardwire exponentially many outputs. Here comes partition-programming technique to the rescue! We show how to use this technique to remove $sk_{\mathbf{i}}$ after which we can argue the semantic security of MFHE, and thus the security of the obfuscator. To apply this technique, we need an alternate simulated distribution that simulates the partial decryption keys. We use the scheme of [MW16] who define such a simulatability property where the simulated distribution is statistically close to the real distribution. Refer to Section 5.1 for the formal construction.

The above LWE-based construction, unlike the DDH-based construction, satisfies some additional properties that are used to design a special type of IO combiner (we call this decomposable IO combiner in Section 4.1) which will then be used to construct universal indistinguishability obfuscation.

2.3 Universal Witness Encryption

We have discussed the construction of an IO combiner, and how to use the combiner to achieve a universal construction of IO. We describe our construction of a universal witness encryption (WE)

scheme. We show that a universal WE scheme exists on the sole assumption of the existence of a one-way function. First, we construct a WE combiner. This is achieved similarly to combiners for public-key encryption [HKN+05], using secret sharing. To encrypt a message m one secret shares the message to n shares such that all of them are needed to recover the message. Then, he encrypts each share using a different candidate. If at least one of the candidate schemes is secure then at least one share is unrecoverable and the message remains hidden.

The main challenge constructing a universal WE scheme is handling correctness. In the universal IO construction we had two main steps. The first was to test whether a candidate is approximately correct. This step was accomplished easily by sampling the obfuscated circuit on random inputs and verifying its correctness. Notice that although we cannot verify that the candidate is approximately correct for *all* circuits, we can verify that it is correct for the circuit in hand. The second step was to boost the correctness to achieve (almost) perfect security. This was obtained by suitably adapting the transformation described by Bitansky and Vaikuntanathan [BV16] to work in our setting where we only have a correctness guarantee for a single circuit.

The techniques used for the universal IO scheme seem not to apply for WE. Consider a language L with a relation R and a candidate scheme Π. To test correctness on an input x and a message m, one needs to encrypt the message and decrypt the resulting ciphertext. However, decryption requires a valid witness for x, where it might be NP-hard to find one! Testing, therefore, is limited to instances where it is easy to find a witness, a regime where witness encryption is trivial. Moreover, even given a approximate candidate, the boosting techniques used for the universal IO scheme do not apply for witness encryption.

Witness Injection. We describe a transformation that modifies any WE candidate scheme to be "testable" and also show how to boost the correctness of such testable schemes. Our first technique is to inject a "fake" witness for any x such that it will be easy to find this witness, for a party which has a trapdoor and computationally hard without the trapdoor (this is as in Feige and Shamir [FS90]. Moreover, this transformation will be indistinguishable for the (computationally bounded) candidate scheme.

Denote $(x, w) \in R$ for an instance x with a valid witness w. Let PRG be a length doubling pseudorandom generator. For any string z, we augment the language L and define L_z with the relation R_z such that

$$(x, w) \in R_z \iff (x, w) \in R \vee \mathsf{PRG}(w) = z.$$

Notice that if we choose $z = \mathsf{PRG}(s)$ for a random seed s, then L_z is the trivial language of all strings. Whereas, if z is chosen uniformly at random then with high probability L_z is equivalent to L, and these two cases are indistinguishable for anyone not holding the seed. This step enables us to test a candidate for some specific instance x: We choose $z \leftarrow \mathsf{PRG}(s)$, encrypt relatively to L_z, decrypt using the "fake" witness s and verify the output. After testing, we replace z with a random string (outside the range of the PRG) to get back the original language L. The problem is that this guarantees correctness only on our specific witness. The decryption algorithm, however, might refuse to cooperate for any other witness the user chooses to use.

Witness Protection Program. The next step is to apply what we call a *witness-worst-case* transformation. That is, a scheme that works on all witnesses with the same probability. Our main tool is a non-interactive zero knowledge (NIZK) proof system with statistical soundness. Suppose

(P, V) is such a NIZK scheme with a common random string σ. Then we further augment the language L_z to $L_{z,\sigma}$ with relation $R_{z,\sigma}$ such that:

$$(x, \pi) \in R_{z,\sigma} \iff V(\sigma, x, \pi) = 1.$$

If $(x, w) \in R$ is a valid instance witness pair for L, then the corresponding witness for $L_{z,\sigma}$ will be $\pi \leftarrow P(\sigma, x, w)$. That is, executing the transformed scheme on x, w relative to the language L translate to executing the original scheme on x, π relative to the language $L_{z,\sigma}$ for a randomly chosen z. Finally, to boost the success probability we apply a standard "BPP amplification"; encrypt many times and take the majority.

The result is roughly the following algorithm. We take any scheme and apply our witness-worst-case transformation for $z \leftarrow \mathsf{PRG}(s)$. Afterwards, we can test it on a fake witness while we are assured that it will work the same for any other witness. Then, if the scheme passes all tests, we replace z with a random string, and boost the correctness such that it will work for any witness with all but negligible probability. Finally, we apply the WE combiner to get a universal scheme. For the exact details see Section 2.3.

Relying on One-Way Functions. The description above of a universal witness encryption scheme used NIZK proof system as a building block, where we promised using only one-way functions. These proofs are not known to be implied by one-way functions and moreover no universal NIZK scheme is known (and this is an interesting open problem!). However, standard interactive zero knowledge can be constructed for any language in NP for one-way functions and moreover there exist a universal one-way function [Lev87]. Of course, we cannot use an interactive protocol, but, taking a closer look we observe that we can simulate a protocol between a verifier and a prover before the actual witness is given. That is, we can simulate a zero-knowledge protocol that might have many rounds, however, only the final round depends on the witness itself. Such protocols are known as *pre-process non-interactive zero-knowledge protocols* and where studied in [DMP88, LS90] where they proved how to construct them based on way-one functions.

For the final scheme, we will run the pre-process protocol to get two private states σ_V and σ_P for the verifier and the prover respectively, just before the final round. The modified language will be L_{z,σ_V} with relation R_{z,σ_V}, where

$$(x, \pi) \in R_{z,\sigma_V} \iff V(\sigma_V, x, \pi) = 1.$$

We will publish σ_P as part of the encryption so that a user, given witness w can produce the corresponding final round of the proof $\pi \leftarrow P(\sigma_P, x, w)$. Notice that the given the state of the prover, σ_P, the proof π is *not* zero-knowledge. However, since the decryption algorithm of the scheme does not get the state of the prover (only the state of the verifier) then from his perspective it is zero-knowledge.

3 Preliminaries

Let λ be the security parameter. For a distribution $\mathcal{D}$ we denote by $x \xleftarrow{\$}$ an element chosen from $\mathcal{D}$ uniformly at random. We denote that $\{\mathcal{D}_{1,\lambda}\} \approx_c \{\mathcal{D}_{2,\lambda}\}$, if for every PPT distinguisher $\mathcal{A}$, $\left| Pr\left[\mathcal{A}(1^\lambda, x \xleftarrow{\$} \mathcal{D}_{1,\lambda}) = 1\right] - Pr\left[\mathcal{A}(1^\lambda, x \xleftarrow{\$} \mathcal{D}_{2,\lambda}) = 1\right] \right| \leq \mathsf{negl}(\lambda)$ where negl is a negligible function. For a language L associated with a relation R with denote by $(x, w) \in R$ an instance $x \in L$ with a valid witness w. For an integer $n \in \mathbb{N}$ we denote by $[n]$ the set $\{1, \ldots, n\}$.

Important Notation. We introduce some notation that will be useful throughout this work. Consider an algorithm A. We define the *time function* of A to be T if the runtime of $A(x) \leq T(|x|)$. We are only interested in time functions which satisfy the property that $T(\text{poly}(n)) = |\text{poly}(T(n))|$.

3.1 Puncturable Pseudorandom Functions

A pseudorandom function family consisting of functions of the form $PRF_K(\cdot)$, that is defined over input space $\{0,1\}^*$, output space $\{0,1\}^{\chi(\lambda)}$ and key $K \in \{0,1\}^\lambda$, is said to be a *puncturable PRF family* if there exists a PPT algorithm $\mathsf{Puncture}$ that satisfies the following properties:

- **Functionality preserved under puncturing.** $\mathsf{Puncture}$ takes as input a PRF key $K \in \{0,1\}^\lambda$, input $x \in \{0,1\}^*$ and outputs K_x such that for all $x' \neq x$, $PRF_{K_x}(x') = PRF_K(x')$.
- **Pseudorandom at punctured points.** For every PPT adversary $(\mathcal{A}_1, \mathcal{A}_2)$ such that $\mathcal{A}_1(1^\lambda)$ outputs an input $x \in \{0,1\}^*$, consider an experiment where $K \xleftarrow{\$} \{0,1\}^\lambda$ and $K_x \leftarrow PRF(K,x)$. Then for all sufficiently large $\lambda \in \mathbb{N}$, for a negligible function μ,

$$\left|\mathsf{Pr}[\mathcal{A}_2(K_x, x, PRF_K(x)) = 1] - Pr[\mathcal{A}_2(K_x, x, U_{\chi(\lambda)}) = 1]\right| \leq \mu(\lambda)$$

  where $U_{\chi(\lambda)}$ is a string drawn uniformly at random from $\{0,1\}^{\chi(\lambda)}$.

As observed by [BW13, BGI14, KPTZ13], the GGM construction [GGM86] of PRFs from one-way functions yields puncturable PRFs.

Theorem 6 ([GGM86, BW13, BGI14, KPTZ13])**.** *If μ-secure one-way functions[9] exist, then for a polynomial $\chi(\lambda)$, there exists a μ-secure puncturable PRF family mapping $\{0,1\}^*$ to $\{0,1\}^{\chi(\lambda)}$.*

3.2 Witness Encryption

We recall the notion of witness encryption as defined by [GGSW13].

Definition 1 (Witness encryption [GGSW13, GLW14])**.** *A witness encryption scheme for an* NP *language L (with a corresponding relation R) consists of the following two polynomial-time algorithms:*

> $\mathsf{WE.Enc}(1^\lambda, x, m)$*: Takes as input a security parameter 1^λ, a string x and a message m, and outputs a ciphertext* CT*.*
>
> $\mathsf{WE.Dec}(\mathsf{CT}, w)$*: Takes as input a ciphertext* CT *and a string w, and outputs a message m or the symbol $\perp$.*

These algorithms satisfy the following two conditions:

1. ***Almost Perfect Correctness:*** *For any security parameter λ, any $m \in \{0,1\}^*$, any $x \in L$ and $w \in \{0,1\}^*$ such that $R(x,w)$ holds, we have that*

$$\Pr[\mathsf{WE.Dec}(\mathsf{WE.Enc}(1^\lambda, x, m), w) = m] \geq 1 - 2^{-\lambda}.$$

[9] We say that a one-way function family is μ-secure if the probability of inverting a one-way function, that is sampled from the family, is at most $\mu(\lambda)$.

2. ***Security:*** *For any probabilistic polynomial-time adversary A and any polynomial $p(\cdot)$, there exists a negligible function $\mathsf{negl}(\cdot)$, such that for any $\lambda \in \mathbb{N}$, any $x \notin L$ and any two equal-length messages m_1 and m_2 such that $|x|, |m_1| \leq p(\lambda)$, we have that*

$$\left|\Pr[A(\mathsf{WE.Enc}(1^\lambda, x, m_1)) = 1] - \Pr[A(\mathsf{WE.Enc}(1^\lambda, x, m_2)) = 1]\right| \leq \mathsf{negl}(\lambda).$$

We say that a WE scheme is parametrized by a polynomial T if both encryption and decryption algorithms run in time at most T.

3.3 NIZK with Pre-Processing

We consider a specific type of zero knowledge proof system where the messages exchanged is independent of the input instance till the last round. We call this zero knowledge proof system with pre-processing. The pre-processing algorithm essentially simulates the interaction between the prover and the verifier till the last round and outputs views of the prover and the verifier.

Definition 2. *Let L be a language with relation R. A scheme $\mathsf{PZK} = (\mathsf{PZK.Pre}, \mathsf{PZK.Prove}, \mathsf{PZK.Verify})$ of PPT algorithms is a zero knowledge proof system with pre-processing, PZK, between a verifier and a prover if they satisfy the following properties. Let $(\sigma_V, \sigma_P) \leftarrow \mathsf{PZK.Pre}(1^\lambda)$ be a preprocessing stage where the prover and the verifier interact. Then:*

1. *Completeness: there exists a polynomial r such that for every $(x, w) \in R$ we have that:*

$$\Pr\left[\mathsf{PZK.Verify}(\sigma_V, x, \pi) = 1 \ : \ \pi \leftarrow \mathsf{PZK.Prove}(\sigma_P, x, w)\right] = 1.$$

   where the probability is over the internal randomness of all the PZK algorithms.

2. *Soundness: for every $x \notin L$ we have that:*

$$\Pr[\exists \pi : \mathsf{PZK.Verify}(\sigma_V, x, \pi) = 1] < 2^{-n}$$

   where the probability is only over $\mathsf{PZK.Pre}$.

3. *Zero-Knowledge: there exists a PPT algorithm S such that for any x, w where $V(x, w) = 1$ it hold that:*

$$\{\sigma_V, \mathsf{PZK.Prove}(\sigma_P, x, w)\} \approx_c \{S(x)\}$$

Such schemes were studied in [DMP88, LS90] where they proposed constructions based on one-way functions.

3.4 Commitment Schemes

We consider commitment schemes secure in the CRS model. We give the definition of this primitive below.

Definition 3 (Commitment scheme in the CRS model)**.** *A polynomial-time computable function $\mathsf{Com}\colon \{0,1\}^n \times \{0,1\}^\lambda \times \{0,1\}^m \to \{0,1\}^*$, where $n = \mathsf{poly}(\lambda)$ is the length of the string to commit, λ is the length of the randomness, $m = \mathsf{poly}(\lambda)$ is the length of the CRS. We say that Com is a (non-interactive perfectly binding) commitment scheme in the CRS model if for any two inputs $x_1, x_2 \in \{0,1\}^n$ such that $x_1 \neq x_2$ it holds that:*

1. Computational Hiding: *Let* $\mathsf{crs} \leftarrow \{0,1\}^m$ *be chosen uniformly at random. The random variables* $\mathsf{Com}(x_1, \mathbf{U}_\lambda, \mathsf{crs})$ *and* $\mathsf{Com}(x_2, \mathbf{U}_\lambda, \mathsf{crs})$ *are computationally indistinguishable (given* crs*).*

2. Perfect Binding: *With all but negligible probability over the CRS, the supports of the above random variables are disjoint.*

Commitment schemes that satisfy the above definition, in the CRS model, can be constructed based on any pseudorandom generator [Nao91] (which can be based on any one-way functions [HILL99]). For simplicity, throghout the paper we ignore the CRS and simply write $\mathsf{Com}(\cdot,\cdot)$. We say that $\mathsf{Com}(x,r)$ is the commitment of the value x with the opening r.

3.5 Threshold Multi-key FHE

We recall the definition of multi-key fully homomorphic encryption from [LTV12, CM15, MW16]. A $\mu-$threshold multi-key FHE scheme is a tuple of algorithms $\mathsf{TMFHE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Expand}, \mathsf{FHEEval}, \mathsf{Dec}, \mathsf{PartDec}, \mathsf{FinDec})$ described as follows:

- $\mathsf{params} \leftarrow \mathsf{Setup}(1^\lambda, 1^d)$. The setup algorithm takes as input the security parameter λ, circuit depth d and outputs the system parameter params. We assume that all algorithms take params as input implicitly.

- $(sk, pk) \leftarrow \mathsf{KeyGen}(\mathsf{params})$. Keygen algorithm takes as input the system parameters and outputs a key pair.

- $\mathsf{CT} \leftarrow \mathsf{Enc}(pk, m)$. The encryption algorithm takes as input the public key pk, some message m and outputs a ciphertext CT.

- $\widehat{CT} \leftarrow \mathsf{Expand}\big((pk_1, .., pk_N), i, \mathsf{CT}\big)$. Given a sequence of N public-keys and a fresh cipher-text CT under the $i-$th key pk_i, it outputs an expanded ciphertext $\widehat{CT}$.

- $\mathsf{CT}_{eval} \leftarrow \mathsf{FHEEval}(\mathsf{params}, C, \widehat{\mathsf{CT}_1}, ..., \widehat{\mathsf{CT}_l})$. On input params, a circuit C of depth $\leq d$ and expanded ciphertexts $(\widehat{\mathsf{CT}_1}, ..., \widehat{\mathsf{CT}_l})$ the $\mathsf{FHEEval}$ algorithm outputs an evaluated ciphertext CT_{eval}.

- $m \leftarrow \mathsf{Dec}(\mathsf{params}, (sk_1, .., sk_N), \widehat{\mathsf{CT}})$. On input an expanded ciphertext $\widehat{\mathsf{CT}}$ and a sequence of secret keys $(sk_1, .., sk_N)$ the decryption algorithm outputs a message m.

- $p_i \leftarrow \mathsf{PartDec}(\widehat{\mathsf{CT}}, (pk_1, .., pk_N), i, sk_i)$. On input an expanded ciphertext $\widehat{\mathsf{CT}}$ under a sequence of N keys and the $i-th$ secret key output a partial decryption p_i. Note that this is a randomized procedure.

- $m \leftarrow \mathsf{FinDec}(p_1, .., p_N)$. On input N partial decryptions p_i for $i \in [N]$ output the plaintext m.

We require following properties from the scheme.

Semantic Security of encryption: For any polynomial $d = d(\lambda)$ and any two messages of equal length m_0, m_1 the following distributions are computationally indistinguishable:

$$\big(\mathsf{params}, pk, \mathsf{Enc}(pk, m_0)\big) \approx \big(\mathsf{params}, pk, \mathsf{Enc}(pk, m_1)\big)$$

where $\mathsf{params} \leftarrow \mathsf{Setup}(1^\lambda, 1^d)$, $(sk, pk) \leftarrow \mathsf{KeyGen}(\mathsf{params})$.

Correctness and Compactness: Let $\mathsf{params} \leftarrow \mathsf{Setup}(1^\lambda, 1^d)$. Consider any sequences of N correctly generated key pairs $\{(pk_i, sk_i) \leftarrow \mathsf{KeyGen}(\mathsf{params})\}_{i\in[N]}$ and any l-tuple of messages $(m_1, .., m_l)$. For any sequence of indices $(I_1, .., I_l)$ where each $I_i \in [N]$ let $\{\mathsf{CT}_i \leftarrow \mathsf{Enc}(pk_{I_i}, m_i)\}_{i\in[l]}$ and $\{\widehat{\mathsf{CT}_i} \leftarrow \mathsf{Expand}\big((pk_1, .., pk_N), I_i, \mathsf{CT}_i\big)\}_{i\in[l]}$ be the corresponding expanded ciphertexts. Let C be a boolean circuit of depth $\leq d$ and $\widehat{\mathsf{CT}} = \mathsf{FHEEval}\left(C, \widehat{\mathsf{CT}_{I_1}}, \ldots, \widehat{\mathsf{CT}_{I_l}}\right)$ be the evaluated ciphertext. Then the following holds:

Correctness of Expansion: $\forall i \in [l]$ $\mathsf{Dec}\big(\mathsf{params}, (sk_1, .., sk_N), \widehat{\mathsf{CT}_i}\big) = m_i$.

Correctness of Evaluation: $\mathsf{Dec}\big((sk_1, .., sk_N), \widehat{\mathsf{CT}}\big) = C(m_1, .., m_l)$.

Compactness: There exists a polynomial $p()$ such that $|\widehat{\mathsf{CT}}| \leq p(\lambda, d, N)$. In other words the size of $\widehat{\mathsf{CT}}$ should be independent of C and l, but can depend on λ, d, N.

Correctness of Part and Final decryption: It holds that

$$\mathsf{FinDec}(p_1, \ldots, p_N) = C(m_1, .., m_l),$$

where $\{p_i \leftarrow \mathsf{PartDec}(\widehat{\mathsf{CT}}, (pk_1, .., pk_N), i, sk_i)\}_{i\in[N]}$ are partial decryptions.

Simultability of partial decryption: There exists a PPT simulator Sim which, on input an index $i \in [N]$ and all but the $i-th$ keys $\{sk_j\}_{i\in[N]\setminus i}$ the evaluated ciphertext $\widehat{\mathsf{CT}}$ and the output message $m = C(m_1, .., m_N)$ produces a simulated partial decryption $p_i' \leftarrow \mathsf{Sim}(m, \widehat{ct}, i, \{sk_j\}_{i\in[N]\setminus i})$ such that

$$p_i \approx_\mu p_i'$$

where $p_i \leftarrow \mathsf{PartDec}(\widehat{\mathsf{CT}}, (pk_1, .., pk_N), i, sk_i)$. Note that the randomness is only over the random coins of the simulator and the $\mathsf{PartDec}$ procedure and all other values are assumed to be fixed (and known). Here the indistinguishability is statistical. We call a TMFHE scheme $\mu-$secure if the statistical distance between the distributions is bounded by μ. Note that in the construction of [MW16] the value μ is $O(2^{-\lambda^{\Omega(1)}})$.

4 Indistinguishability Obfuscation (IO) Combiners

Suppose we have many indistinguishability obfuscation (IO) schemes, also referred to as *IO candidates*). We are additionally guaranteed that one of the candidates is secure. No guarantee is placed

on the rest of the candidates and they could all be potentially broken. Indistinguishability obfuscation combiners provides a mechanism of combining all these candidates into a single monolithic IO scheme *that is secure*. We emphasize that the only guarantee we are provided is that one of the candidates is secure and in particular, it is unknown exactly which of the candidates is secure. If we knew which candidate is secure then achieving an IO combiner is straightforward – the IO combiner will just be the secure candidate and the rest of the candidates are discarded.

We give a thorough formal treatment of the concept of IO combiners next. We start by providing the syntax of an obfuscation scheme. We then present the definitions of an IO candidate and a secure IO candidate. Once we have set up these notions, in Section 4.1 we finally present the definition of IO combiner.

Syntax of Obfuscation Scheme. An obfuscation scheme associated to a class of circuits $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ consists of two PPT algorithms $(\mathsf{Obf}, \mathsf{Eval})$ defined below.

- **Obfuscate,** $\overline{C} \leftarrow \mathsf{Obf}(1^\lambda, C)$: It takes as input security parameter λ, a circuit $C \in \mathcal{C}_\lambda$ and outputs an obfuscation of C, $\overline{C}$.

- **Evaluation,** $y \leftarrow \mathsf{Eval}\left(\overline{C}, x\right)$: This is a deterministic algorithm. It takes as input an obfuscation $\overline{C}$, input $x \in \{0,1\}^\lambda$ and outputs y.

Throughout this work, we will only be concerned with *uniform* Obf algorithms. That is, Obf and Eval are represented as Turing machines (or equivalently uniform circuits).

μ-Correct IO candidate. We define the notion of an IO candidate below. The following definition of obfuscation scheme incorporates only the correctness and polynomial slowdown properties of an indistinguishability obfuscation scheme [BGI+01, GR07, GGH+13b]. In particular, an IO candidate need not necessarily have any security property associated with it. Formally,

Definition 4 (μ-Correct IO candidate)**.** *An obfuscation scheme* $\Pi = (\mathsf{Obf}, \mathsf{Eval})$ *is an* **μ-correct IO candidate** *for a class of circuits* $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$*, with every* $C \in \mathcal{C}_\lambda$ *has size* $\mathrm{poly}(\lambda)$*, if it satisfies the following properties:*

- **Correctness***: For every* $C : \{0,1\}^\lambda \to \{0,1\} \in \mathcal{C}_\lambda, x \in \{0,1\}^\lambda$ *it holds that:*

$$\mathsf{Pr}\left[\mathsf{Eval}\left(\mathsf{Obf}(1^\lambda, C), x\right) = C(x)\right] \geq \mu(\lambda),$$

  over the random coins of Obf*.*

- **Polynomial Slowdown***: For every* $C : \{0,1\}^\lambda \to \{0,1\} \in \mathcal{C}_\lambda$*, we have the running time of* Obf *on input* $(1^\lambda, C)$ *to be* $\mathrm{poly}(|C|, \lambda)$*. Similarly, we have the running time of* Eval *on input* $(\overline{C}, x)$ *is* $\mathrm{poly}(|\overline{C}|, \lambda)$*.*

Note that an identity function I is a valid IO candidate. We make use of this fact later on.

Remark 1. *We say that* Π *is an IO candidate if it is a* μ*-correct IO candidate with* $\mu = 1$*.*

μ-Correct ϵ-Secure IO candidate. If any IO candidate additionally satisfies the following (informal) security property then we define it to be a *secure* IO candidate: for every pair of circuits C_0 and C_1 that are equivalent[10] we have obfuscations of C_0 and C_1 to be indistinguishable by any PPT adversary.

Definition 5 (μ-Correct ϵ-Secure IO candidate)**.** *An obfuscation scheme $\Pi = (\mathsf{Obf}, \mathsf{Eval})$ for a class of circuits $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ is a* **μ-correct ϵ-secure IO candidate** *if it satisfies the following conditions:*

- Π *is an μ-correct IO candidate with respect to $\mathcal{C}$,*
- **Security.** *For every PPT adversary $\mathcal{A}$, for every sufficiently large $\lambda \in \mathbb{N}$, for every $C_0, C_1 \in \mathcal{C}_\lambda$ with $C_0(x) = C_1(x)$ for every $x \in \{0,1\}^\lambda$ and $|C_0| = |C_1|$, we have:*

$$\left|\mathsf{Pr}\left[0 \leftarrow \mathcal{A}\Big(\mathsf{Obf}(1^\lambda, C_0), C_0, C_1\Big)\right] - \left[0 \leftarrow \mathcal{A}\Big(\mathsf{Obf}(1^\lambda, C_1), C_0, C_1\Big)\right]\right| \leq \epsilon(\lambda)$$

We remarked earlier that identity function is an IO candidate. However, note that the identity function is *not* a secure IO candidate.

Remark 2. *We say that Π is a secure IO candidate if it is a μ-correct ϵ-secure IO candidate with $\mu = 1$ and $\epsilon(\lambda) = \mathsf{negl}(\lambda)$, for some negligible function negl.*

In the literature [GGH+13b, SW14], a secure IO candidate is simply referred to as an indistinguishability obfuscation scheme.

We have the necessary ingredients to define an IO combiner.

4.1 Definition of IO Combiner

We present the formal definition of IO combiner below. First, we provide the syntax of the IO combiner. Later we present the properties associated with an IO combiner.

There are two PPT algorithms associated with an IO combiner, namely, $\mathsf{CombObf}$ and $\mathsf{CombEval}$. Procedure $\mathsf{CombObf}$ takes as input circuit C along with the description of multiple IO candidates[11] and outputs an obfuscation of C. Procedure $\mathsf{CombEval}$ takes as input the obfuscated circuit, input x, the description of the candidates and outputs the evaluation of the obfuscated circuit on input x.

Syntax of IO Combiner. We define an IO combiner $\Pi_{\mathsf{comb}} = (\mathsf{CombObf}, \mathsf{CombEval})$ for a class of circuits $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$.

- **Combiner of Obfuscate algorithms,** $\overline{C} \leftarrow \mathsf{CombObf}(1^\lambda, C, \Pi_1, \ldots, \Pi_n)$: It takes as input security parameter λ, a circuit $C \in \mathcal{C}$, description of IO candidates $\{\Pi_i\}_{i\in[n]}$ and outputs an obfuscated circuit $\overline{C}$.
- **Combiner of Evaluation algorithms,** $y \leftarrow \mathsf{CombEval}(\overline{C}, x, \Pi_1, \ldots, \Pi_n)$: It takes as input obfuscated circuit $\overline{C}$, input x, descriptions of IO candidates $\{\Pi_i\}_{i\in[n]}$ and outputs y.

[10] Two circuits C_0 and C_1 are equivalent if they (a) have the same size, (b) have the same input domain and, (c) for every x in the input domain, $C_0(x) = C_1(x)$.

[11] The description of an IO candidate includes the description of the obfuscation and the evaluation algorithms.

We define the properties associated to any IO combiner. There are three main properties – correctness, polynomial slowdown and security. The correctness and the polynomial slowdown properties are defined on the same lines as the corresponding properties of the IO candidates.

The intuitive security notion of IO combiner says the following: suppose one of the candidates is a secure IO candidate then the output of obfuscator ($\mathsf{CombObf}$) of the IO combiner on C_0 is computationally indistinguishable from the output of the obfuscator on C_1, where C_0 and C_1 are equivalent circuits.

Definition 6 ((μ', μ)-correct (ϵ', ϵ)-secure IO combiner)**.** *Consider a circuit class $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$. We say that $\Pi_{\mathsf{comb}} = (\mathsf{CombObf}, \mathsf{CombEval})$ is a (μ', μ)-**correct** (ϵ', ϵ)-**secure IO combiner** if the following conditions are satisfied: Let $\Pi_1, \ldots, \Pi_n$ be n μ-correct IO candidates for P/poly, where μ is a function of μ' and ϵ is a function of ϵ'.*

- **Correctness.** *Let $C \in \mathcal{C}_{\lambda\in\mathbb{N}}$ and $x \in \{0,1\}^\lambda$. Consider the following process: (a) $\overline{C} \leftarrow \mathsf{CombObf}(1^\lambda, C, \Pi_1, \ldots, \Pi_n)$, (b) $y \leftarrow \mathsf{CombEval}(\overline{C}, x, \Pi_1, \ldots, \Pi_n)$.*

  Then, $\mathsf{Pr}[y = C(x)] \geq \mu'(\lambda)$ over the randomness of $\mathsf{CombObf}$.

- **Polynomial Slowdown.** *For every $C : \{0,1\}^\lambda \to \{0,1\} \in \mathcal{C}_\lambda$, we have the running time of $\mathsf{CombObf}$ on input $(1^\lambda, C, \Pi_1, \ldots, \Pi_n)$ to be at most $\mathsf{poly}(|C| + n + \lambda)$. Similarly, we have the running time of $\mathsf{CombEval}$ on input $(\overline{C}, x, \Pi_1, \ldots, \Pi_n)$ to be at most $\mathsf{poly}(|\overline{C}| + n + \lambda)$.*

- **Security.** *Let Π_i be ϵ-secure for some $i \in [n]$. For every PPT adversary $\mathcal{A}$, for every sufficiently large $\lambda \in \mathbb{N}$, for every $C_0, C_1 \in \mathcal{C}_\lambda$ with $C_0(x) = C_1(x)$ for every $x \in \{0,1\}^\lambda$ and $|C_0| = |C_1|$, we have:*

$$\Big|\mathsf{Pr}\Big[0 \leftarrow \mathcal{A}\Big(\overline{C_0}, C_0, C_1, \Pi_1, \ldots, \Pi_n\Big)\Big] - \Big[0 \leftarrow \mathcal{A}\Big(\overline{C_1}, C_0, C_1, \Pi_1, \ldots, \Pi_n\Big)\Big]\Big| \leq \epsilon'(\lambda),$$

  where $\overline{C_b} \leftarrow \mathsf{CombObf}(1^\lambda, C_b, \Pi_1, \ldots, \Pi_n)$ for $b \in \{0,1\}$.

Some remarks are in order.

Remark 3. *We say that Π_{comb} is an IO combiner if it is a (μ', μ)-correct (ϵ', ϵ)-secure IO combiner, where (a) $\mu' = 1$, (b) $\mu = 1$, (c) $\epsilon' = \mathsf{negl}'$ and, (d) $\epsilon = \mathsf{negl}$ with negl and negl' being negligible functions.*

Remark 4. *We alternatively call the IO combiner defined in Definition 6 to be a 1-out-n IO combiner. We can more generally define c-out-n IO combiners where the IO combiner combines n IO candidates securely as long as at least c candidates are secure. If c = n-1 then we can construct a (n-1)-out-n combiner as follows: given a circuit C, discard all but two candidates; obfuscate C using one of the candidates and re-obfuscate the resulting obfuscated circuit using the other candidate. In fact this approach can be generalized to (n-k)-out-n, where k is a constant. In this work we only focus on 1-out-n IO combiners.*

5 Constructions of IO Combiners

We propose constructions of combiners for indistinguishability obfuscation. We first present a construction based on the learning with errors assumption. In Appendix 5.2, we present a construction based on the decisional Diffie Hellman assumption.

5.1 LWE-Based Construction

We present the formal construction below. For an informal explanation of the construction, we refer the reader to Introduction.

Construction. Consider a circuit class $\mathcal{C}$. We use a threshold multi-key FHE scheme $\mathsf{TMFHE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Expand}, \mathsf{FHEEval}, \mathsf{Dec}, \mathsf{PartDec}, \mathsf{FinDec})$. We additionally use a puncturable PRF family $\mathcal{F}$.

We construct an IO combiner $\Pi_{\mathsf{comb}} = \Pi_{\mathsf{comb}}[\Pi_1, \ldots, \Pi_n]$ for $\mathcal{C}$ below.

$\underline{\mathsf{CombObf}(1^\lambda, C, \Pi_1, \ldots, \Pi_n)}$: It takes as input security parameter λ, circuit $C \in \mathcal{C}_\lambda$, description of candidates $\{\Pi_i = (\Pi_i.\mathsf{Obf}, \Pi_i, \mathsf{Eval})\}_{i\in[n]}$ and does the following.

1. **Initialization of TMFHE parameters:**
   - Execute the setup of the threshold multi-key FHE scheme, $\mathsf{params} \leftarrow \mathsf{Setup}(1^\lambda, 1^d)$, where $d = \mathrm{poly}(\lambda, |C|)$ [12]. Execute $\{(sk_i, pk_i) \leftarrow \mathsf{KeyGen}(\mathsf{params})\}_{i\in[n]}$.
   - Sample n random strings $\{S_i\}_{i\in[n]}$ of size $\mid C \mid$ such that $\bigoplus_{i\in[n]} S_i = C$.
   - For all $i \in [n]$, encrypt the string S_i using pk_i, $\mathsf{CT}_i \leftarrow \mathsf{Enc}(pk_i, S_i)$.
   - For every $i \in [n]$, generate the expanded ciphertext under pk_i by executing $\widehat{\mathsf{CT}_i} \leftarrow \mathsf{Expand}((pk_1, \ldots, pk_n), i, \mathsf{CT}_i)$.
2. **Obfuscating Circuits using IO candidates:**
   - For every $i \in [n]$, sample puncturable PRF keys $K^i \xleftarrow{\$} \{0,1\}^\lambda$.
   - For every $i \in [n]$, construct circuit $G_i = G_i\left[K^i, sk_i, \{pk_i\}_{i\in[n]}, \{\widehat{\mathsf{CT}_i}\}_{i\in[n]}\right] \in \mathcal{C}^i$ as described in Figure 1.
   - Generate $\overline{G_i} \leftarrow \Pi_i.\mathsf{Obf}(1^\lambda, G_i)$.

Output the obfuscation $\overline{C} = \left(\overline{G_1}, \ldots, \overline{G_n}\right)$.

$\underline{\mathsf{CombEval}(\overline{C}, x, \Pi_1, \ldots, \Pi_n)}$: On input an obfuscation $\overline{C}$, an input x, descriptions of candidates $\{\Pi_i\}_{i\in[n]}$ evaluate the obfuscations on input x to obtain $p_i \leftarrow \Pi_i.\mathsf{Eval}(\overline{G_i}, x)$ for all $i \in [n]$. Execute the final decryption algorithm, $y \leftarrow \mathsf{FinDec}(p_1, \ldots, p_n)$. Output y.

5.1.1 Correctness and Security of Π_{comb}

We prove the correctness and security of Π_{comb} below.

[12] Looking ahead, we set d to be the size of C as against its depth so that a PPT adversary will not be able to distinguish obfuscations of two functionally equivalent circuits C_0 and C_1 with the same size but potentially different depths by just measuring the size of params.

$$G_i\left[K^i, sk_i, \{pk_i\}_{i\in[n]}, \{\widehat{\mathsf{CT}_i}\}_{i\in[n]}\right]$$

Hardwired values: PRF key K^i, TMFHE partial decryption key sk_i, TMFHE public keys $\{pk_i\}_{i\in[n]}$, TMFHE expanded ciphertext $\{\widehat{\mathsf{CT}_i}\}_{i\in[n]}$.
Input: $x \in \{0,1\}^\lambda$.

- Perform $\widehat{\mathsf{CT}_{\mathsf{out}}} \leftarrow \mathsf{FHEEval}\left(\mathsf{params}, U_x(\cdot), \widehat{\mathsf{CT}_1}, \ldots, \widehat{\mathsf{CT}_n}\right)$, where $U_x(\cdot)$ is a universal circuit that takes as input n strings $S_1, .., S_n$ and first computes $\bigoplus_{j\in[n]} S_j = C$ where $C \in \mathcal{C}_\lambda$ and outputs $C(x)$.
- Generate randomness $r_i \leftarrow PRF_{K^i}(x)$.
- Execute the partial decryption algorithm, $p_i \leftarrow \mathsf{PartDec}\Big(\widehat{\mathsf{CT}_{\mathsf{out}}}, pk_1, \ldots, pk_n, i, sk_i;\ r_i\Big)$
- Output p_i.

Figure 1: Circuit G_i

Correctness. Consider a circuit $C \in \mathcal{C}_\lambda$ and let $x \in \{0,1\}^\lambda$. Let G_i, for every $i \in [n]$, be the circuit generated during the execution of $\mathsf{Obf}(1^\lambda, C, \Pi_1, \ldots, \Pi_n)$. Let $\overline{G_i} \leftarrow \Pi_i.\mathsf{Obf}(1^\lambda, G_i)$. Finally let $\overline{C} = (\overline{G_1} \ldots \overline{G_n})$ be the output of $\mathsf{CombObf}$.

We first prove the following lemma.

Lemma 1. *Suppose for every $i \in [n]$, $\mathsf{Pr}[\Pi_i.\mathsf{Eval}(\overline{G_i}, x) = G_i(x)] \geq 1 - \mu_i$ then we have $\mathsf{Pr}[\mathsf{CombEval}(\overline{C}, x) = C(x)] \geq \mu'$ with $\mu' = (1 - \sum_{i=1}^n \mu_i)$, assuming the perfect correctness of TMFHE.*

Proof. Suppose on an input x, $\Pi_i.\mathsf{Eval}(\overline{G_i}, x) = G_i(x)$ holds for every $i \in [n]$ then we have the corresponding output of $\mathsf{CombEval}$ to be $C(x)$. Thus we have,

$$\begin{aligned}
\mathsf{Pr}[\mathsf{CombEval}(\overline{C}, x) \neq C(x)] &= \mathsf{Pr}[\exists i \in [n] :\ \Pi_i.\mathsf{Eval}(\overline{G_i}, x) \neq G_i(x)] \\
&\leq \sum_{i=1}^n \mathsf{Pr}[\Pi_i.\mathsf{Eval}(\overline{G_i}, x) \neq G_i(x)] \text{ (union bound)} \\
&\leq \sum_{i=1}^n \mu_i
\end{aligned}$$

Thus, we have $\mathsf{Pr}[\mathsf{CombEval}(\overline{C}, x) \neq C(x)] \geq 1 - \sum_{i=1}^n \mu_i$. □

The proof of the following theorem follows directly from the above lemma.

Theorem 7. *If for all $i \in [n]$, Π_i is $(1 - \mu_i)$-correct, then $\Pi_{\mathsf{comb}} = \Pi_{\mathsf{comb}}[\Pi_1, \ldots, \Pi_n]$ is at least $(1 - \sum_{i=1}^n \mu_i)$-correct.*

Security. We now move on to proving the security of Π_{comb}.

Theorem 8. *Consider a circuit class $\mathcal{C} \in P/poly$. Assuming that $\mathcal{F}$ is a subexponentially secure puncturable PRFs and* TMFHE *is $2^{-2\lambda}$-secure threshold multi-key FHE, the scheme Π_{comb} is an $(2^{-2\lambda}, \mathsf{negl}(\lambda))-$secure IO combiner for $\mathcal{C}$.*

Proof. Consider a set of IO candidates $\Pi_1, \ldots, \Pi_n$. Let Π_{ind} be a secure IO candidate for some $\mathsf{ind} \in [n]$. We employ the standard hybrid argument to prove the theorem. Consider two circuits $C_0, C_1 \in \mathcal{C}_\lambda$ such that $C_0(x) = C_1(x)$ for all $x \in \{0,1\}^\lambda$. In the first hybrid (Hyb_1), the circuit C_b is obfuscated honestly with $b \xleftarrow{\$} \{0,1\}$. In the final hybrid ($\mathsf{Hyb}_4$), the obfuscated circuit the adversary has no information about b. At this point, the probability of guessing the bit b is exactly $1/2$. By arguing indistinguishability of every consecutive intermediate hybrids, we show that the probability of guessing b in the first hybrid is negligibly close to $1/2$ (or the advantage is 0), which proves the theorem.

In the following set of hybrids, denote $\mathcal{A}$ to be a PPT adversary.

$\underline{\mathsf{Hyb}_1}$: Pick a bit b at random. The adversary receives the obfuscation $\overline{C_b}$, where $\overline{C_b} \leftarrow \Pi_{\mathsf{comb}}.\mathsf{CombObf}(1^\lambda, C_b, \Pi_1, \ldots, \Pi_n)$. Let $\mathcal{A}$ output b'. Output of Hyb_1 is b'.

$\underline{\mathsf{Hyb}_{2.1.z} \text{ for } z \in \{0,1\}^\lambda}$: Pick a bit b at random. First generate the TMFHE parameters $\widehat{\mathsf{CT}_i}$ for every $i \in [n]$ as a function of circuit C_b. Sample puncturable PRF key $K^{\mathsf{ind}} \xleftarrow{\$} \{0,1\}^\lambda$. Execute $K^{\mathsf{ind}}_z \leftarrow \mathsf{Puncture}(K^{\mathsf{ind}}, z)$. Then generate p^z_{ind} as below.

- Perform $\widehat{\mathsf{CT}^z_{\mathsf{out}}} \leftarrow \mathsf{FHEEval}\left(\mathsf{params}, U_z(\cdot), \widehat{\mathsf{CT}_1}, \ldots, \widehat{\mathsf{CT}_n}\right)$ (where $U_z(\cdot)$ is a universal circuit that takes as input n strings $S_1, .., S_n$ and first computes $\bigoplus_{j\in[n]} S_j = C$ where $C \in \mathcal{C}_\lambda$ and outputs $C(z)$.)
- Generate randomness $r^z_{\mathsf{ind}} \leftarrow PRF_{K^{\mathsf{ind}}}(z)$.
- Execute the partial decryption algorithm, $p^z_{\mathsf{ind}} \leftarrow \mathsf{PartDec}\Big(\widehat{\mathsf{CT}_{\mathsf{out}}}, pk_1, \ldots, pk_n, \mathsf{ind}, sk_{\mathsf{ind}};\ r^z_{\mathsf{ind}}\Big)$.
- Set $v = p^z_{\mathsf{ind}}$.

Generate the obfuscation, $\overline{\mathsf{H}_1} \leftarrow \Pi_{\mathsf{ind}}.\mathsf{Obf}(1^\lambda, \mathsf{H}_1)$, where $\mathsf{H}_1 = \mathsf{H}_1\big[z, C_0, K^{\mathsf{ind}}_z, sk_{\mathsf{ind}}, \{sk_i\}_{i\in[n]\setminus\{\mathsf{ind}\}}, \{pk_i\}_{i\in[n]}, \{\widehat{\mathsf{CT}_i}\}_{i\in[n]}, v\big]$ is described in Figure 2. Set $\overline{G_{\mathsf{ind}}} = \overline{H_1}$. Generate the obfuscations $\overline{G_i} \leftarrow \Pi_{\mathsf{ind}}.\mathsf{Obf}(1^\lambda, G_i)$ for $i \neq \mathsf{ind}$ and $i \in [n]$ where G_i is designed as presented in the scheme. Set $\overline{C_b} = (\ \overline{G_1}, \ldots, \overline{G_n}\ )$.

The adversary receives $\overline{C_b}$ and outputs b'. Output of $\mathsf{Hyb}_{2.1.z}$ is b'.

$\underline{\mathsf{Hyb}_{2.2.z} \text{ for } z \in \{0,1\}^\lambda}$: Proceed as in the previous hybrid, except that r^z_{ind} is sampled at random from $\{0,1\}^{\chi(\lambda)}$.

$\underline{\mathsf{Hyb}_{2.3.z} \text{ for } z \in \{0,1\}^\lambda}$: Let the ciphertext $\widehat{\mathsf{CT}^z_{\mathsf{out}}}$, partial decryption keys $\{sk_i\}_{i\in[n]\setminus\{\mathsf{ind}\}}$ be defined as before. Execute $p^z_{\mathsf{ind}} \leftarrow \mathsf{Sim}\big(C_0(z), \widehat{\mathsf{CT}^z_{\mathsf{out}}}, \mathsf{ind}, \{sk_i\}_{i\in[n]\setminus\{\mathsf{ind}\}};\ r_{\mathsf{ind}}\big)$. Set $v = p^z_{\mathsf{ind}}$. The rest of the hybrid is as before.

$$\mathsf{H}_1\left[z, C_0, K_z^{\mathsf{ind}}, sk_{\mathsf{ind}}, \{sk_i\}_{i\in[n]\setminus\{\mathsf{ind}\}}, \{pk_i\}_{i\in[n]}, \{\widehat{\mathsf{CT}_i}\}_{i\in[n]}, v\right]$$

Hardwired values: PRF key K_z^{ind}, TMFHE partial decryption key sk_{ind}, rest of partial decryption keys $\{sk_i\}_{i\in[n]\setminus\{\mathsf{ind}\}}$, TMFHE public keys $\{pk_i\}_{i\in[n]}$, TMFHE expanded ciphertext $\{\widehat{\mathsf{CT}_i}\}_{i\in[n]}$, hardwired value v.
Input: $x \in \{0,1\}^\lambda$.

- If $x < z$:
  - Perform $\widehat{\mathsf{CT}_{\mathsf{out}}} \leftarrow \mathsf{FHEEval}\left(\mathsf{params}, U_x(\cdot), \widehat{\mathsf{CT}_1}, \ldots, \widehat{\mathsf{CT}_n}\right)$, where $U_x(\cdot)$ is a universal circuit that takes as input n strings $S_1, .., S_n$ and first computes $\bigoplus_{j\in[n]} S_j = C$ where $C \in \mathcal{C}_\lambda$ and outputs $C(x)$.
  - Generate randomness $r_{\mathsf{ind}} \leftarrow PRF_{K^{\mathsf{ind}}}(x)$.
  - Simulate the partial decryption, $p_{\mathsf{ind}} \leftarrow \mathsf{Sim}\Big(C_0(x), \widehat{\mathsf{CT}_{\mathsf{out}}}, \mathsf{ind}, \{sk_i\}_{i\in[n]\setminus\{\mathsf{ind}\}};\ r_{\mathsf{ind}}\Big)$.
  - Output p_{ind}.
- If $x = z$ then output v.
- If $x > z$:
  - Perform $\widehat{\mathsf{CT}_{\mathsf{out}}} \leftarrow \mathsf{FHEEval}\left(\mathsf{params}, U_x(\cdot), \widehat{\mathsf{CT}_1}, \ldots, \widehat{\mathsf{CT}_n}\right)$, where $U_x(\cdot)$ is a universal circuit that takes as input n strings $S_1, .., S_n$ and first computes $\bigoplus_{j\in[n]} S_j = C$ where $C \in \mathcal{C}_\lambda$ and outputs $C(x)$.
  - Generate randomness $r_{\mathsf{ind}} \leftarrow PRF_{K_z^{\mathsf{ind}}}(x)$.
  - Execute the partial decryption algorithm, $p_{\mathsf{ind}} \leftarrow \mathsf{PartDec}\Big(\widehat{\mathsf{CT}_{\mathsf{out}}}, pk_1, \ldots, pk_n, \mathsf{ind}, sk_{\mathsf{ind}};\ r_{\mathsf{ind}}\Big)$.
  - Output p_{ind}.

Figure 2: Circuit H_1

$\underline{\mathsf{Hyb}_{2.4.z} \text{ for } z \in \{0,1\}^\lambda}$: Proceed as in the previous hybrid, except that r^z_{ind} is set to be $PRF_{K^{\mathsf{ind}}}(z)$.

$\underline{\mathsf{Hyb}_{2.5.z} \text{ for } z \in \{0,1\}^\lambda}$: This is same as the previous hybrid except for the following: Generate the obfuscation, $\overline{\mathsf{H}_2} \leftarrow \Pi_{\mathsf{ind}}.\mathsf{Obf}(1^\lambda, \mathsf{H}_2)$, where $\mathsf{H}_2 = \mathsf{H}_2\big[z, C_0, K_z^{\mathsf{ind}}, sk_{\mathsf{ind}}, \{sk_i\}_{i\in[n]\setminus\{\mathsf{ind}\}}, \{pk_i\}_{i\in[n]}, \{\widehat{\mathsf{CT}_i}\}_{i\in[n]}\big]$ is described in Figure 3. Set $\overline{G_{\mathsf{ind}}} = \overline{H_2}$. Generate the obfuscations $\overline{G_i} \leftarrow \Pi_{\mathsf{ind}}.\mathsf{Obf}(1^\lambda, G_i)$ for $i \neq \mathsf{ind}$ and $i \in [n]$ where G_i is designed as presented in the scheme. Set $\overline{C_b} = \big(\ \overline{G_1}, \ldots, \overline{G_n}\ \big)$.

$\underline{\mathsf{Hyb}_3}$: This is same as the previous hybrid except for the following: Generate the obfuscation, $\overline{\mathsf{H}_3} \leftarrow \Pi_{\mathsf{ind}}.\mathsf{Obf}(1^\lambda, \mathsf{H}_3)$, where $\mathsf{H}_3 = \mathsf{H}_3\big[C_0, K^{\mathsf{ind}}, \{sk_i\}_{i\in[n]\setminus\{\mathsf{ind}\}}, \{pk_i\}_{i\in[n]}, \{\widehat{\mathsf{CT}_i}\}_{i\in[n]}\big]$ is described in Figure 4 (here K^{ind} is the actual PRF key). Set $\overline{G_{\mathsf{ind}}} = \overline{H_3}$. Generate the obfuscations $\overline{G_i} \leftarrow \Pi_{\mathsf{ind}}.\mathsf{Obf}(1^\lambda, G_i)$ for $i \neq \mathsf{ind}$ and $i \in [n]$ where G_i is designed as presented in the scheme. Set

$$\mathsf{H}_2\left[z, C_0, K_z^{\mathsf{ind}}, sk_{\mathsf{ind}}, \{sk_i\}_{i\in[n]\setminus\{\mathsf{ind}\}}, \{pk_i\}_{i\in[n]}, \{\widehat{\mathsf{CT}}_i\}_{i\in[n]}\right]$$

Hardwired values: PRF key K_z^{ind}, TMFHE partial decryption key sk_{ind}, rest of partial decryption keys $\{sk_i\}_{i\in[n]\setminus\{\mathsf{ind}\}}$, TMFHE public keys $\{pk_i\}_{i\in[n]}$, TMFHE expanded ciphertext $\{\widehat{\mathsf{CT}}_i\}_{i\in[n]}$.
Input: $x \in \{0,1\}^\lambda$.

- If $x \leq z$:
  - Perform $\widehat{\mathsf{CT}_{\mathsf{out}}} \leftarrow \mathsf{FHEEval}\left(\mathsf{params}, U_x(\cdot), \widehat{\mathsf{CT}}_1, \ldots, \widehat{\mathsf{CT}}_n\right)$, where $U_x(\cdot)$ is a universal circuit that takes as input n strings $S_1, .., S_n$ and first computes $\bigoplus_{j\in[n]} S_j = C$ where $C \in \mathcal{C}_\lambda$ and outputs $C(x)$.
  - Generate randomness $r_{\mathsf{ind}} \leftarrow PRF_{K^{\mathsf{ind}}}(x)$.
  - Simulate the partial decryption, $p_{\mathsf{ind}} \leftarrow \mathsf{Sim}\Big(C_0(x), \widehat{\mathsf{CT}_{\mathsf{out}}}, \mathsf{ind}, \{sk_i\}_{i\in[n]\setminus\{\mathsf{ind}\}};\ r_{\mathsf{ind}}\Big)$.
  - Output p_{ind}.
- If $x > z$:
  - Perform $\widehat{\mathsf{CT}_{\mathsf{out}}} \leftarrow \mathsf{FHEEval}\left(\mathsf{params}, U_x(\cdot), \widehat{\mathsf{CT}}_1, \ldots, \widehat{\mathsf{CT}}_n\right)$, where $U_x(\cdot)$ is a universal circuit that takes as input n strings $S_1, .., S_n$ and first computes $\bigoplus_{j\in[n]} S_j = C$ where $C \in \mathcal{C}_\lambda$ and outputs $C(x)$.
  - Generate randomness $r_{\mathsf{ind}} \leftarrow PRF_{K_z^{\mathsf{ind}}}(x)$.
  - Execute the partial decryption algorithm, $p_{\mathsf{ind}} \leftarrow \mathsf{PartDec}\Big(\widehat{\mathsf{CT}_{\mathsf{out}}}, pk_1, \ldots, pk_n, \mathsf{ind}, sk_{\mathsf{ind}};\ r_{\mathsf{ind}}\Big)$.
  - Output p_{ind}.

Figure 3: Circuit H_2

$\overline{C_b} = (\ \overline{G_1}, \ldots, \overline{G_n}\ )$.

Hyb_4: This hybrid is the same as the previous hybrid except that $\mathsf{CT}_{\mathsf{ind}}$ is computed as an encryption of $\overline{S_{\mathsf{ind}}} \oplus C_b \oplus C_0$. In this hybrid, $S_1, .., S_n$ form a xor secret sharing of C_0. This hybrid is independent of b, hence the advantage of adversary in this hybrid is exactly 0.

Indistinguishability of Hybrids: We prove the indistinguishability of hybrids next.

Notation: We begin with some notation. Let us assume that the secure candidate Π_{ind} be $\delta_{io}-$secure and the PRF be $\delta_{prf}-$secure. Also let TMFHE be $\mu-$secure with polynomial security.

Lemma 2. *If Π_{ind} is a δ_{io}-secure IO candidate then for any PPT adversary $\mathcal{A}$, it holds that $\Big|Pr\big[\mathcal{A}(\mathsf{Hyb}_1) = 1\big] - Pr\big[\mathcal{A}(\mathsf{Hyb}_{2.1.0}) = 1\big]\Big| < \delta_{io}$.*

Proof. The only difference in Hyb_1 and $\mathsf{Hyb}_{2.1.0}$ is the manner in which $\overline{G_{\mathsf{ind}}}$ is generated. In Hyb_1

$$\mathsf{H}_3\left[C_0, K^{\mathsf{ind}}, \{sk_i\}_{i\in[n]\setminus\{\mathsf{ind}\}}, \{pk_i\}_{i\in[n]}, \{\widehat{\mathsf{CT}_i}\}_{i\in[n]}\right]$$

Hardwired values: PRF key K_z^{ind}, TMFHE partial decryption keys $\{sk_i\}_{i\in[n]\setminus\{\mathsf{ind}\}}$, TMFHE public keys $\{pk_i\}_{i\in[n]}$, TMFHE expanded ciphertext $\{\widehat{\mathsf{CT}_i}\}_{i\in[n]}$.
Input: $x \in \{0,1\}^\lambda$.

- Perform $\widehat{\mathsf{CT}_{\mathsf{out}}} \leftarrow \mathsf{FHEEval}\left(\mathsf{params}, U_x(\cdot), \widehat{\mathsf{CT}_1}, \ldots, \widehat{\mathsf{CT}_n}\right)$, where $U_x(\cdot)$ is a universal circuit that takes as input n strings $S_1, .., S_n$ and first computes $\bigoplus_{j\in[n]} S_j = C$ where $C \in \mathcal{C}_\lambda$ and outputs $C(x)$.
- Generate randomness $r_{\mathsf{ind}} \leftarrow PRF_{K^{\mathsf{ind}}}(x)$.
- Simulate the partial decryption, $p_{\mathsf{ind}} \leftarrow \mathsf{Sim}\left(C_0(x), \widehat{\mathsf{CT}_{\mathsf{out}}}, \mathsf{ind}, \{sk_i\}_{i\in[n]\setminus\{\mathsf{ind}\}};\ r_{\mathsf{ind}}\right)$.
- Output p_{ind}.

Figure 4: Circuit H_3

it is generated as an obfuscation of circuit H_0 as done in a genuine obfuscation of C_b.
In $\mathsf{Hyb}_{2.1.0}$ it is generated as follows: $\overline{\mathsf{H}_1} \leftarrow \Pi_{\mathsf{ind}}.\mathsf{Obf}(1^\lambda, \mathsf{H}_1)$, where $\mathsf{H}_1 = \mathsf{H}_1\big[z = 0, C_0, K_z^{\mathsf{ind}}, sk_{\mathsf{ind}}, \{sk_i\}_{i\in[n]\setminus\{\mathsf{ind}\}}, \{pk_i\}_{i\in[n]}, \{\widehat{\mathsf{CT}_i}\}_{i\in[n]}, v\big]$ is described in Figure 2. $\overline{G_{\mathsf{ind}}}$ is set as $\overline{H_1}$. Here v is set as in $\mathsf{Hyb}_{2.1.0}$.
Note that H_1 instantiated with $(z = 0, v)$ is functionally equivalent to H_0. This is because the only point at which these two circuits may differ is the input $z = 0^\lambda$. Because at all points $x > 0$, both circuits evaluate the same code except that H_0 uses a PRF key K^{ind} while H_1 uses a punctured PRF key (punctured at 0^λ). Since the puncturable PRF is correct, the evaluation at all points $x > 0$ guarantees same output.
At this input $z = 0^\lambda$, H_1 outputs the hard coded value v. By the definition of hybrid $\mathsf{Hyb}_{2.1.0}$, v is set to be equal to the output of H_0 at 0^λ which causes two circuits to be functionally equivalent.
The security now follows from the security of indistinguishibility obfuscator candidate Π_{ind}. □

Lemma 3. *If puncturable PRF is δ_{prf}-secure then for any PPT adversary $\mathcal{A}$ and $z \in \{0,1\}^\lambda$, it holds that* $\left|Pr\big[\mathcal{A}(\mathsf{Hyb}_{2.1.z}) = 1\big] - Pr\big[\mathcal{A}(\mathsf{Hyb}_{2.2.z}) = 1\big]\right| < \delta_{prf}$.

Proof. Note that in both these hybrids $\mathsf{Hyb}_{2.1.z}$ and $\mathsf{Hyb}_{2.2.z}$, K^{ind} is absent and both these hybrids depend only on the key punctured at z, K_z^{ind} and a string r_z^{ind} (evaluation of the PRF at z). In $\mathsf{Hyb}_{2.2.z}$, r_z^{ind} is sampled randomly while in $\mathsf{Hyb}_{2.1.z}$ it is generated as the evaluation of the PRF using the (unpunctured) key K^{ind} at z.
The security now follows from the security of δ_{prf}-secure puncturable PRF at the punctured point z. □

Lemma 4. *If* TMFHE *is μ-secure then for any PPT adversary $\mathcal{A}$ and $z \in \{0,1\}^\lambda$, it holds that* $\left|Pr\big[\mathcal{A}(\mathsf{Hyb}_{2.2.z}) = 1\big] - Pr\big[\mathcal{A}(\mathsf{Hyb}_{2.3.z}) = 1\big]\right| < \mu$.

Proof. The only difference between $\mathsf{Hyb}_{2.2.z}$ and $\mathsf{Hyb}_{2.3.z}$ is the manner in which the hard coded value v is generated. In $\mathsf{Hyb}_{2.2.z}$ it is generated by executing the partial decryption algorithm,

$p^z_{\mathsf{ind}} \leftarrow \mathsf{PartDec}\Big(\widehat{\mathsf{CT}_{\mathsf{out}}}, pk_1, \ldots, pk_n, \mathsf{ind}, sk_{\mathsf{ind}};\ r^z_{\mathsf{ind}}\Big)$ using uniformly sampled randomness r^z_{ind}. v is then set as p^z_{ind}.
In $\mathsf{Hyb}_{2.3.z}$ it is generated by executing $p^z_{\mathsf{ind}} \leftarrow \mathsf{Sim}\big(C_0(z), \widehat{\mathsf{CT}_{\mathsf{out}}}, \mathsf{ind}, \{sk_i\}_{i\in[n]\setminus\{\mathsf{ind}\}};\ r_{\mathsf{ind}}\big)$. v is then set as p^z_{ind}.
The security now follows from the statistical closeness of the simulated share for the index ind from the shares generated by the partial decryption algorithm. □

Lemma 5. *If puncturable PRF is δ_{prf}-secure then for any PPT adversary $\mathcal{A}$ and $z \in \{0,1\}^\lambda$, it holds that* $\Big|Pr\big[\mathcal{A}(\mathsf{Hyb}_{2.3.z}) = 1\big] - Pr\big[\mathcal{A}(\mathsf{Hyb}_{2.4.z}) = 1\big]\Big| < \delta_{prf}$.

Proof. Note that in both these hybrids $\mathsf{Hyb}_{2.3.z}$ and $\mathsf{Hyb}_{2.4.z}$, PRF key K^{ind} is absent and both these hybrids depend only on the key punctured at z, K^{ind}_z and a string r^{ind}_z. In $\mathsf{Hyb}_{2.3.z}$ r^{ind}_z is sampled randomly while in $\mathsf{Hyb}_{2.4.z}$ it is generated as the evaluation of the PRF using the (unpunctured) key K^{ind} at z.
As the puncturable PRF is δ_{io}-secure, the lemma follows. □

Lemma 6. *If Π_{ind} is a δ_{io}-secure IO candidate then for any PPT adversary $\mathcal{A}$ and $z \in \{0,1\}^\lambda$, it holds that* $\Big|Pr\big[\mathcal{A}(\mathsf{Hyb}_{2.4.z}) = 1\big] - Pr\big[\mathcal{A}(\mathsf{Hyb}_{2.5.z}) = 1\big]\Big| < \delta_{io}$.

Proof. The only difference between $\mathsf{Hyb}_{2.5.z}$ and $\mathsf{Hyb}_{2.4.z}$ is the manner in which $\overline{G_{\mathsf{ind}}}$ is generated. In $\mathsf{Hyb}_{2.5.z}$ it is generated as follows: $\overline{\mathsf{H}_2} \leftarrow \Pi_{\mathsf{ind}}.\mathsf{Obf}(1^\lambda, \mathsf{H}_2)$, where $\mathsf{H}_2 = \mathsf{H}_2\big[z, C_0, K^{\mathsf{ind}}, sk_{\mathsf{ind}}, \{sk_i\}_{i\in[n]\setminus\{\mathsf{ind}\}}, \{pk_i\}_{i\in[n]}, \{\widehat{\mathsf{CT}_i}\}_{i\in[n]}\big]$ is described in Figure 3. $\overline{G_{\mathsf{ind}}}$ is set to be equal to $\overline{H_2}$.
In $\mathsf{Hyb}_{2.4.z}$ it is generated as follows: Generate the obfuscation, $\overline{\mathsf{H}_1} \leftarrow \Pi_{\mathsf{ind}}.\mathsf{Obf}(1^\lambda, \mathsf{H}_1)$, where $\mathsf{H}_1 = \mathsf{H}_1\big[z+1, C_0, K^{\mathsf{ind}}_z, sk_{\mathsf{ind}}, \{sk_i\}_{i\in[n]\setminus\{\mathsf{ind}\}}, \{pk_i\}_{i\in[n]}, \{\widehat{\mathsf{CT}_i}\}_{i\in[n]}, v\big]$ is described in Figure 2. $\overline{G_{\mathsf{ind}}}$ is set as $= \overline{H_1}$. Here v is instantiated as in $\mathsf{Hyb}_{2.4.z}$.
Note that circuits H_2 when instantiated with z and H_1 instantiated with z and v are functionally equivalent. This is because the only point at which these two circuits may differ is the input z. Because at all points $x \neq z$, both circuits evaluate the same code except that H_2 uses a PRF key K^{ind} while H_1 uses a punctured PRF key (punctured at z). Since the puncturable PRF is correct, the evaluation at all points $x \neq z$ guarantees same output.
At this input z, H_1 outputs the hard coded value v. By the definition of hybrid $\mathsf{Hyb}_{2.4.z}$, v is set to be equal to the output of H_2 at z which causes two circuits to be functionally equivalent. The security then follows from the security of indistinguishibility obfuscator candidate Π_{ind}. □

Lemma 7. *If Π_{ind} is a δ_{io}-secure IO candidate then for any PPT adversary $\mathcal{A}$ and $z \in \{0,1\}^\lambda \setminus 2^\lambda - 1$, it holds that* $\Big|Pr\big[\mathcal{A}(\mathsf{Hyb}_{2.5.z}) = 1\big] - Pr\big[\mathcal{A}(\mathsf{Hyb}_{2.1.z+1}) = 1\big]\Big| < \delta_{io}$.

Proof. The only difference between $\mathsf{Hyb}_{2.5.z}$ and $\mathsf{Hyb}_{2.1.z+1}$ is the manner in which $\overline{G_{\mathsf{ind}}}$ is generated. In $\mathsf{Hyb}_{2.5.z}$ it is generated as follows: $\overline{\mathsf{H}_2} \leftarrow \Pi_{\mathsf{ind}}.\mathsf{Obf}(1^\lambda, \mathsf{H}_2)$, where $\mathsf{H}_2 = \mathsf{H}_2\big[z, C_0, K^{\mathsf{ind}}, sk_{\mathsf{ind}}, \{sk_i\}_{i\in[n]\setminus\{\mathsf{ind}\}}, \{pk_i\}_{i\in[n]}, \{\widehat{\mathsf{CT}_i}\}_{i\in[n]}\big]$ is described in Figure 3. $\overline{G_{\mathsf{ind}}}$ is set to be equal to $\overline{H_2}$.
In $\mathsf{Hyb}_{2.1.z+1}$, it is generated as follows: Generate the obfuscation, $\overline{\mathsf{H}_1} \leftarrow \Pi_{\mathsf{ind}}.\mathsf{Obf}(1^\lambda, \mathsf{H}_1)$, where $\mathsf{H}_1 = \mathsf{H}_1\big[z+1, C_0, K^{\mathsf{ind}}_{z+1}, sk_{\mathsf{ind}}, \{sk_i\}_{i\in[n]\setminus\{\mathsf{ind}\}}, \{pk_i\}_{i\in[n]}, \{\widehat{\mathsf{CT}_i}\}_{i\in[n]}, v\big]$ is described in Figure 2. $\overline{G_{\mathsf{ind}}}$ is set as $= \overline{H_1}$. Here v is instantiated as in $\mathsf{Hyb}_{2.1.z+1}$.
Note that circuits H_2 when instantiated with z and H_1 instantiated with $z+1$ and v are functionally

equivalent. This is because the only point at which these two circuits may differ is the input $z+1$. Because at all inputs $x \neq z+1$, both circuits evaluate the same code except that H_2 uses a PRF key K^{ind} while H_1 uses a punctured PRF key (punctured at $z+1$). Since the puncturable PRF is correct, the evaluation at all points $x \neq z+1$ guarantees same output.
At this input $z+1$, H_1 outputs the hard coded value v. By the definition of hybrid $\mathsf{Hyb}_{2.1.z+1}$, v is set to be equal to the output of H_2 at $z+1$ which causes two circuits to be functionally equivalent. The security then follows from the security of indistinguishibility obfuscator candidate Π_{ind}. □

Lemma 8. *If Π_{ind} is δ_{io}-secure then for any PPT adversary $\mathcal{A}$, it holds that $\left|Pr\left[\mathcal{A}(\mathsf{Hyb}_{2.5.2^\lambda-1}) = 1\right] - Pr\left[\mathcal{A}(\mathsf{Hyb}_3) = 1\right]\right| < \delta_{io}$.*

Proof. The only difference between $\mathsf{Hyb}_{2.5.2^\lambda-1}$ and Hyb_3 is the manner in which $\overline{G_{\mathsf{ind}}}$ is generated. In $\mathsf{Hyb}_{2.5.2^\lambda-1}$ it is generated as follows: $\overline{\mathsf{H}_2} \leftarrow \Pi_{\mathsf{ind}}.\mathsf{Obf}(1^\lambda, \mathsf{H}_2)$, where $\mathsf{H}_2 = \mathsf{H}_2\left[z = 2^\lambda - 1, C_0, K_z^{\mathsf{ind}}, sk_{\mathsf{ind}}, \{sk_i\}_{i\in[n]\setminus\{\mathsf{ind}\}}, \{pk_i\}_{i\in[n]}, \{\widehat{\mathsf{CT}_i}\}_{i\in[n]}\right]$ is described in Figure 3. $\overline{G_{\mathsf{ind}}}$ is set to be equal to $\overline{H_2}$.
In Hyb_3, it is generated as follows: $\overline{\mathsf{H}_3} \leftarrow \Pi_{\mathsf{ind}}.\mathsf{Obf}(1^\lambda, \mathsf{H}_3)$, where $\mathsf{H}_3 = \mathsf{H}_3\left[C_0, K^{\mathsf{ind}}, \{sk_i\}_{i\in[n]\setminus\{\mathsf{ind}\}}, \{pk_i\}_{i\in[n]}, \{\widehat{\mathsf{CT}_i}\}_{i\in[n]}\right]$ is described in Figure 4. Set $\overline{G_{\mathsf{ind}}} = \overline{H_3}$.
Note that circuits H_2 when instantiated with $z = 2^\lambda - 1$ and H_3 are functionally equivalent. This is because at all inputs x, both circuits evaluate the same code except that H_2 has sk_{ind} hardcoded while H_1 does not.
The security then follows from the security of indistinguishibility obfuscator candidate Π_{ind}.
□

Lemma 9. *If TMFHE is secure if for any PPT adversary $\mathcal{A}$, it holds that $\left|Pr\left[\mathcal{A}(\mathsf{Hyb}_3) = 1\right] - Pr\left[\mathcal{A}(\mathsf{Hyb}_4) = 1\right]\right| < \mathsf{negl}(\lambda)$.*

Proof. The only difference between Hyb_3 and Hyb_4 is the manner how $\mathsf{CT}_{\mathsf{ind}}$ is generated. In Hyb_3 it is generated as an TMFHE encryption of S_{ind} under pk_{ind} while in Hyb_4 it is generated as an encryption of $S_{\mathsf{ind}} \oplus C_b \oplus C_0$ under pk_{ind}. Note that sk_{ind} is not involved in both these hybrids. The indistinguishability then follows from the semantic security of TMFHE.
□

Lemma 10. *For any PPT adversary $\mathcal{A}$, the advantage of guessing b' correctly in Hyb_4 is exactly 0.*

Proof. In Hyb_4, the strings $\{S_i\}_{i\in[n]}$ used while computing $\{\widehat{\mathsf{CT}_i}\}_{i\in[n]}$ is distributed identically as shares of C_0. Hence, Hyb_4 carries no information about but b and the claim follows. □

Summing up advantages, we get that the advantage of adversary in the obfuscation security game is bounded by $2^{\lambda+1}\mu + 2^{\lambda+1}\delta_{io} + 2^{\lambda+1}\delta_{prf} + \mathsf{negl}$. Upon setting μ, δ_{io} and δ_{prf} to be $O(2^{-2\lambda})$ we get our claim.
□

5.2 DDH-Based Construction

In this section we present our construction from DDH. Before doing so, we look at some of the primitives that are used for our construction of IO combiner based on DDH.

5.2.1 Main Ingredients

SSS-NIZK. 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 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 (Σ, 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

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

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

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

Computational zero-knowledge. 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 (Σ, 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 $x \in L$

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

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

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

Such statistical sound NIZK's can be constructed as in [GGH+13b].

Re-randomizable Garbled Circuits. We import the definitions of a variant of re-randomizable garbled circuits [GHV10] from [ACG+14]. This construction can be based on DDH and suffices for our construction. The garbling scheme consists of four algorithms $\mathsf{Garble} = (\mathsf{YaoGarbledCkt}, \mathsf{FHEEval}, \mathsf{Sim}, \mathsf{reRand})$ described below.

- $\mathsf{YaoGarbledCkt}(1^\lambda, C) \rightarrow (\mathsf{GC}, \mathbf{w})$. Takes as input a circuit C with n inputs and outputs a garbled circuit GC and set of wire labels $\mathbf{w} = \{w_{i,0}, w_{i,1}\}_{i\in[n]}$.

- $\mathsf{Eval}(\mathsf{GC}, \mathbf{w}_x) \rightarrow y$. The evaluate algorithm takes as input the garbled circuit and set of labels $\mathbf{w}_x = \{w_{i,x_i}\}_{i\in[n]}$ corresponding to some input x and outputs y.

- $\mathsf{Sim}(1^\lambda, |C|, C(x)) \rightarrow (\tilde{\mathsf{GC}}, \tilde{\mathbf{w}}_x)$. takes as input a value $C(x)$ and a size $|C|$ and produces a simulated garbled circuit $\tilde{\mathsf{GC}}$ and labels $\tilde{\mathbf{w}}_x$.

- $\mathsf{reRand}(\mathsf{GC}, \mathbf{w}_x; R) \rightarrow (\mathsf{GC}', \mathbf{w}'_x)$. The re-randomization algorithm takes as input a randomness value R a garbled circuit GC and input labels $\mathbf{w}_x$ for some input x and computes a new garbled circuit GC' and labels $\mathbf{w}'_x$.

We require following properties from the scheme:
Correctness: The correctness is two fold. First, the $\mathsf{FHEEval}$ algorithm on input a garbled circuit GC and labels $\mathbf{w}_x$ generated for some circuit C an input x by $\mathsf{YaoGarbledCkt}$ algorithm must output $C(x)$. Second, re-randomized garbled circuit and labels should also give correct outputs.

Simulation security: We require the following distributions to be computationally close:

$$\left\{\mathsf{GC}, \mathbf{w}_x\right\} \approx_c \left\{\tilde{\mathsf{GC}}, \tilde{\mathbf{w}}_x\right\}$$

Here GC and $\mathbf{w}_x$ are the garbled circuit and labels corresponding to some circuit C and input x respectively which are generated by $\mathsf{YaoGarbledCkt}$ algorithm. $\tilde{\mathsf{GC}}, \tilde{\mathbf{w}}_x$ are generated using the simulated algorithm Sim.

Re-randomization security: Let GC $\mathbf{w}$ be a garbled circuit and set of labels for C generated by the $\mathsf{YaoGarbledCkt}$ algorithm. Then it must hold that.

$$\left\{\mathsf{GC}', \mathbf{w}'_x, \mathsf{GC}, \mathbf{w}\right\} \approx_c \left\{\mathsf{GC}'', \mathbf{w}''_x, \mathsf{GC}, \mathbf{w}\right\}$$

Here the garbled circuit GC' and labels w'_x (corresponding to input x) is obtained by re-randomizing the garbled circuit GC and the labels $\mathbf{w}_x$ using reRand algorithm. Garbled circuit GC'' and labels $\mathbf{w}''_x$ (corresponding to input x), is generated afresh by garbling C using the $\mathsf{YaoGarbledCkt}$ algorithm.

A re-randomizable garbled circuit is $\mu-$secure if the advantage of any PPT adversary in breaking the simulation and re-randomization security is less than μ.

Homomorphic Re-encryption Scheme. We also require a weakly homomorphic encryption scheme $\mathsf{HE} = (\mathsf{KeyGen}, \mathsf{ReKey}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{HEEval}, \mathsf{ReEncrypt})$. All the algorithms are as in a standard public key re-encryption scheme except that additionally we have an algorithm HEEval that takes in a ciphertext corresponding to set of ciphertext corresponding to a re-randomizable garbled circuit and labels for any input x and a randomness value and outputs a ciphertext encrypting re-randomized garbled circuit and labels for the same input according to the algorithm reRand.

The algorithm ReKey takes as input two secret keys sk_1 and sk_2 and outputs a re-encryption key $\tau_{1,2}$. The $\mathsf{ReEncrypt}$ algorithm is a randomized algorithm that takes as input a ciphertext CT corresponding to message m encrypted under pk_1 and a re-encryption key $\tau_{1,2}$ and outputs a ciphertext encrypting the same message under pk_2.

From security point of view we require that the distribution of a re-encrypted cipher-text (re-encrypted from under public-key pk_1 to pk_2) for any message m should be statistically close (with distance ϵ) to the distribution of a fresh ciphertext encrypting m encrypted under pk_2. We call such a scheme ϵ-secure weakly homomorphic re-encryption scheme. A description of re-randomizable garbled circuit and re-encryption scheme satisfying these properties constructed assuming DDH can be found in [BBS98].

5.2.2 Construction

We use a puncturable PRF and a non-interactive statistically binding commitment scheme Com. Let HE be a $2^{-2\lambda}$-secure weakly homomorphic encryption scheme and Garble be a re-randomizable garbled circuit scheme as described above $\mathsf{SSSNIZK} = (\Sigma, P, V, S)$ be a statistically simulation sound NIZK proof system. We are now ready to present our construction. The construction is described at a high level as follows. The obfuscator computes a re-randomizable garbled circuit and labels for the circuit C as GC and $\mathbf{w}$. It samples $n+1$ public key-secret key pair for the re-encryption scheme HE and also computes n re-encryption keys τ_i for all $i \in [n]$ (τ_i converts a ciphertext under pk_i to one under pk_{i+1}). Then $\mathsf{GC}, \mathbf{w}$ are encrypted under pk_1. Finally for all $i \in [n]$, we obfuscate a special circuits G_i (using Π_i) which takes input as a ciphertext encrypting garbled circuit and labels (under pk_i) corresponding to an input x along with some proofs (with respect to some commitments, that the computations are done correctly). G_i verifies these proofs and then re-encrypts/re-randomizes these cipher-text under the public key pk_{i+1}. The obfuscation consists of the encryptions of the garbled circuits and the labels under pk_1, n obfuscated programs, the secret key sk_{n+1}.

We construct an IO combiner $\Pi_{\mathsf{comb}} = \Pi_{\mathsf{comb}}[\Pi_1, \ldots, \Pi_n]$ for $\mathcal{C}$ below.

$\underline{\mathsf{CombObf}}(1^\lambda, C, \Pi_1, \ldots, \Pi_n)$: It takes as input security parameter λ, circuit $C \in \mathcal{C}_\lambda$ with λ inputs, description of candidates $\{\Pi_i = (\Pi_i.\mathsf{Obf}, \Pi_i, \mathsf{Eval})\}_{i\in[n]}$ and does the following.

1. **Setup phase:**
   - Run the setup of encryption scheme $\{\mathsf{HE.KeyGen}(1^\lambda) \to (pk_i, sk_i)\}_{i\in[n+1]}$.
   - Compute the re-encryption keys $\mathsf{HE.ReKey}(sk_i, sk_{i+1}) \to \tau_i$.
   - Sample puncturable PRF keys $\{\mathsf{PRF.Setup}(1^\lambda) \to K_{i,j}\}_{i\in[n],j\in[3]}$
   - Compute commitments $Z = \mathsf{Com}(0)$, $Z_i = \mathsf{Com}(\tau_i)$ and $\{Z_{i,j} = \mathsf{Com}(K_{i,j}) \forall i \in [n], j \in [2]\}$. Let $\left\{a, \{a_i, a_{i,j}\}_{i\in[n],j\in[2]}\right\}$ be the openings to the commitments.
2. **Setting up Circuits:**
   - Garble the circuit C, i.e. $(\mathsf{GC}, \mathbf{w}) \leftarrow \mathsf{YaoGarbledCkt}(1^\lambda, C)$.
   - Compute encryptions of the garbled circuit and the labels under pk_1, $\mathsf{CT} = \mathsf{HE.Enc}(pk_1, \mathsf{GC})$ and $\mathsf{CT}_{i,j} = \mathsf{HE.Enc}(pk_1, w_{i,j})$ for $i \in [\lambda]$, $j \in \{0, 1\}$.

- Run the setup of $\mathsf{SSSNIZK}$ proof, $\Sigma(1^\lambda) \to \sigma$.
- Now we define relation R_0. X satisfies the relation R_0 if $X = (x, y)$ iff:
  (a) $x \in \{0,1\}^\lambda$
  (b) $y = \{\mathsf{CT}, \mathsf{CT}_{i,x_i}\}_{i\in\lambda}$
- For $i \in [n]$, we now define a language L_i for a $\mathsf{SSSNIZK}$ proof (let V_i be the $\mathsf{SSSNIZK}$ verifier):

  For instance $X = (x, \mathsf{CT}^{i-1}, \{\mathsf{CT}^{i-1}_{j,x_j}\}_{j\in[\lambda]}, \mathsf{CT}^i, \{\mathsf{CT}^i_{j,x_j}\}_{j\in[\lambda]})$, $(X, v) \in L_i$ if the following happens.
  * *Either* all of the conditions described below occur
    1 $Z_i = \mathsf{Com}(\tau_i; \mathsf{a_i})$
    2 $Z_{i,j} = \mathsf{Com}(K_{i,j}; a_{i,j}) \forall j \in [2]$
    3 Let $Q = \mathsf{HEEval}\Big(pk_i, \mathsf{CT}^{i-1}, \{\mathsf{CT}^{i-1}_{k,x_k}\}_{k\in[\lambda]}, \mathsf{reRand}(; \mathsf{F}(K_{i,1}, x))\Big)$. Then $\{\mathsf{CT}^i, \{\mathsf{CT}^i_{k,x_k}\}_{k\in[\lambda]}\} =$
    $$\mathsf{ReEncrypt}\Big(\tau_i, Q; \mathsf{F}(K_{i,2}, x)\Big)$$
  * *or,* $Z = \mathsf{Com}(i; a)$
- For every $i \in [n]$, construct circuit $G_i = G_i[Z, \{Z_k, Z_{k,j}, K_{i,j}\}_{k\in[n],j\in[3]}, pk_{j\in[n+1]}, \tau_i, \mathsf{CT}, \{\mathsf{CT}_{l,j}\}_{l\in[\lambda],j\in\{0,1\}}] \in \mathcal{C}^i$ as described in Figure 5.

3. **Obfuscating Circuits with Candidates**

- Generate $\overline{G_i} \leftarrow \Pi_i.\mathsf{Obf}(1^\lambda, G_i)$.

Output the obfuscation $\overline{C} = \big(\ \mathsf{CT}, \{\mathsf{CT}_{i,j}\}_{i\in[\lambda],j\in\{0,1\}}, sk_{n+1}, \overline{G_1}, \ldots, \overline{G_n}\ \big)$.

$\underline{\mathsf{CombEval}(\overline{C}, x, \Pi_1, \ldots, \Pi_n)}$: Parse $\overline{C} = (\mathsf{CT}, \{\mathsf{CT}_{i,j}\}_{i\in[\lambda],j\in\{0,1\}}, sk_{n+1}, \overline{G_1}, .., \overline{G_n})$. On input an obfuscation $\overline{C}$, to evaluate on x, compute y_0 by selecting encryptions $\mathsf{CT}, \{\mathsf{CT}_{i,x_i}\}_{i\in[\lambda]}$. Then iteratively compute $\Pi_i.\mathsf{Eval}\big(\overline{G_i}, (x, y_0, ..., y_{i-1}, \pi_1, .., \pi_i)\big) = (y_i, \pi_i)$ for $i \in [n]$. Recover a garbled circuit and set of labels for x (i.e. $\mathsf{GC}_n, \mathbf{w}_{n,x}$) by decrypting y_n using sk_{n+1}. Evaluate and output $\mathsf{Garble.FHEEval}(\mathsf{GC}_n, \mathbf{w}_{n,x})$.

Instantiations: We note that Com can be instantiated using ElGamal commitments. $\mathsf{SSSNIZK}$ are known from IO and one way functions [BP15]. We remark that for the security proof, it is enough to consider a construction in which each circuit G_i uses a $\mathsf{SSSNIZK}$ constructed from Π_i and DDH to prove statements.

5.2.3 Correctness and Security of Π_{comb}

Theorem 9. *If for all $i \in [n]$, Π_i is (1)-correct, then $\Pi_{\mathsf{comb}} = \Pi_{\mathsf{comb}}[\Pi_1, \ldots, \Pi_n]$ is (1)-correct.*

Proof Sketch. We prove the correctness of the above scheme below. Consider a circuit $C \in \mathcal{C}_\lambda$ and let $x \in \{0,1\}^\lambda$. Let G_i, for every $i \in [n]$, be the circuit generated during the execution of $\mathsf{Obf}(1^\lambda, C, \Pi_1, \ldots, \Pi_n)$. Let $\overline{G_i} \leftarrow \Pi_i.\mathsf{Obf}(1^\lambda, G_i)$. Finally let $\overline{C} = (\mathsf{CT}, \{ct_{i,j}\}_{i\in[\lambda],j\in\{0,1\}}, sk_{n+1},$

$$G_i\left[Z, \{Z_k, Z_{k,j}, K_{i,j}\}_{k\in[n],j\in[3]}, pk_{j\in[n+1]}, \tau_i, \mathsf{CT}, \{\mathsf{CT}_{l,j}\}_{l\in[\lambda],j\in\{0,1\}}\right]$$

Hardwired values: Commitments $Z, \{Z_k, Z_{k,j}\}_{k\in[n],j\in[2]}$, Openings $a_{i,j\in[2]}$, PRF keys $\{K_{i,j}\}_{j\in[3]}$, Public Keys $pk_{j\in[n+1]}$, Re-encryption keys τ_i, Encrypted garbled circuit and labels $\mathsf{CT}, \{\mathsf{CT}_{l,j}\}_{l\in[\lambda],j\in\{0,1\}}$
Input: $X = (x, y_0, y_1, .., y_{i-1}, \pi_1, ..., \pi_{i-1})$.

- Check that y_0 is a garbled circuit and set of labels (corresponding to input x), encrypted under pk_1. That is, $R(x, y_0) = 1$.
- Using the $\mathsf{SSSNIZK}$ verifier, check that $V_l(x, y_{l-1}, y_l, \pi_l) = 1$ for every $l \in [i-1]$. If any check fails then abort.
- Compute randomness $\alpha_l \leftarrow \mathsf{F}(K_{i,l}, x)$ for $l \in [3]$.
- Let $Q = \mathsf{HE.HEEval}\left(pk_i, \mathsf{CT}^{i-1}, \{\mathsf{CT}^{i-1}_{k,x_k}\}_{k\in[\lambda]}, \mathsf{reRand}\left(;\alpha_1\right)\right)$. Compute $y_i \leftarrow \mathsf{HE.ReEncrypt}\left(\tau_i, Q; \alpha_2\right)$.
- Compute a $\mathsf{SSSNIZK}$ proof π_i for the statement that there exists a witness v such that $(x, y_{i-1}, y_i, v) \in L_i$ by proving the first statement for the language L_i using the hard-wired witness.
- Output (y_i, π_i)

Figure 5: Circuit G_i

$\overline{G_1} \ldots \overline{G_n}$) be the output of $\mathsf{CombObf}$. On input an obfuscation $\overline{C}$, to evaluate on x, compute y_0 by selecting encryptions $\mathsf{CT}, \{\mathsf{CT}_{i,x_i}\}_{i\in[\lambda]}$. Then iteratively compute $\Pi_i.\mathsf{Eval}\big(\overline{G_i}, (x, y_0, ..., y_{i-1}, \pi_1, .., \pi_i)\big) = (y_i, \pi_i)$ for $i \in [n]$. Recover a garbled circuit and set of labels for x (i.e. $\mathsf{GC}_n, \mathbf{w}_{n,x}$) by decrypting y_n using sk_{n+1}. Evaluate and output $\mathsf{Garble.FHEEval}(\mathsf{GC}_n, \mathbf{w}_{n,x})$.
Note that when each Π_i is correct, evaluation of each $\overline{G_i}$ on input x is the same as the evaluation of G_i on x. The circuits G_1 takes in an encrypted garbled circuit and labels for an input x under pk_1 and computes a re-encryption/re-randomization of the garbled circuit and labels under pk_2 along with a statistically sound proof that this computation is correct. The circuit G_2 takes in this proof along with set of encrypted labels and circuits along with the input x. It verifies these proofs and computes a re-encryption/re-randomization of garbled circuit and labels for x in pk_3 along with a proof. Iteratively, G_n computes computes a re-encryption/re-randomization of garbled circuit and labels for x in pk_{n+1}. If the re-encryption algorithm and garbled circuits are correct this evaluation gives us $C(x)$ as output. □

Theorem 10. *Consider a circuit class $\mathcal{C} \in P/poly$. Assuming subexponentially secure DDH the scheme Π_{comb} is an $(2^{-2\lambda}, \mathsf{negl}(\lambda))$ IO combiner for $\mathcal{C}$.*

Proof. We first sketch the proof and give the formal details later. We now sketch a sequence of 2^λ hybrids and argue indistinguishability. The first hybrid corresponds to an obfuscation of C_b, while the last hybrid is independent of b (hence, the advantage of adversary in this hybrid is 0). Each hybrid is indistinguishable from each other implying security.
From the first hybrid, we go to a hybrid where $Z = \mathsf{Com}(\mathsf{ind})$. Since, the opening of Z was not used in any of the proofs, these two hybrids are indistinguishable. Changing Z allows the obfuscation

$\overline{G_i}$ to cheat in the proofs. Now define a sequence of exponential number of hybrids. We go input by input where inputs to G_i are of the form $(x, y_0, .., y_{\mathsf{ind}-1}, \pi_1, ..., \pi_{\mathsf{ind}-1})$. We first puncture $K_{i,3}$ (used for generating $\mathsf{SSSNIZK}$) at point $x = z$ ($z = 0$ initially) and use the actual evaluation at z for generating proofs (if the input proofs verify). These two hybrids are close because of correctness of the PRF and security of Π_{ind}. In the next hybrid, we use a random value instead of the actual PRF and hard-wire the proof. These two hybrids are close because of the security of the puncturable PRF. Now we simulate the $\mathsf{SSSNIZK}$ at input such that $x = z$. Security holds because of the zero knowledge property of $\mathsf{SSSNIZK}$. In the next hybrid we puncture keys $K_{i,1}$ and $K_{i,2}$ at input x. These two hybrids are close because of the correctness of PRF and security of Π_{ind}. We then hard-wire the re-encryption/re-randomization with a fresh encryption of a new garbled circuit for C_0 and input x. This hybrid is indistinguishable from the previous hybrid due to $\mu-$security of the re-encryption scheme and re-randomization security of the garbled circuit. In the next hybrid ,we output an encryption of simulated garbled circuit at inputs such that $x = z$. These two hybrids are indistinguishable due to the security of the simulation security of the garbled circuit. Now we give a genuine $\mathsf{SSSNIZK}$ using the trapdoor for Z. In the next hybrid, we unpuncture the PRF keys $K_{i,1}, K_{i,2}$ and $K_{i,3}$ (by first going from random to actual evaluations at z). These hybrids are indistinguishable from security of Π_{ind} and security/correctness of the PRF. This way we go input by input from $z = 0$ to $z = 2^\lambda$ to a circuit that outputs a fresh encryptions of simulated garbled circuit (simulated using C_0). At this point we remove the re-encryption key τ_{ind} from the obfuscation $\overline{G}_{\mathsf{ind}}$. These hybrids are indistinguishable due the security of Π_{ind}. Finally, we can can change the encryptions $\mathsf{CT}, \{\mathsf{CT}_{i,j}\}_{i\in[\lambda],j\in\{0,1\}}$ to an encryption of garbled circuit for C_0. These two hybrids are close due to polynomial security of the re-encryption scheme. Note that hybrid carry no information about b.

We give the formal description of the hybrids next.

Hybrids: Consider a set of IO candidates $\Pi_1, \ldots, \Pi_n$. Let Π_{ind} be a secure IO candidate for some $\mathsf{ind} \in [n]$. We employ the standard hybrid argument to prove the theorem. Consider two circuits $C_0, C_1 \in \mathcal{C}_\lambda$ such that $C_0(x) = C_1(x)$ for all $x \in \{0,1\}^\lambda$. In the first hybrid (Hyb_1), the circuit C_b is obfuscated honestly with $b \xleftarrow{\$} \{0,1\}$. In the final hybrid ($\mathsf{Hyb}_7$), the obfuscated circuit the adversary has no information about b. At this point, the probability of guessing the bit b is exactly $1/2$. By arguing indistinguishability of every consecutive intermediate hybrids, we show that the probability of guessing b in the first hybrid is negligibly close to $1/2$ (or the advantage is 0), which proves the theorem.

In the following set of hybrids, denote $\mathcal{A}$ to be a PPT adversary.

Hyb_1: Pick a bit b at random. The adversary receives the obfuscation $\overline{C_b}$, where $\overline{C_b} \leftarrow \Pi_{\mathsf{comb}}.\mathsf{CombObf}(1^\lambda, C_b, \Pi_1, \ldots, \Pi_n)$. Let $\mathcal{A}$ output b'. Output of Hyb_1 is b'.

Hyb_2: This hybrid is the same as the previous hybrid except that Z is a commitment of ind.

$\mathsf{Hyb}_{3.1.z}$ for $z \in [0, 2^\lambda]$: This hybrid is the same as the previous hybrid except that $\overline{G_{\mathsf{ind}}}$ is generated as an obfuscation of the circuit (denoted by H_1) described in 6.

$\mathsf{Hyb}_{3.2.z}$ for $z \in \{0,1\}^\lambda$: This hybrid is the same as the previous hybrid except the following.

- Puncture $K_{\mathsf{ind},3}$ at z to compute, K^z_{ind} and $r^z_{\mathsf{ind}} \leftarrow PRF_{K_{\mathsf{ind},3}}(z)$.

$\mathsf{H}_1\left[z, Z, \{Z_k, Z_{k,j}, K_{\mathsf{ind},j}\}_{k\in[n],j\in[3]}, pk_{j\in[n+1]}, \tau_{\mathsf{ind}}, \mathsf{CT}, \{\mathsf{CT}_{l,j}\}_{l\in[\lambda],j\in\{0,1\}}, a, a_{\mathsf{ind}}, a_{\mathsf{ind},j\in[2]}\right]$

Hardwired values: Input index z, Commitments $Z, \{Z_k, Z_{k,j}\}_{k\in[n],j\in[2]}$, Openings $a, a_{\mathsf{ind}}, a_{\mathsf{ind},j\in[2]}$, PRF keys $\{K_{\mathsf{ind},j}\}_{j\in[3]}$, Public Keys $pk_{j\in[n+1]}$, Re-encryption key τ_{ind}, Encrypted garbled circuit and labels $\mathsf{CT}, \{\mathsf{CT}_{l,j}\}_{l\in[\lambda],j\in\{0,1\}}$.
Input: $X = (x, y_0, y_1, .., y_{\mathsf{ind}-1}, \pi_1, ..., \pi_{\mathsf{ind}-1})$.

- Check that y_0 is a garbled circuit and set of labels (corresponding to input x), encrypted under pk_1. That is, $R(x, y_0) = 1$.
- Parse $y_i = (\mathsf{CT}^{i-1}, \{\mathsf{CT}^{i-1}_{k,x_k}\}_{k\in[\lambda]})$ for $i \geq 1$.
- Using the $\mathsf{SSSNIZK}$ verifier, check that $V_l(x, y_{l-1}, y_l, \pi_l) = 1$ for every $l \in [\mathsf{ind} - 1]$. If any check fails then abort.
- Compute randomness $\alpha_l \leftarrow \mathsf{F}(K_{i,l}, x)$ for $l \in [3]$.
- Let $Q = \mathsf{HE.HEEval}\left(pk_{\mathsf{ind}-1}, \mathsf{CT}^{\mathsf{ind}-1}, \{\mathsf{CT}^{\mathsf{ind}-1}_{k,x_k}\}_{k\in[\lambda]}, \mathsf{reRand}\,(;\alpha_1)\right)$. Compute $y_{\mathsf{ind}} \leftarrow \mathsf{HE.ReEncrypt}\,(\tau_{\mathsf{ind}}, Q; \alpha_2)$.
- Compute a $\mathsf{SSSNIZK}$ proof π_{ind} for the statement that there exists a witness v such that $(x, y_{i-1}, y_i, v) \in L_i$. If $x \geq z$ prove the first statement for the language L_i using the hard-wired witness. If $x < z$ prove the second statement by using the trapdoor witness (opening of Z). This proof is computed using the randomness $PRF_{K_{\mathsf{ind},3}}(x)$.
- Output $(y_{\mathsf{ind}}, \pi_{\mathsf{ind}})$

Figure 6: Circuit H_1

- Construct circuits G_i for $i \leq \mathsf{ind}$. Then iteratively compute $G_i(x, y_0, ..., y_{i-1}, \pi_1, .., \pi_i) = (y_i, \pi_i)$ for $i \in [\mathsf{ind}]$
- Compute a $\mathsf{SSSNIZK}$ proof π^z_{ind} that $(z, y_{\mathsf{ind}-1}, y_{\mathsf{ind}}) \in L_{\mathsf{ind}}$ by proving the first statement and using the randomness r^z_{ind}.
- $\overline{G_{\mathsf{ind}}}$ is now generated as an obfuscation of the circuit (denoted by H_2) described in 7.

$\underline{\mathsf{Hyb}_{3.3.z} \text{ for } z \in \{0,1\}^\lambda}$: This hybrid is the same as the previous hybrid except that r^z_{ind} is sampled randomly.

$\underline{\mathsf{Hyb}_{3.4.z} \text{ for } z \in \{0,1\}^\lambda}$: This hybrid is the same as the previous hybrid except that π^z_{ind} is computed by proving the second statement of the proof (Z is a commitment of ind).

$\underline{\mathsf{Hyb}_{3.5.z} \text{ for } z \in \{0,1\}^\lambda}$: This hybrid is the same as the previous hybrid except that r^z_{ind} is computed as $PRF_{K_{\mathsf{ind},3}}(z)$.

$\underline{\mathsf{Hyb}_4}$: This hybrid is the same as the previous hybrid except that $\overline{G_{\mathsf{ind}}}$ is now generated as an obfuscation of the circuit (denoted by H_3) described in Figure 8.

$\underline{\mathsf{Hyb}_{5.0} \text{ for } z \in \{0,1\}^\lambda}$: This hybrid is the same as the previous hybrid except that $Z_{\mathsf{ind}}, Z_{\mathsf{ind},1}, Z_{\mathsf{ind},2}$ is now a commitment of 0.

$\mathsf{H}_2\big[z, Z, \{Z_k, Z_{k,j}, K_{\mathsf{ind},j}\}_{k\in[n],j\in[2]}, K^z_{\mathsf{ind}}, pk_{j\in[n+1]}, \tau_{\mathsf{ind}}, \mathsf{CT}, \{\mathsf{CT}_{l,j}\}_{l\in[\lambda],j\in\{0,1\}}, a, a_{\mathsf{ind}}, a_{\mathsf{ind},j\in[2]}, \pi^z_{\mathsf{ind}}\big]$

Hardwired values: Commitments $Z, \{Z_k, Z_{k,j}\}_{k\in[n],j\in[2]}$, Openings $a, a_{\mathsf{ind}}, a_{\mathsf{ind},j\in[2]}$, PRF keys $\{K_{\mathsf{ind},j}\}_{j\in[2]}$ and a punctured key K^z_{ind}, Public Keys $pk_{j\in[n+1]}$, Re-encryption key τ_{ind}, Encrypted garbled circuit and labels $\mathsf{CT}, \{\mathsf{CT}_{l,j}\}_{l\in[\lambda],j\in\{0,1\}}$, Proof π^z_{ind}
Input: $X = (x, y_0, y_1, .., y_{\mathsf{ind}-1}, \pi_1, ..., \pi_{\mathsf{ind}-1})$.

- Check that y_0 is a garbled circuit and set of labels (corresponding to input x), encrypted under pk_1. That is, $R(x, y_0) = 1$.
- Using the $\mathsf{SSSNIZK}$ verifier, check that $V_l(x, y_{l-1}, y_l, \pi_l) = 1$ for every $l \in [\mathsf{ind} - 1]$. If any check fails then abort.
- Compute randomness $\alpha_l \leftarrow \mathsf{F}(K_{\mathsf{ind},l}, x)$ for $l \in [3]$.
- Parse $y_i = (\mathsf{CT}^{i-1}, \{\mathsf{CT}^{i-1}_{k,x_k}\}_{k\in[\lambda]})$ for $i \geq 1$.
- Let $Q = \mathsf{HE.HEEval}\left(pk_i, \mathsf{CT}^{\mathsf{ind}-1}, \{\mathsf{CT}^{\mathsf{ind}-1}_{k,x_k}\}_{k\in[\lambda]}, \mathsf{reRand}\,(;\alpha_1)\right)$. Compute $y_{\mathsf{ind}} \leftarrow \mathsf{HE.ReEncrypt}\,(\tau_{\mathsf{ind}}, Q; \alpha_2)$.
- Compute a $\mathsf{SSSNIZK}$ proof π_{ind} for the statement that there exists a witness v such that $(x, y_{i-1}, y_i, v) \in L_i$. If $x > z$ prove the first statement for the language L_i using the hard-wired witness. If $x < z$ prove the second statement by using the trapdoor witness (opening of Z). The proof is generated using the randomness $PRF_{K^z_{\mathsf{ind}}}(x)$. At $x = z$ use the hard-wired proof $\pi_{\mathsf{ind}} = \pi^z_{\mathsf{ind}}$
- Output $(y_{\mathsf{ind}}, \pi_{\mathsf{ind}})$

Figure 7: Circuit H_2

$\underline{\mathsf{Hyb}_{5.1.z} \text{ for } z \in \{0,1\}^\lambda}$: This hybrid is the same as the previous hybrid except that $\overline{G_{\mathsf{ind}}}$ is an obfuscation of the circuit (denoted as H_4) described in Figure 9.

$\underline{\mathsf{Hyb}_{5.2.z} \text{ for } z \in \{0,1\}^\lambda}$: This hybrid is the same as the previous hybrid except that following is done.

- Puncture $K_{\mathsf{ind},2}$ at z to compute, $K^z_{\mathsf{ind},2}$ and $r^z_{\mathsf{ind},2} \leftarrow PRF_{K_{\mathsf{ind},2}}(z)$. Also puncture $K_{\mathsf{ind},3}$ at z to compute, $K^z_{\mathsf{ind},3}$ and $r^z_{\mathsf{ind},3} \leftarrow PRF_{K_{\mathsf{ind},3}}(z)$.
- Construct circuits G_i for $i \leq \mathsf{ind}$. Then iteratively compute $G_i(z, y_0, ..., y_{i-1}, \pi_1, .., \pi_i) = (y_i, \pi_i)$ for $i \in [\mathsf{ind} - 1]$. Parse $y_{\mathsf{ind}-1} = \mathsf{CT}^{\mathsf{ind}-1}, \{\mathsf{CT}^{\mathsf{ind}-1}_{k,z_k}\}_{k\in[\lambda]}$.
- Let $Q = \mathsf{HE.HEEval}\left(pk_{\mathsf{ind}}, \mathsf{CT}^{\mathsf{ind}-1}, \{\mathsf{CT}^{\mathsf{ind}-1}_{k,z_k}\}_{k\in[\lambda]}, \mathsf{reRand}\,(;\alpha_1)\right)$. Compute $y^z_{\mathsf{ind}} \leftarrow \mathsf{HE.ReEncrypt}\,(\tau_{\mathsf{ind}}, Q; \alpha_2)$.
- Compute a $\mathsf{SSSNIZK}$ proof π^z_{ind} that $(z, y_{\mathsf{ind}-1}, y_{\mathsf{ind}}) \in L_{\mathsf{ind}}$ by proving the first statement and using the randomness r^z_{ind}.
- $\overline{G_{\mathsf{ind}}}$ is now generated as an obfuscation of the circuit (denoted by H_5) described in 10.

$\mathsf{H}_3\left[Z, \{Z_k, Z_{k,j}, K_{\mathsf{ind},j}\}_{k\in[n],j\in[3]}, pk_{j\in[n+1]}, \tau_{\mathsf{ind}}, \mathsf{CT}, \{\mathsf{CT}_{l,j}\}_{l\in[\lambda],j\in\{0,1\}}, a\right]$

Hardwired values: Commitments $Z, \{Z_k, Z_{k,j}\}_{k\in[n],j\in[2]}$, Opening a, PRF keys $\{K_{\mathsf{ind},j}\}_{j\in[3]}$, Public Keys $pk_{j\in[n+1]}$, Re-encryption key τ_{ind}, Encrypted garbled circuit and labels $\mathsf{CT}, \{\mathsf{CT}_{l,j}\}_{l\in[\lambda],j\in\{0,1\}}$.
Input: $X = (x, y_0, y_1, .., y_{\mathsf{ind}-1}, \pi_1, ..., \pi_{\mathsf{ind}-1})$.

- Check that y_0 is a garbled circuit and set of labels (corresponding to input x), encrypted under pk_1. That is, $R(x, y_0) = 1$.
- Parse $y_i = (\mathsf{CT}^{i-1}, \{\mathsf{CT}^{i-1}_{k,x_k}\}_{k\in[\lambda]})$ for $i \geq 1$.
- Using the $\mathsf{SSSNIZK}$ verifier, check that $V_l(x, y_{l-1}, y_l, \pi_l) = 1$ for every $l \in [\mathsf{ind} - 1]$. If any check fails then abort.
- Compute randomness $\alpha_l \leftarrow \mathsf{F}(K_{\mathsf{ind},l}, x)$ for $l \in [3]$.
- Let $Q = \mathsf{HE.HEEval}\left(pk_{\mathsf{ind}}, \mathsf{CT}^{\mathsf{ind}-1}, \{\mathsf{CT}^{\mathsf{ind}-1}_{k,x_k}\}_{k\in[\lambda]}, \mathsf{reRand}\left(;\alpha_1\right)\right)$. Compute $y_{\mathsf{ind}} \leftarrow \mathsf{HE.ReEncrypt}\left(\tau_{\mathsf{ind}}, Q; \alpha_2\right)$.
- Compute a $\mathsf{SSSNIZK}$ proof π_{ind} for the statement that there exists a witness v such that $(x, y_{\mathsf{ind}-1}, y_{\mathsf{ind}}, v) \in L_{\mathsf{ind}}$. This is done by proving the second statement by using the trapdoor (opening of Z) as a witness. This proof is computed using the randomness $PRF_{K_{\mathsf{ind},3}}(x)$.
- Output $(y_{\mathsf{ind}}, \pi_{\mathsf{ind}})$

Figure 8: Circuit H_3

$\underline{\mathsf{Hyb}_{5.3.z} \text{ for } z \in \{0,1\}^\lambda}$: This hybrid is the same as the previous except that $r^z_{\mathsf{ind},1}$ and $r^z_{\mathsf{ind},2}$ are sampled randomly.

$\underline{\mathsf{Hyb}_{5.4.z} \text{ for } z \in \{0,1\}^\lambda}$: This hybrid is the same as the previous hybrid except that y^z_{ind} is computed differently.

- Construct circuits G_i for $i \leq \mathsf{ind}$. Then iteratively compute $G_i(z, y_0, ..., y_{i-1}, \pi_1, .., \pi_i) = (y_i, \pi_i)$ for $i \in [\mathsf{ind} - 1]$. Parse $y_{\mathsf{ind}-1} = \mathsf{CT}^{\mathsf{ind}-1}, \{\mathsf{CT}^{\mathsf{ind}-1}_{k,z_k}\}_{k\in[\lambda]}$. Decrypt $y_{\mathsf{ind}-1}$ with $sk_{\mathsf{ind}-1}$ to get $\mathsf{GC}^z_{\mathsf{ind}-1}, \mathbf{w}^z_{\mathsf{ind}-1}$.
- Let $Q = \mathsf{reRand}(\mathsf{GC}^z_{\mathsf{ind}-1}, \mathbf{w}^z_{\mathsf{ind}-1})$. Compute $y^z_{\mathsf{ind}} \leftarrow \mathsf{HE.Enc}(pk_{\mathsf{ind}}, Q)$.
  $\underline{\mathsf{Hyb}_{5.5.z} \text{ for } z \in \{0,1\}^\lambda}$: This hybrid is the same as the previous hybrid except that y^z_{ind} is computed differently.
- Garble the circuit C_b, i.e. $(\mathsf{GC}, \mathbf{w}) \leftarrow \mathsf{YaoGarbledCkt}(1^\lambda, C_b; r^z_{\mathsf{ind},1})$. Let $\mathbf{w}^z$ denote the labels for input z.
- Compute $y^z_{\mathsf{ind}} = \mathsf{HE.Enc}(pk_{\mathsf{ind}}, \mathsf{GC}, \mathbf{w}^z; r^z_{\mathsf{ind},2})$.

$\underline{\mathsf{Hyb}_{5.6.z} \text{ for } z \in \{0,1\}^\lambda}$: This hybrid is the same as the previous hybrid except that y^z_{ind} is computed differently.

$\mathsf{H}_4\left[z, Z, \{Z_k, Z_{k,j}, K_{\mathsf{ind},j}\}_{k\in[n],j\in[3]}, pk_{j\in[n+1]}, \tau_{\mathsf{ind}}, \mathsf{CT}, \{\mathsf{CT}_{l,j}\}_{l\in[\lambda],j\in\{0,1\}}, a, C_0\right]$

Hardwired values: Input index z, Commitments $Z, \{Z_k, Z_{k,j}\}_{k\in[n],j\in[2]}$, Openings a, PRF keys $\{K_{\mathsf{ind},j}\}_{j\in[3]}$, Public Keys $pk_{j\in[n+1]}$, Re-encryption key τ_{ind}, Encrypted garbled circuit and labels $\mathsf{CT}, \{\mathsf{CT}_{l,j}\}_{l\in[\lambda],j\in\{0,1\}}$, circuit C_0.
Input: $X = (x, y_0, y_1, .., y_{\mathsf{ind}-1}, \pi_1, ..., \pi_{\mathsf{ind}-1})$.

- Check that y_0 is a garbled circuit and set of labels (corresponding to input x), encrypted under pk_1. That is, $R(x, y_0) = 1$.
- Using the $\mathsf{SSSNIZK}$ verifier, check that $V_l(x, y_{l-1}, y_l, \pi_l) = 1$ for every $l \in [\mathsf{ind} - 1]$. If any check fails then abort.
- Compute randomness $\alpha_l \leftarrow \mathsf{F}(K_{i,l}, x)$ for $l \in [3]$.
- If $x \geq z$, let $Q = \mathsf{HE.HEEval}\left(pk_{\mathsf{ind}}, \mathsf{CT}^{\mathsf{ind}-1}, \{\mathsf{CT}^{\mathsf{ind}-1}_{k,x_k}\}_{k\in[\lambda]}, \mathsf{reRand}\left(;\alpha_1\right)\right)$. Compute $y_{\mathsf{ind}} \leftarrow \mathsf{HE.ReEncrypt}\left(\tau_{\mathsf{ind}}, Q; \alpha_2\right)$.
- If $x < z$, compute a simulated garbled circuit $(\tilde{\mathsf{GC}}, \tilde{\mathbf{w}}_x) \leftarrow \mathsf{Sim}(1^\lambda, |C|, C_0(x); \alpha_1)$. Encrypt $(\tilde{\mathsf{GC}}, \tilde{\mathbf{w}}_x)$ under pk_{ind} using randomness α_2 to get y_{ind}
- Compute a $\mathsf{SSSNIZK}$ proof π_{ind} for the statement that there exists a witness v such that $(x, y_{\mathsf{ind}-1}, y_{\mathsf{ind}}, v) \in L_{\mathsf{ind}}$. This is done by proving the second statement by using the trapdoor (opening of Z) as a witness. This proof is computed using the randomness $PRF_{K_{\mathsf{ind},3}}(x)$.
- Output $(y_{\mathsf{ind}}, \pi_{\mathsf{ind}})$

Figure 9: Circuit H_4

- Garble the circuit C_b, i.e. $(\mathsf{GC}, \mathbf{w}^z) \leftarrow \mathsf{Sim}(1^\lambda, |C_0|, C_0(z), r^z_{\mathsf{ind},1})$. Let $\mathbf{w}^z$ denote the labels for input z.
- Compute $y^z_{\mathsf{ind}} = \mathsf{HE.Enc}(pk_{\mathsf{ind}}, \mathsf{GC}, \mathbf{w}^z; r^z_{\mathsf{ind},2})$.

$\underline{\mathsf{Hyb}_{5.7.z}}$ for $z \in \{0,1\}^\lambda$: This hybrid is the same as the previous hybrid except that $r^z_{\mathsf{ind},1}$ and $r^z_{\mathsf{ind},2}$ are generated as $PRF_{K_{\mathsf{ind},1}}(z)$ and $PRF_{K_{\mathsf{ind},2}}(z)$.

$\underline{\mathsf{Hyb}_6}$: This hybrid is the same as the previous hybrid except that $\overline{G_{\mathsf{ind}}}$ is an obfuscation of the circuit (denoted as H_6) described in Figure 11.

$\underline{\mathsf{Hyb}_7}$: This hybrid is the same as the previous hybrid except that now $\mathsf{CT}, \mathsf{CT}^{i,j}$ for all $i \in [\lambda], j \in \{0,1\}$ encrypt 0.

Indistinguishability of Hybrids: We begin with some notation.

Notation: Let us assume that the secure candidate Π_{ind} be $\delta_{io}-$secure and the PRF be $\delta_{prf}-$secure. Also assume that the garbling scheme is atleast δ_{gb} re-randomization and simulation secure and that HE is atleast δ_{he}-secure. We also denote $\mathsf{SSSNIZK}$ to be $\delta_{\mathsf{SSSNIZK}}-$secure.

$\mathsf{H}_5\left[z, Z, \{Z_k, Z_{k,j}, K_{\mathsf{ind},j}\}_{k\in[n],j\in[3]}, pk_{j\in[n+1]}, \tau_{\mathsf{ind}}, \mathsf{CT}, \{\mathsf{CT}_{l,j}\}_{l\in[\lambda],j\in\{0,1\}}, a, C_0, y^z_{\mathsf{ind}}, \pi^z_{\mathsf{ind}}\right]$

Hardwired values: Point z, Commitments $Z, \{Z_k, Z_{k,j}\}_{k\in[n],j\in[2]}$, Opening a, Punctured PRF keys $\{K^z_{\mathsf{ind},j}\}_{j\in[2]}$, PRF key $K_{\mathsf{ind},3}$, Public Keys $pk_{j\in[n+1]}$, Re-encryption key τ_{ind}, Encrypted garbled circuit and labels $\mathsf{CT}, \{\mathsf{CT}_{l,j}\}_{l\in[\lambda],j\in\{0,1\}}$, hardwired output $(y^z_{\mathsf{ind}}, \pi^z_{\mathsf{ind}})$.
Input: $X = (x, y_0, y_1, .., y_{\mathsf{ind}-1}, \pi_1, ..., \pi_{\mathsf{ind}-1})$.

- Check that y_0 is a garbled circuit and set of labels (corresponding to input x), encrypted under pk_1. That is, $R(x, y_0) = 1$.
- Using the $\mathsf{SSSNIZK}$ verifier, check that $V_l(x, y_{l-1}, y_l, \pi_l) = 1$ for every $l \in [\mathsf{ind} - 1]$. If any check fails then abort.
- Parse $y_i = (\mathsf{CT}^{i-1}, \{\mathsf{CT}^{i-1}_{k,x_k}\}_{k\in[\lambda]})$ for $i \geq 1$.
- Compute randomness $\alpha_l \leftarrow \mathsf{F}(K_{\mathsf{ind},l}, x)$ for $l \in [3]$.
- If $x > z$, let $Q = \mathsf{HE.HEEval}\left(pk_{\mathsf{ind}}, \mathsf{CT}^{\mathsf{ind}-1}, \{\mathsf{CT}^{\mathsf{ind}-1}_{k,x_k}\}_{k\in[\lambda]}, \mathsf{reRand}\left(;\alpha_1\right)\right)$. Compute $y_{\mathsf{ind}} \leftarrow \mathsf{HE.ReEncrypt}\left(\tau_{\mathsf{ind}}, Q; \alpha_2\right)$.
- If $x < z$, compute a simulated garbled circuit $(\tilde{\mathsf{GC}}, \tilde{\mathbf{w}}_x) \leftarrow \mathsf{Sim}(1^\lambda, |C|, C_0(x); \alpha_1)$. Encrypt $(\tilde{\mathsf{GC}}, \tilde{\mathbf{w}}_x)$ under pk_{ind} using randomness α_2 to get y_{ind}
- If $x = z$ output $(y^z_{\mathsf{ind}}, \pi^z_{\mathsf{ind}})$.
- Compute a $\mathsf{SSSNIZK}$ proof π_{ind} for the statement that there exists a witness v such that $(x, y_{\mathsf{ind}-1}, y_{\mathsf{ind}}, v) \in L_{\mathsf{ind}}$. This is done by proving the second statement by using the trapdoor (opening of Z) as a witness. This proof is computed using the randomness $PRF_{K_{\mathsf{ind},3}}(x)$.
- Output $(y_{\mathsf{ind}}, \pi_{\mathsf{ind}})$

Figure 10: Circuit H_5

Lemma 11. *If* Com *is computationally hiding, then for any PPT adversary* $\mathcal{A}$*, it holds that* $\left|Pr\left[\mathcal{A}(\mathsf{Hyb}_1) = 1\right] - Pr\left[\mathcal{A}(\mathsf{Hyb}_2) = 1\right]\right| < \mathsf{negl}(\lambda)$.

Proof. The only difference in Hyb_1 and Hyb_2 is the manner in which Z is generated. In Hyb_1 it is generated as a commitment of 0 and in Hyb_2 it is generated as a commitment of ind. Note that the opening of Z is not used in any of the hybrids, hence the security follows from the security of the commitment scheme. □

Lemma 12. *If* Π_{ind} *is* $\delta_{io}-$*secure, then for any PPT adversary* $\mathcal{A}$*, it holds that* $\left|Pr\left[\mathcal{A}(\mathsf{Hyb}_2) = 1\right] - Pr\left[\mathcal{A}(\mathsf{Hyb}_{3.1.0}) = 1\right]\right| < \delta_{io}$.

Proof. The only difference in Hyb_2 and $\mathsf{Hyb}_{3.1.0}$ is the manner in which $\overline{G_i}$ is generated. In Hyb_2 it is generated as an obfuscation of circuit in Figure 5 while in $\mathsf{Hyb}_{3.1.0}$ it is an obfuscation of circuit H_1 described in Figure 2 (instantiated with $z = 0$) are equivalent. This is because on every input these circuits implement the same code. Proof now follows from the security of Π_{ind}. □

$\mathsf{H}_6\left[z, Z, \{Z_k, Z_{k,j}, K_{\mathsf{ind},j}\}_{k\in[n],j\in[3]}, pk_{j\in[n+1]}, \mathsf{CT}, \{\mathsf{CT}_{l,j}\}_{l\in[\lambda],j\in\{0,1\}}, a, C_0\right]$

Hardwired values: Point z, Commitments $Z, \{Z_k, Z_{k,j}\}_{k\in[n],j\in[2]}$, Opening a, PRF keys $\{K_{\mathsf{ind},j}\}_{j\in[3]}$, Public Keys $pk_{j\in[n+1]}$, Encrypted garbled circuit and labels $\mathsf{CT}, \{\mathsf{CT}_{l,j}\}_{l\in[\lambda],j\in\{0,1\}}$.
Input: $X = (x, y_0, y_1, .., y_{\mathsf{ind}-1}, \pi_1, ..., \pi_{\mathsf{ind}-1})$.

- Check that y_0 is a garbled circuit and set of labels (corresponding to input x), encrypted under pk_1. That is, $R(x, y_0) = 1$.
- Using the $\mathsf{SSSNIZK}$ verifier, check that $V_l(x, y_{l-1}, y_l, \pi_l) = 1$ for every $l \in [\mathsf{ind} - 1]$. If any check fails then abort.
- Compute randomness $\alpha_l \leftarrow \mathsf{F}(K_{\mathsf{ind},l}, x)$ for $l \in [3]$.
- Compute a simulated garbled circuit $(\tilde{\mathsf{GC}}, \tilde{\mathbf{w}}_x) \leftarrow \mathsf{Sim}(1^\lambda, |C|, C_0(x); \alpha_1)$. Encrypt $(\tilde{\mathsf{GC}}, \tilde{\mathbf{w}}_x)$ under pk_{ind} using randomness α_2 to get y_{ind}
- If $x = z$ output $(y^z_{\mathsf{ind}}, \pi^z_{\mathsf{ind}})$.
- Compute a $\mathsf{SSSNIZK}$ proof π_{ind} for the statement that there exists a witness v such that $(x, y_{\mathsf{ind}-1}, y_{\mathsf{ind}}, v) \in L_{\mathsf{ind}}$. This is done by proving the second statement by using the trapdoor (opening of Z) as a witness. This proof is computed using the randomness $PRF_{K_{\mathsf{ind},3}}(x)$.
- Output $(y_{\mathsf{ind}}, \pi_{\mathsf{ind}})$

Figure 11: Circuit H_6

Lemma 13. *If PRF is correct and Π_{ind} is $\delta_{io}-$secure, then for any PPT adversary $\mathcal{A}$, it holds that* $\left|Pr\left[\mathcal{A}(\mathsf{Hyb}_{3.1.z}) = 1\right] - Pr\left[\mathcal{A}(\mathsf{Hyb}_{3.2.z}) = 1\right]\right| < \delta_{io}$.

Proof. The only difference $\mathsf{Hyb}_{3.1.z}$ and $\mathsf{Hyb}_{3.2.z}$ is the way $\overline{G_{\mathsf{ind}}}$ is generated. In $\mathsf{Hyb}_{3.1.z}$ it is generated as an obfuscation of H_1 (instantiated with z) while in $\mathsf{Hyb}_{3.2.z}$ it is generated as an obfuscation of H_2 (instantiated with z). These circuits execute at the same code except at inputs of the form $(z, y_0, .., y_{\mathsf{ind}-1}, \pi_0, .., \pi_{\mathsf{ind}-1})$ where H_2 uses a PRF key punctured at z.
At this point the output of H_2 is hard-coded to be the same as that of H_1. The security now follows from the security of Π_{ind}. □

Lemma 14. *If the PRF is $\delta_{prf}-$secure, then for any PPT adversary $\mathcal{A}$, it holds that* $\left|Pr\left[\mathcal{A}(\mathsf{Hyb}_{3.2.z}) = 1\right] - Pr\left[\mathcal{A}(\mathsf{Hyb}_{3.3.z}) = 1\right]\right| < \delta_{prf}$.

Proof. Note that in both these hybrids $\mathsf{Hyb}_{3.2.z}$ and $\mathsf{Hyb}_{3.3.z}$ PRF key $K_{\mathsf{ind},3}$ is absent and both these hybrids depend only on the key punctured at z, K^z_{ind} and a string r^z_{ind}. In $\mathsf{Hyb}_{3.2.z}$ r^z_{ind} is sampled randomly while in $\mathsf{Hyb}_{3.2.z}$ it is generated as the evaluation of the PRF using the (unpunctured) key Kind at z. As the puncturable PRF is $\delta_{prf}-$secure, the lemma follows □

Lemma 15. *If the $\mathsf{SSSNIZK}$ is $\delta_{\mathsf{SSSNIZK}}-$secure, then for any $z \in \{0,1\}^\lambda$ any PPT adversary $\mathcal{A}$, it holds that* $\left|Pr\left[\mathcal{A}(\mathsf{Hyb}_{3.3.z}) = 1\right] - Pr\left[\mathcal{A}(\mathsf{Hyb}_{3.4.z}) = 1\right]\right| < \delta_{\mathsf{SSSNIZK}}$.

Proof. The only difference between hybrids $\mathsf{Hyb}_{3.3.z}$ and $\mathsf{Hyb}_{3.4.z}$ is the way the proof π^z_{ind} is generated. In both hybrids the proof is generated for the statement but the witness used is different. The security now follows from the witness indistinguishability property of the SSSNIZK. Witness indistinguishable property follows from the zero knowledge property via an intermediate hybrid. First the proof is simulated and in the next hybrid the proof uses the other witness. □

Lemma 16. *If the PRF is $\delta_{prf}-$secure then for all $z \in \{0,1\}^\lambda$, then for any PPT adversary $\mathcal{A}$, it holds that $\left|Pr\left[\mathcal{A}(\mathsf{Hyb}_{3.4.z}) = 1\right] - Pr\left[\mathcal{A}(\mathsf{Hyb}_{3.5.z}) = 1\right]\right| < \delta_{prf}$.*

Proof. Note that in both these hybrids $\mathsf{Hyb}_{3.4.z}$ and $\mathsf{Hyb}_{3.5.z}$ PRF key $K_{\mathsf{ind},3}$ is absent and both these hybrids depend only on the key punctured at z, K^z_{ind} and a string r^z_{ind}. In $\mathsf{Hyb}_{3.4.z}$ r^z_{ind} is sampled randomly while in $\mathsf{Hyb}_{3.5.z}$ it is generated as the evaluation of the PRF using the (unpunctured) key Kind at z. As the puncturable PRF is $\delta_{prf}-$secure, the lemma follows □

Lemma 17. *If the PRF is correct and $\Pi_{\mathsf{ind}}-$secure then for all $z \in \{0,1\}^\lambda$, then for any PPT adversary $\mathcal{A}$, it holds that $\left|Pr\left[\mathcal{A}(\mathsf{Hyb}_{3.5.z}) = 1\right] - Pr\left[\mathcal{A}(\mathsf{Hyb}_{3.1.z+1}) = 1\right]\right| < \delta_{io}$.*

Proof. Note that hybrids $\mathsf{Hyb}_{3.1.z+1}$ and $\mathsf{Hyb}_{3.5.z}$ differ only in the way $\overline{G_{\mathsf{ind}}}$ is generated. In hybrid $\mathsf{Hyb}_{3.1.z+1}$ it is generated as an obfuscation of circuit H_1 described in Figure 6 (instantiated with $z+1$) while in hybrid $\mathsf{Hyb}_{3.5.z}$ it is generated as an obfuscation of circuit H_2 described in Figure 7. These two programs execute the same code except that at input $(z, y_0, y_1, .., y_{\mathsf{ind}-1}, \pi_0, .., \pi_{\mathsf{ind}-1})$ where the output of the circuit in $\mathsf{Hyb}_{3.5.z}$ are hard-wired to be the same as in $\mathsf{Hyb}_{3.1.z+1}$ □

Lemma 18. *If the Π_{ind} is $\delta_{io}-$secure then for any PPT adversary $\mathcal{A}$, it holds that $\left|Pr\left[\mathcal{A}(\mathsf{Hyb}_{3.1.2^\lambda}) = 1\right] - Pr\left[\mathcal{A}(\mathsf{Hyb}_4) = 1\right]\right| < \delta_{io}$.*

Proof. Note that hybrids $\mathsf{Hyb}_{3.1.2^\lambda}$ and Hyb_4 differ only in the way $\overline{G_{\mathsf{ind}}}$ is generated. In hybrid $\mathsf{Hyb}_{3.1.z+1}$ it is generated as an obfuscation of circuit H_1 described in Figure 6 (instantiated with $z+1$) while in hybrid Hyb_4 it is generated as an obfuscation of circuit H_3 described in Figure 8. These two programs execute the same code at all inputs hence are functionally equivalent. Security now follows from the security of the candidate Π_{ind}. □

Lemma 19. *If the Com is computationally hiding then for any PPT adversary $\mathcal{A}$, it holds that $\left|Pr\left[\mathcal{A}(\mathsf{Hyb}_4) = 1\right] - Pr\left[\mathcal{A}(\mathsf{Hyb}_{5.0}) = 1\right]\right| < \mathsf{negl}(\lambda)$.*

Proof. The only difference between these two hybrids is the way the commitments $Z_{\mathsf{ind}}, Z_{\mathsf{ind},1}, Z_{\mathsf{ind},2}$ are generated. In Hyb_4 they are generated as commitments of $\tau_{\mathsf{ind}}, K_{\mathsf{ind},1}, K_{\mathsf{ind},2}$ respectively while in $\mathsf{Hyb}_{5.0}$ it is generated as a commitment of 0. The security now holds due to computational hiding property of the commitment scheme. □

Lemma 20. *If the Π_{ind} is $\delta_{io}-$secure then for any PPT adversary $\mathcal{A}$, it holds that $\left|Pr\left[\mathcal{A}(\mathsf{Hyb}_{5.1.0}) = 1\right] - Pr\left[\mathcal{A}(\mathsf{Hyb}_{5.0}) = 1\right]\right| < \delta_{io}$.*

Proof. Note that hybrids $\mathsf{Hyb}_{5.1.0}$ and $\mathsf{Hyb}_{5.0}$ differ only in the way $\overline{G_{\mathsf{ind}}}$ is generated. In hybrid $\mathsf{Hyb}_{5.1.0}$ it is generated as an obfuscation of circuit H_4 described in Figure 9 (instantiated with $z = 0$) while in hybrid $\mathsf{Hyb}_{5.0}$ it is generated as an obfuscation of circuit H_3 described in Figure 8.

These two programs execute the same code at all inputs hence are functionally equivalent. Security now follows from the security of the candidate Π_{ind}. □

Lemma 21. *If the* Π_{ind} *is* $\delta_{io}-$*secure then for any* $z \in \{0,1\}^\lambda$ *PPT adversary* $\mathcal{A}$*, it holds that* $\left|Pr\left[\mathcal{A}(\mathsf{Hyb}_{5.1.z}) = 1\right] - Pr\left[\mathcal{A}(\mathsf{Hyb}_{5.2.z}) = 1\right]\right| < \delta_{io}$.

Proof. Note that hybrids $\mathsf{Hyb}_{5.1.z}$ and $\mathsf{Hyb}_{5.2.z}$ differ only in the way $\overline{G_{\mathsf{ind}}}$ is generated. In hybrid $\mathsf{Hyb}_{5.1.z}$ it is generated as an obfuscation of circuit H_4 described in Figure 9 (instantiated with z) while in hybrid $\mathsf{Hyb}_{5.2.z}$ it is generated as an obfuscation of circuit H_5 described in Figure 10. The only difference between these programs is the fact that H_5 uses PRF keys $K^z_{\mathsf{ind},1}$ and $K^z_{\mathsf{ind},2}$ punctured at z. At all points $(x, y_0, .., y_{\mathsf{ind}-1}, \pi_0, .., \pi_{\mathsf{ind}-1})$ where $x \neq z$ the circuit outputs are same because of correctness of the puncturable PRF. At remaining points H_5 uses the evaluation of the PRF's at z, $r^z_{\mathsf{ind},1}$ and $r^z_{\mathsf{ind},2}$ to ensure same outputs. Security now follows from the security of the candidate Π_{ind}. □

Lemma 22. *If the PRF is* $\delta_{prf}-$*secure then for all* $z \in \{0,1\}^\lambda$*, then for any PPT adversary* $\mathcal{A}$*, it holds that* $\left|Pr\left[\mathcal{A}(\mathsf{Hyb}_{5.2.z}) = 1\right] - Pr\left[\mathcal{A}(\mathsf{Hyb}_{5.3.z}) = 1\right]\right| < \delta_{prf}$.

Proof. Note that in both these hybrids $\mathsf{Hyb}_{5.2.z}$ and $\mathsf{Hyb}_{5.3.z}$ PRF keys $K_{\mathsf{ind},1}, K_{\mathsf{ind},2}$ is absent and both these hybrids depend only on the key punctured at z, $K^z_{\mathsf{ind},1}, K^z_{\mathsf{ind}}$ and a strings $r^z_{\mathsf{ind},1}, r^z_{\mathsf{ind},2}$. In $\mathsf{Hyb}_{5.3.z}$, they are sampled randomly while in $\mathsf{Hyb}_{5.2.z}$ they generated as the evaluation of the PRF using the (unpunctured) keys at z. As the puncturable PRF is $\delta_{prf}-$secure, the lemma follows □

Lemma 23. *If* HE *is* $\delta_{he}-$*re-randomizable secure, then for any PPT adversary* $\mathcal{A}$*, it holds that* $\left|Pr\left[\mathcal{A}(\mathsf{Hyb}_{5.3.z}) = 1\right] - Pr\left[\mathcal{A}(\mathsf{Hyb}_{5.4.z}) = 1\right]\right| < \delta_{he}$.

Proof. The only difference between these hybrids is the way y^z_{ind} is generated. In $\mathsf{Hyb}_{5.3.z}$, it is generated as a re-encryption of a cipher-text encrypting of Q while in $\mathsf{Hyb}_{5.4.z}$ it is generated as a fresh encryption of Q under pk_{ind}. The security now holds from the security of the re-encryption scheme. □

Lemma 24. *If* Garble *is* $\delta_{gb}-$*re-randomizable secure, then for any PPT adversary* $\mathcal{A}$*, it holds that* $\left|Pr\left[\mathcal{A}(\mathsf{Hyb}_{5.4.z}) = 1\right] - Pr\left[\mathcal{A}(\mathsf{Hyb}_{5.5.z}) = 1\right]\right| < \delta_{gb}$.

Proof. The only difference between the hybrids $\mathsf{Hyb}_{5.4.z}$ and $\mathsf{Hyb}_{5.5.z}$ is the way Q is generated. In $\mathsf{Hyb}_{5.4.z}$ it is a re-ranomized garbled circuit and labels for input z (denoted by $y_{\mathsf{ind}-1}$). In $\mathsf{Hyb}_{5.5.z}$ it is a freshly generated garbled circuit for C_b. The security now holds from the re-randomization security of the garbling scheme. □

Lemma 25. *If* Garble *is* $\delta_{gb}-$*secure, then for any PPT adversary* $\mathcal{A}$*, it holds that* $\left|Pr\left[\mathcal{A}(\mathsf{Hyb}_{5.5.z}) = 1\right] - Pr\left[\mathcal{A}(\mathsf{Hyb}_{5.6.z}) = 1\right]\right| < \delta_{gb}$.

Proof. The only difference between the hybrids $\mathsf{Hyb}_{5.5.z}$ and $\mathsf{Hyb}_{5.6.z}$ is the way Q is generated. In $\mathsf{Hyb}_{5.5.z}$ it is freshly generated grbled circuit for C_b and labels corresponding to input z. In $\mathsf{Hyb}_{5.6.z}$ it is generated as a simulated garbled circuit and labels constructed using $C_b(x) = C_0(x)$. □

Lemma 26. *If PRF is $\delta_{prf}-$secure, then for any PPT adversary $\mathcal{A}$, it holds that* $\Big|Pr\big[\mathcal{A}(\mathsf{Hyb}_{5.6.z}) = 1\big] - Pr\big[\mathcal{A}(\mathsf{Hyb}_{5.7.z}) = 1\big]\Big| < \delta_{prf}$.

Proof. Note that in both these hybrids $\mathsf{Hyb}_{5.6.z}$ and $\mathsf{Hyb}_{5.7.z}$ PRF keys $K_{\mathsf{ind},1}, K_{\mathsf{ind},2}$ is absent and both these hybrids depend only on the keys punctured at z, $K^z_{\mathsf{ind},1}, K^z_{\mathsf{ind},2}$ and a strings $r^z_{\mathsf{ind},1}, r^z_{\mathsf{ind},2}$. In $\mathsf{Hyb}_{5.6.z}$ $r^z_{\mathsf{ind},1}, r^z_{\mathsf{ind},2}$ are sampled randomly while in $\mathsf{Hyb}_{5.7.z}$ they are generated as the evaluation of the PRF using the (unpunctured) keys at z. As the puncturable PRF is $\delta_{prf}-$secure, the lemma follows □

Lemma 27. *If Π_{ind} is $\delta_{io}-$secure, then for any PPT adversary $\mathcal{A}$, it holds that* $\Big|Pr\big[\mathcal{A}(\mathsf{Hyb}_{5.7.z}) = 1\big] - Pr\big[\mathcal{A}(\mathsf{Hyb}_{5.1.z+1}) = 1\big]\Big| < \delta_{io}$.

Proof. Note that hybrids $\mathsf{Hyb}_{5.1.z+1}$ and $\mathsf{Hyb}_{5.7.z}$ differ only in the way $\overline{G_{\mathsf{ind}}}$ is generated. In hybrid $\mathsf{Hyb}_{5.1.z+1}$ it is generated as an obfuscation of circuit H_4 described in Figure 9 (instantiated with $z+1$) while in hybrid $\mathsf{Hyb}_{5.7.z}$ it is generated as an obfuscation of circuit H_5 described in Figure 10. These two programs execute the same code except that at input $(z, y_0, y_1, .., y_{\mathsf{ind}-1}, \pi_0, .., \pi_{\mathsf{ind}-1})$ where the output of the circuit in $\mathsf{Hyb}_{5.7.z}$ are hard-wired to be the same as in $\mathsf{Hyb}_{5.1.z+1}$. Observe that H_5 uses punctured PRF keys $K^z_{\mathsf{ind},1}$ and $K^z_{\mathsf{ind},2}$ but this key is never evaluated at z and hence these two circuits are equivalent. The security now follows from the security of Π_{ind}. □

Lemma 28. *If Π_{ind} is $\delta_{io}-$secure, then for any PPT adversary $\mathcal{A}$, it holds that* $\Big|Pr\big[\mathcal{A}(\mathsf{Hyb}_{5.1.2^\lambda}) = 1\big] - Pr\big[\mathcal{A}(\mathsf{Hyb}_6) = 1\big]\Big| < \delta_{io}$.

Proof. Note that hybrids $\mathsf{Hyb}_{5.1.2^\lambda}$ and Hyb_6 differ only in the way $\overline{G_{\mathsf{ind}}}$ is generated. In hybrid $\mathsf{Hyb}_{5.1.2^\lambda}$ it is generated as an obfuscation of circuit H_4 described in Figure 9 (instantiated with $z = 2^\lambda$) while in hybrid Hyb_6 it is generated as an obfuscation of circuit H_6 described in Figure 11. These two programs execute the same code at all inputs hence are functionally equivalent. Security now follows from the security of the candidate Π_{ind}. □

Lemma 29. *If HE is $\delta_{he}-$secure, then for any PPT adversary $\mathcal{A}$, it holds that* $\Big|Pr\big[\mathcal{A}(\mathsf{Hyb}_6) = 1\big] - Pr\big[\mathcal{A}(\mathsf{Hyb}_7) = 1\big]\Big| < \delta_{io}$.

Proof. Note that hybrids Hyb_6 and Hyb_7 differ only in the way $\mathsf{CT}, \mathsf{CT}_{i,j} \forall i \in [\lambda], j \in \{0,1\}$ is generated. In hybrid Hyb_6 it is generated as an obfuscation of a garbled circuit C_b and in Hyb_7 it is generated as an encryption of 0. Note that the hybrids consists of $(pk, \tau_1, .., \tau_{\mathsf{ind}-1}, \tau_{\mathsf{ind}+1}, .., sk_n)$. Since τ_{ind} is missing, the security holds by the security of re-encryption scheme. □

We now note that the advantage of adversary in Hyb_7 is 0. This is because Hyb_7 is independent of b. To complete the security proof we sum up the advantages and claim that the sum is negligible. Summing up advantages, we get that the advantage of adversary in the obfuscation security game is bounded by $2^{\lambda+2}\delta_{io} + 2^{\lambda+2}\delta_{prf} + 2^\lambda\delta_{\mathsf{SSSNIZK}} + 2^\lambda\delta_{he} + 2^{\lambda+1}\delta_{gb} + \mathsf{negl}(\lambda)$. Setting $\delta_{he}, \delta_{prf}, \delta_{gb}, \delta_{io}, \delta_{\mathsf{SSSNIZK}}$ to be $O(2^{-2\lambda})$ we get our claim.

□

6 Universal Obfuscation

We introduce the notion of universal obfuscation. We define a pair of Turing machines $\Pi_{\mathsf{univ}}.\mathsf{Obf}$ and $\Pi_{\mathsf{univ}}.\mathsf{Eval}$ to be an universal obfuscation if the existence of a secure IO candidate implies that $(\Pi_{\mathsf{univ}}.\mathsf{Obf}, \Pi_{\mathsf{univ}}.\mathsf{Eval})$ is also a secure IO candidate. Constructing an universal obfuscation scheme means that we can turn the mere existence of a secure IO candidate into an explicit construction. Formally,

Definition 7 ((T,ϵ)-Universal Obfuscation)**.** *We say that a pair of Turing machines* $\Pi_{\mathsf{univ}} = (\Pi_{\mathsf{univ}}.\mathsf{Obf}, \Pi_{\mathsf{univ}}.\mathsf{Eval})$ *is a* **universal obfuscation***, parameterized by* T *and* ϵ*, if there exists an* ϵ*-secure indistinguishability obfuscator for P/poly with time function* T *then* Π_{univ} *is an indistinguishability obfuscator for P/poly with time function* $\mathrm{poly}(T)$*.*

We present a construction of (T,ϵ)-universal obfuscation next.

6.1 Construction of (T,ϵ)-Universal Obfuscation

We proceed to construct a (T,ϵ)-universal obfuscation. The core building block in our construction is a *decomposable* IO combiner – this is a specific type of IO combiner that satisfies additional properties (explained below). Once we have this tool, we then construct a (T,ϵ)-universal obfuscation that is approximately correct. That is, for every circuit, the corresponding (universal) obfuscated circuit agrees with the original circuit on a significant fraction of the inputs. We then apply the transformation of Bitansky-Vaikuntanathan [BV16] to obtain a universal obfuscation scheme that is exact. We flesh out the technical details below.

Main Ingredient: Decomposable IO Combiner. A decomposable IO combiner is a type of IO combiner, where the obfuscate algorithm has a specific structure. In particular, the obfuscate algorithm takes as input circuit C to be obfuscated, the description of the candidates $\Pi_1, \ldots, \Pi_n$ and executes in two main steps. In the first step, circuit C is preprocessed into n circuits $[\mathbf{C}]_1, \ldots, [\mathbf{C}]_n$. In the second step, each individual circuit $[\mathbf{C}]_i$ is obfuscated using the candidate Π_i. The concatenation of the resulting obfuscated circuits is the final output.

In addition to the standard properties of IO combiner, we require that the decomposable IO combiner satisfies more properties:

- *Circuit-Specific Correctness*: We require that if the underlying candidate Π_i is approximately-correct (say, η-correct) on the *specific circuit* $[\mathbf{C}]_i$ and this holds for every $i \in [n]$, then the IO combiner is approximately-correct (say, η'-correct where η' is a function of η) on the original circuit C. This is different from the correctness property (Definition 6) where it was required that Π_i is approximately-correct on *every circuit*.

- *Decomposable Security*: This property states that any $n-1$ of $\{[\mathbf{C}]_1, \ldots, [\mathbf{C}]_n\}$ hides the original circuit C. The hiding property is formalized in the form of an indistinguishability experiment. Consider two arbitrary circuits C_0, C_1 – in particular, they need not be equivalent. Then, the distribution of $\{[\mathbf{C}]_1^0, \ldots, [\mathbf{C}]_n^0\} \setminus \{[\mathbf{C}]_{\mathbf{i}}^0\}$ derived from C_0 is computationally indistinguishable from the distribution of $\{[\mathbf{C}]_1^1, \ldots, [\mathbf{C}]_n^1\} \setminus \{[\mathbf{C}]_{\mathbf{i}}^1\}$ derived from C_1 for every $i \in [n]$.

We present the formal definition below.

Definition 8 (Decomposable IO Combiner)**.** *A (ϵ', ϵ)-secure IO combiner $\Pi_{\mathsf{comb}} = (\Pi_{\mathsf{comb}}.\mathsf{Obf}, \Pi_{\mathsf{comb}}.\mathsf{Eval})$ of $(\Pi_1, \ldots, \Pi_n)$ for a class of circuits $\mathcal{C} = \{\mathcal{C}_\lambda\}$ is said to be* (ϵ', ϵ)**-secure** (η', η)**-decomposable IO combiner** *if there exists a PPT algorithm* $\mathsf{Preproc}$ *such that the following holds:* $\Pi_{\mathsf{comb}}.\mathsf{Obf}$ *on input* $(1^\lambda, C \in \mathcal{C}_\lambda, \Pi_1, \ldots, \Pi_n)$ *executes the steps:*

(a) (Preprocessing step) $\overline{C} = ([\mathbf{C}]_1, \ldots, [\mathbf{C}]_n, aux) \leftarrow \mathsf{Preproc}(1^\lambda, 1^n, C)$,

(b) (Candidate Obfuscation step) for all $i \in [n]$, $\overline{[\mathbf{C}]_i} \leftarrow \Pi_i.\mathsf{Obf}(1^\lambda, [\mathbf{C}]_i)$,

(c) Outputs $\overline{C} = \left(\overline{[\mathbf{C}]_1}, \ldots, \overline{[\mathbf{C}]_n}, aux\right)$.

Additionally, we require the following properties to hold:

- (η', η)**-Circuit-Specific Correctness.** *Consider a circuit* $C \in \mathcal{C}_\lambda$. *Let* $([\mathbf{C}]_1, \ldots, [\mathbf{C}]_n, aux) \leftarrow \mathsf{Preproc}(1^\lambda, 1^n, C)$. *Let for all* $i \in [n]$, $\overline{[\mathbf{C}]_i} \leftarrow \Pi_i.\mathsf{Obf}(1^\lambda, [\mathbf{C}]_i)$. *Denote* $\overline{C} = (\overline{[\mathbf{C}]_1}, \ldots, \overline{[\mathbf{C}]_n})$

  If for all $i \in [n]$, $\underset{x \xleftarrow{\$} \{0,1\}^\lambda}{\mathsf{Pr}} \left[\overline{[\mathbf{C}]_i}(x) = [\mathbf{C}]_i(x) \right] \geq \eta(\lambda)$ *then*

  $\underset{x \xleftarrow{\$} \{0,1\}^\lambda}{\mathsf{Pr}} \left[\overline{C}(x) = C(x) \right] \geq \eta'(\lambda)$.

- **Decomposable Security***: For every* $C_0, C_1 \in \mathcal{C}_\lambda$ *such that* $|C_0| = |C_1|$, *for every* $\mathbf{i} \in [n]$, *we have:*

$$\left\{ \left\{ [\mathbf{C}]_i^0 \right\}_{\substack{i \neq \mathbf{i}, \\ i \in [n]}} \right\} \approx_c \left\{ \left\{ [\mathbf{C}]_i^1 \right\}_{\substack{i \neq \mathbf{i}, \\ i \in [n]}} \right\},$$

where $[\mathbf{C}]_i^b \leftarrow \mathsf{Preproc}(1^\lambda, 1^n, C_b \in \mathcal{C}_\lambda)$ *for* $b \in \{0, 1\}$.

We claim that the construction of IO Combiner in Section 5 is already a decomposable IO combiner. To show this, we first note that the obfuscator $\Pi_{\mathsf{univ}}.\mathsf{Obf}$ in the construction in Section 5 can be decomposed in a preprocessing step and candidate obfuscation step: the preprocessing step comprises of all the steps till the generation of circuits $\{G_i\}_{i \in [n]}$ (Figure 1). The output of the preprocessing step is $(G_1, \ldots, G_n)$.

Furthermore, the circuit-specific correctness property was already proved in Lemma 1. More specifically, we showed the aforementioned construction satisfies $(1 - n\mu, 1 - \mu)$-circuit specific correctness property. All is remaining is to show that the construction satisfies decomposable security. To show that, we prove the following claim.

Claim 1. *Consider the following process:*

- $\mathsf{params} \leftarrow \mathsf{Setup}(1^\lambda, 1^d)$, *where* $d = \mathrm{poly}(\lambda, |C|)$
- $\{(sk_i, pk_i) \leftarrow \mathsf{KeyGen}(\mathsf{params})\}_{i \in [n]}$
- *Sample* n *random strings* $\{S_i^b\}_{i \in [n]}$ *of size* $| C_b |$, *for* $b \in \{0, 1\}$, *such that* $\bigoplus_{i \in [n]} S_i^b = C$.
- *For all* $i \in [n]$, *encrypt the string* S_i *using* pk_i, $\mathsf{CT}_i^b \leftarrow \mathsf{Enc}(pk_i, S_i^b)$.

- *For every* $i \in [n]$, *execute* $\widehat{\mathsf{CT}_i^b} \leftarrow \mathsf{Expand}((pk_1, \ldots, pk_n), i, \mathsf{CT}_i)$.

For any $\mathbf{i} \in [n]$ *we have:*

$$\{G_1^0, \ldots, G_n^0\} \backslash \{G_{\mathbf{i}}^0\} \approx_c \{G_1^1, \ldots, G_n^1\} \backslash \{G_{\mathbf{i}}^1\},$$

assuming the semantic security of TMFHE, where $G_i^b = G_i^b\left[K^i, sk_i, \{pk_i\}_{i\in[n]}, \{\widehat{\mathsf{CT}_i^b}\}_{i\in[n]}\right]$ *is defined in Figure 1.*

Proof. Suppose there is a PPT adversary $\mathcal{A}$ that distinguishes the two distributions in the statement of the claim. We design a reduction $\mathcal{B}$, that internally uses $\mathcal{A}$ to break the security of TMFHE. $\mathcal{B}$ requests the TMFHE public keys $\{pk_i\}_{i\in[n]}$, secret keys $\{sk_1, \ldots, sk_n\} \backslash \{sk_i\}$, ciphertexts $\{\mathsf{CT}_i = \mathsf{Enc}(pk_i, S_i^0)\}_{i\neq \mathbf{i}}$ and challenge ciphertext $\mathsf{CT}_{\mathbf{i}}^*$, where $\mathsf{CT}_{\mathbf{i}} = \mathsf{CT}_{\mathbf{i}}^*$ is either $\mathsf{Enc}(pk_{\mathbf{i}}, S_{\mathbf{i}}^0)$ or $\mathsf{Enc}(pk_{\mathbf{i}}, R)$, where R is a random string. Compute the expanded ciphertext $\widehat{\mathsf{CT}_i^b} \leftarrow \mathsf{Expand}((pk_1, \ldots, pk_n), i, \mathsf{CT}_i)$ for every $i \in [n]$. Generate the puncturable PRF keys K^i for every $i \in [n]$. Finally, generate the circuits $G_i = G_i[K^i, sk_i, \{pk_i\}_{i\in[n]}, \{\widehat{\mathsf{CT}_i^b}\}_{i\in[n]}]$ for every $i \neq \mathbf{i}$. Note that we cannot generate $G_{\mathbf{i}}$ since the secret key $sk_{\mathbf{i}}$ is missing. We hand over the circuits $\{G_1, \ldots, G_n\} \backslash \{G_{\mathbf{i}}\}$ to $\mathcal{A}$. If the output of $\mathcal{A}$ is a bit b, output b.

If $\mathsf{CT}_{\mathbf{i}}^* = \mathsf{Enc}(pk_{\mathbf{i}}, S_{\mathbf{i}}^0)$ then the circuits G_i's are derived from a circuit C and in the other case when $\mathsf{CT}_{\mathbf{i}}^* = \mathsf{Enc}(pk_{\mathbf{i}}, R)$, G_i's correspond to a random circuit (of size $|C|$). Thus, the distinguishing advantage of $\mathcal{A}$ translates directly to the distinguishing advantage of $\mathcal{B}$. □

We thus have the following theorem.

Theorem 11. *The construction presented in Section 5 is a* $(\mathsf{negl}, \epsilon)$*-secure* $(1 - \frac{1}{\lambda}, 1 - \frac{1}{\lambda^2})$*-decomposable IO combiner, where the number of candidates is* λ*.*

Step I: Construction of Approx. Correct (T, ϵ)-Universal Obfuscation. We construct a universal obfuscation scheme $\Pi_{\mathsf{univ}} = (\mathsf{Obf}, \mathsf{Eval})$ for a class of circuits $\mathcal{C}$ below. Our scheme will be approximately correct. The main ingredient is a decomposable IO combiner (Definition 8) $\Pi_{\mathsf{comb}} = (\Pi_{\mathsf{comb}}.\mathsf{Obf}, \Pi_{\mathsf{comb}}.\mathsf{Eval})$ for $\mathcal{C}$. But first, we establish some notation.

Notation. Let $\mathcal{S}$ be the class of all possible Turing machines. It is well known result [Göd31] that there is a one-to-one correspondence between $\mathcal{S}^2$ and $\mathbb{N}$ given by $\phi : \mathbb{N} \to \mathcal{S}^2$. Furthermore, there is a fixed polynomial f such that the time to compute $\phi(j)$ is at most $\leq f(j)$, for every $j \in \mathbb{N}$.

$\Pi_{\mathsf{univ}}.\mathsf{Obf}(1^\lambda, C)$: It takes as input security parameter λ, circuit $C \in \mathcal{C}_\lambda$ and executes the following steps:

1. Let $\phi(i) = (\Pi_i.\mathsf{Obf}, \Pi_i.\mathsf{Eval})$, for $i \in \{1, \ldots, \lambda\}$. Denote $\Pi_i = (\Pi_i.\mathsf{Obf}, \Pi_i.\mathsf{Eval})$.

2. **Preprocessing phase of Decomposable IO combiner.** First compute the preprocessing step, $([\mathbf{C}]_1, \ldots, [\mathbf{C}]_n, aux) \leftarrow \mathsf{Preproc}(1^\lambda, 1^n, C)$ $(n = \lambda)$.

3. **Eliminating Candidates with Large Runtimes.** For all $i \in [\lambda]$, execute $\Pi_i.\mathsf{Obf}(1^\lambda, [\mathbf{C}]_i)$ for at most $t = T\left(\lambda, \left|[\mathbf{C}]_i\right|\right)$ number of steps. For every $i \in [\lambda]$, if the computation of $\Pi_i.\mathsf{Obf}\left(1^\lambda, [\mathbf{C}]_i\right)$ does not abort within t number of time steps re-assign $\Pi_i.\mathsf{Obf} = I$ and $\Pi_i.\mathsf{Eval} = UTM$, where I is an identity TM[13] and UTM is an universal TM[14].

   At the end of this step, the execution of $\Pi_i.\mathsf{Obf}(1^\lambda, [\mathbf{C}]_i)$ takes time at most $T\left(\lambda, \left|[\mathbf{C}]_i\right|\right)$.

4. **Eliminates Candidates with Imperfect Correctness.** For all $i \in [\lambda]$, execute $\Pi_i.\mathsf{Obf}(1^\lambda, [\mathbf{C}]_i)$ for at most $t = T(\lambda, \left|[\mathbf{C}]_i\right|)$ number of steps. Denote $\overline{[\mathbf{C}]_i}$ to be the result of computation. Denote ℓ to be the input length of $[\mathbf{C}]_i$. For every $i \in [n]$, sample λ^3 points $x_{1,i}, \ldots, x_{\lambda^3,i} \xleftarrow{\$} \{0,1\}^\ell$. Check if the following condition holds:

$$\bigwedge_{j=1}^{\lambda^3} \left([\mathbf{C}]_i(x_{j,i}) = \Pi_i.\mathsf{Eval}\left(\overline{[\mathbf{C}]_i}, x_{j,i}\right) \right) = 1 \tag{1}$$

   If for any $i \in [\lambda]$ the above condition does not hold, re-assign $\Pi_i.\mathsf{Obf} = I$ and $\Pi_i.\mathsf{Eval} = UTM$. At the end of this step, every candidate satisfies the above condition.

5. **Candidate Obfuscation Phase of Decomposable IO combiner.** For all $i \in [\lambda]$, execute $\Pi_i.\mathsf{Obf}\left(1^\lambda, [\mathbf{C}]_i\right)$ for at most $t = T(\lambda, \left|[\mathbf{C}]_i\right|)$ number of steps. Denote $\overline{[\mathbf{C}]_i}$ to be the result of computation.

6. Output $\overline{C} = \left((\Pi_1, \ldots, \Pi_\lambda),\ (\overline{[\mathbf{C}]_1}, \ldots, \overline{[\mathbf{C}]_\lambda}, aux)\right)$.

$\underline{\Pi_{\mathsf{univ}}.\mathsf{Eval}(\overline{C}, x)}$: On input the obfuscated circuit $\overline{C}$ and input x, do the following. First parse $\overline{C}$ as $\left((\Pi_1, \ldots, \Pi_\lambda),\ \overline{C_{comb}} = (\overline{[\mathbf{C}]_1}, \ldots, \overline{[\mathbf{C}]_\lambda}, aux)\right)$. Compute $y \leftarrow \Pi_{\mathsf{comb}}.\mathsf{Eval}\left(\overline{C_{comb}}, x, \Pi_1, \ldots, \Pi_\lambda\right)$. Output y.

Theorem 12. *Assuming that Π_{comb} is a $(\mathsf{negl}, \epsilon)$-secure $\left(1 - \frac{1}{\lambda}, 1 - \frac{1}{\lambda^2}\right)$-decomposable IO combiner, the above scheme Π_{univ} is a (T, ϵ)-universal obfuscation that is $\left(1 - \frac{1}{\lambda}\right)$-correct.*

Proof. We first remark about the running time of the obfuscator and the evaluator algorithms. First, we consider $\Pi_{\mathsf{univ}}.\mathsf{Obf}$. The running time of first step (Bullet 1) is $\lambda f(\lambda) = \mathrm{poly}(\lambda)$ (where f was defined earlier in the proof). The running time of each of the rest of the steps is $\mathrm{poly}(\lambda, t, |C|)$. Plugging in the fact that $t = T\left(\lambda, \mathrm{poly}(\lambda, |C|)\right)$, we have that the total running time of all the steps to be $\mathrm{poly}((T(\lambda, |C|))$[15]. We move on to $\Pi_{\mathsf{univ}}.\mathsf{Eval}$. Here, the running time is governed by the running time of the $\Pi_{\mathsf{comb}}.\mathsf{Eval}$ algorithm which is $\mathrm{poly}(T(\lambda, |C|))$. And hence, the running time of $\Pi_{\mathsf{univ}}.\mathsf{Eval}$ is again $\mathrm{poly}(T(\lambda, |C|))$.

[13] An identity TM on input C outputs C.

[14] An universal TM on input (C, x) outputs $C(x)$ if the computation terminates. Otherwise, the universal TM does not abort.

[15] Observe that here we used two facts of the time function (beginning of Section 3): (a) $T(\lambda, |C|) \geq |C| + \lambda$ and, (b) $T(\lambda, \mathrm{poly}(|C|)) = \mathrm{poly}'(T(\lambda, |C|))$.

Correctness. Consider the following lemma.

Lemma 30. Π_{univ} *is a* $\left(1-\frac{1}{\lambda}\right)$*-correct IO candidate.*

Proof. Consider a circuit $C \in \mathcal{C}_\lambda$. We prove the following claim. For all $i \in [n]$, let $([\mathbf{C}]_1, \ldots, [\mathbf{C}]_n, aux) \leftarrow \mathsf{Preproc}(1^\lambda, 1^n, C)$ with $n = \lambda$. Also, let $\{\Pi_i\}_{i\in[n]}$ be the description of the candidates at the end of Bullet 3. Note that some of the candidates could be re-assigned in Bullets 2 and 3. Let $\overline{[\mathbf{C}]_i} \leftarrow \Pi_i.\mathsf{Obf}(1^\lambda, [\mathbf{C}]_i)$.

Claim 2. *Let* $i \in [n]$ *be such that*

$$\Pr_{x \xleftarrow{\$} \{0,1\}^\lambda}\left[\ [\mathbf{C}]_i(x) = \Pi_i.\mathsf{Eval}(\overline{[\mathbf{C}]_i}, x)\right] \leq 1 - \frac{1}{\lambda^2}$$

Then, the i^{th} *candidate* Π_i *satisfies Condition (1) (Bullet 4) with* **negligible** *probability (over the random coins of* $x_{j,i}$*).*

Proof. Let $\mathbf{i} \in [n]$ be such that it satisfies the condition stated in the claim. We show that $[\mathbf{C}]_\mathbf{i}$ satisfies Condition (1) (Bullet 4) with negligible probability. We define the following random variables for every $j \in [\lambda^3]$:

$$\mathsf{InEQ}_j = \begin{cases} 1 & \text{if } [\mathbf{C}]_\mathbf{i}(x_{j,\mathbf{i}}) = \overline{[\mathbf{C}]_\mathbf{i}}(x_{j,\mathbf{i}}), \\ 0 & \text{otherwise,} \end{cases}$$

where $x_{j,\mathbf{i}} \in \{0,1\}^\lambda$. Thus, $\Pr[\mathsf{InEQ}_j = 1] < 1 - \frac{1}{\lambda^2}$.

Consider the following:

$$\begin{aligned} \Pr\left[\bigwedge_{j\in[\lambda^3]} \mathsf{InEQ}_j = 1\right] &= \prod_{j\in[\lambda^3]} \left(\Pr\left[\mathsf{InEQ}_j = 1\right]\right) \\ &< \left(1 - \frac{1}{\lambda^2}\right)^{\lambda^3} \\ &< e^{-\lambda} = \mathsf{negl}(\lambda) \end{aligned}$$

This proves the claim. □

The above claim proves that at the end of Bullet 4, with overwhelming probability the following holds for every $i \in [n]$:

$$\Pr_{x \xleftarrow{\$} \{0,1\}^\lambda}\left[\ [\mathbf{C}]_i(x) = \Pi_i.\mathsf{Eval}(\overline{[\mathbf{C}]_i}, x)\right] \geq 1 - \frac{1}{\lambda^2}$$

We now apply the circuit-specific completeness property of the $(1-\frac{1}{\lambda}, 1-\frac{1}{\lambda^2})$-decomposable IO combiner Π_{comb} which ensures that the following holds:

$$\Pr_{x \xleftarrow{\$} \{0,1\}^\lambda}\left[\ C(x) = \Pi_{\mathsf{comb}}.\mathsf{Eval}(\overline{C}, x)\ \right] \geq 1 - \frac{1}{\lambda}$$

where $\overline{C} = (\overline{[\mathbf{C}]_1}, \ldots, \overline{[\mathbf{C}]_n}, aux)$. Note that $\overline{C}$ is the output of $\Pi_{\mathsf{univ}}.\mathsf{Obf}$. Also, the output of $\Pi_{\mathsf{univ}}.\mathsf{Eval}$ on input $(\overline{C}, x)$ is dictated by the result of $\Pi_{\mathsf{comb}}.\mathsf{Eval}(\overline{C}, x)$.

Thus, we have

$$\Pr_{x \xleftarrow{\$} \{0,1\}^\lambda} \left[C(x) = \Pi_{\mathsf{univ}}.\mathsf{Eval}(\overline{C}, x) \right] \geq 1 - \frac{1}{\lambda},$$

where $\overline{C} \leftarrow \Pi_{\mathsf{univ}}.\mathsf{Obf}(1^\lambda, C)$. □

Security. We prove the following lemma.

Lemma 31. Π_{univ} *is a* (negl)*-secure IO candidate.*

Proof. Recall that the universal obfuscator proceeds in two phases. In the first phase, it chooses the "correct" candidates and then in the second phase, it combines all these candidates to produce the obfuscated circuit. At first glance, it should seem that as long as we ensure that one of the "correct" candidates is secure then the security of IO combiner, and thus the security of universal obfuscator, should follow. To make this more precise, lets say C_0 and C_1 are two equivalent circuits. Let $\overrightarrow{\Pi_0} = \Pi_1^0, \ldots, \Pi_{n_0}^0$ and $\overrightarrow{\Pi_1} = \Pi_1^1, \ldots, \Pi_{n_1}^1$ be the "correct" candidates chosen with respect to C_0 and C_1 respectively. Now, assuming that $\overrightarrow{\Pi_0}$ and $\overrightarrow{\Pi_0}$ have at least one secure candidate; the hope is that we can then invoke the security of IO combiner to argue computational indistinguishability of obfuscation of C_0 and C_1. This does not work because the security of IO combiner dictates that $\overrightarrow{\Pi_0} = \overrightarrow{\Pi_1}$. Indeed obfuscation of C_0 (resp., C_1) could potentially reveal $\overrightarrow{\Pi_0}$ (resp., $\overrightarrow{\Pi_1}$) at which point no security holds. While we cannot argue that $\overrightarrow{\Pi_0} = \overrightarrow{\Pi_1}$ – this is not true because of *selective abort obfuscators* described in Introduction – we can still show that $\overrightarrow{\Pi_0} \approx_c \overrightarrow{\Pi_1}$. Arguing the indistinguishability of the candidates then helps us invoke the security of IO combiner and then the proof of the theorem follows. Arguing the indistinguishability of candidates is performed by invoking the decomposable security property of the underlying IO combiner. The technical details are provided below.

Formal details. We first introduce some notation. Consider a circuit $C \in \mathcal{C}_\lambda$. Let $((\Pi_1, \ldots, \Pi_\lambda), (\overline{[\mathbf{C}]_1}, \ldots, \overline{[\mathbf{C}]_\lambda}), aux)$ be the output of $\Pi_{\mathsf{univ}}.\mathsf{Obf}(1^\lambda, C)$. Note that many of the candidates $(\Pi_1, \ldots, \Pi_\lambda)$ could potentially be re-assigned during the execution of $\Pi_{\mathsf{univ}}.\mathsf{Obf}$. This re-assignment is a function of the circuit C that is obfuscated and the random coins of the algorithm. Hence, we can define a distribution $\mathsf{Dist}_{C,\lambda}$, parameterized by C, λ, $\mathbf{i} \in [n]$, on $\{0,1\}^\lambda$ such that $x \xleftarrow{\$} \{0,1\}^\lambda$ defines which of the candidates gets re-assigned – the i^{th} bit $x_i = 1$ indicates that Π_i will remain unchanged and $x_i = 0$ indicates that the Π_i is re-assigned.

In more detail, we define the sampling algorithm of distribution $\mathsf{Dist}_{\lambda,C,\mathbf{i}}$ as follows: denote by $\Pi'_1, \ldots, \Pi'_\lambda$ the set of candidates enumerated in Bullet 1 and let $\Pi_{\mathbf{i}}$ be an IO candidate that is always correct. Note that the description of these candidates are independent of the circuit C and they only depend on the security parameter λ. At the end of Bullet 4, denote the candidates to be $(\Pi_1, \ldots, \Pi_\lambda)$. We then assign x to be such that the i^{th} bit of x, namely, $x_i = 1$ if $\Pi'_i = \Pi_i$ else $x_i = 0$ if $\Pi'_i \neq \Pi_i$. Output x.

The formal description of the sampling algorithm of $\mathsf{Dist}_{\lambda,C,\mathbf{i}}$ is given next.

Sampler of $\mathsf{Dist}_{\lambda,C,\mathbf{i}}$:

- Let $\phi(i) = (\Pi_i.\mathsf{Obf}, \Pi_i.\mathsf{Eval})$, for $i \in \{1, \ldots, \lambda\}$. Denote $\Pi_i = (\Pi_i.\mathsf{Obf}, \Pi_i.\mathsf{Eval})$.
- First compute the preprocessing step, $([\mathbf{C}]_1, \ldots, [\mathbf{C}]_n, aux) \leftarrow \mathsf{Preproc}(1^\lambda, 1^n, C)$. Here, $n = \lambda$. Maintain another copy of the set of candidates - for every $i \in [\lambda]$, set $\Pi'_i = \Pi_i$.
- For all $i \in [\lambda]$, execute $\Pi_i.\mathsf{Obf}(1^\lambda, [\mathbf{C}]_i)$ for at most $t = T\left(\lambda, \left|[\mathbf{C}]_i\right|\right)$ number of steps. For every $i \in [\lambda]$, if the computation of $\Pi_i.\mathsf{Obf}\left(1^\lambda, [\mathbf{C}]_i\right)$ does not abort within t number of time steps re-assign $\Pi_i.\mathsf{Obf} = I$ and $\Pi_i.\mathsf{Eval} = UTM$, where I is an identity TM and UTM is an universal TM.
- For all $i \in [\lambda]$, execute $\Pi_i.\mathsf{Obf}(1^\lambda, [\mathbf{C}]_i)$ for at most $t = T(\lambda, \left|[\mathbf{C}]_i\right|)$ number of steps. Denote $\overline{[\mathbf{C}]_i}$ to be the result of computation. Denote ℓ to be the input length of $[\mathbf{C}]_i$. For every $i \in [n]$, sample λ^3 points $x_{1,i}, \ldots, x_{\lambda^3,i} \xleftarrow{\$} \{0,1\}^\ell$. Check if the following condition holds:

$$\bigwedge_{j=1}^{\lambda^3} \left([\mathbf{C}]_i(x_{j,i}) = \Pi_i.\mathsf{Eval}\left(\overline{[\mathbf{C}]_i}, x_{j,i}\right) \right) = 1 \tag{2}$$

  If for any $i \in [\lambda]$ the above condition does not hold, re-assign $\Pi_i.\mathsf{Obf} = I$ and $\Pi_i.\mathsf{Eval} = UTM$.
- Construct a string $x \in \{0,1\}^\lambda$ such that the i^{th} bit x_i is generated as:

$$x_i = \begin{cases} 1, & \text{if} \quad \Pi_i = \Pi'_i \\ 0, & \text{otherwise} \end{cases}$$

- Output x.

Remark 5. *For every x in the support of $\mathsf{Dist}_{\lambda,C,\mathbf{i}}$ we have $x_{\mathbf{i}} = 1$ ($\mathbf{i}^{th}$ bit of x) since the $\mathbf{i}^{th}$ candidate is always correct.*

We prove the following useful sub-lemma. For every two circuits C_0, C_1 we claim that the outputs of the corresponding distributions $\mathsf{Dist}_{\lambda,C_0,\mathbf{i}}$ and $\mathsf{Dist}_{\lambda,C_1,\mathbf{i}}$ are computationally indistinguishable. Here, $\mathbf{i}$ corresponds to the candidate that is always correct.

SubLemma 1 (Candidate Indistinguishability Lemma)**.** *For large enough security parameter λ, any two circuits $C_0, C_1 \in \mathcal{C}_\lambda$, $\mathbf{i} \in [n]$ we have $\{x \xleftarrow{\$} \mathsf{Dist}_{\lambda,C_0,\mathbf{i}}\} \approx_c \{x \xleftarrow{\$} \mathsf{Dist}_{\lambda,C_1,\mathbf{i}}\}$, where $\mathbf{i}^{th}$ candidate (respresented by $\phi(\mathbf{i})$) is always correct, assuming that Π_{comb} satisfies decomposable security property.*

Proof. Suppose the above statement is false. Let $C_0, C_1 \in \mathcal{C}_\lambda$ be such that $\mathsf{Dist}_{\lambda,C_0,\mathbf{i}} \not\approx_c \mathsf{Dist}_{\lambda,C_1,\mathbf{i}}$, i.e., both the distributions are computationally indistinguishable. Denote the PPT distinguisher by $\mathcal{A}$. We use $\mathcal{A}$ to contradict the decomposable security of Π_{comb}. In more detail, we construct a reduction $\mathcal{B}$ that internally uses $\mathcal{A}$ to break the decomposable security of Π_{comb}. We give the description of the reduction $\mathcal{B}$ below.

$\mathcal{B}^{\mathcal{A}}\left(1^\lambda, \mathbf{i}, \{[\mathbf{C}]_j\}_{\substack{j \neq \mathbf{i} \\ j \in [\lambda]}}\right)$:

Execute bullets 1,3,4 of $\Pi_{\mathsf{univ}}.\mathsf{Obf}$. That is, execute the following steps:

- Let $\phi(i) = (\Pi_i.\mathsf{Obf}, \Pi_i.\mathsf{Eval})$, for $i \in \{1, \ldots, \lambda\}$. Denote $\Pi_i = (\Pi_i.\mathsf{Obf}, \Pi_i.\mathsf{Eval})$. Maintain another copy of the set of candidates - for every $i \in [\lambda]$, set $\Pi'_i = \Pi_i$.
- For all $i \in [\lambda]$ and $i \neq \mathbf{i}$, execute $\Pi_i.\mathsf{Obf}(1^\lambda, [\mathbf{C}]_i)$ for at most $t = T\left(\lambda, \left|[\mathbf{C}]_i\right|\right)$ number of steps. For every $i \in [\lambda]$, if the computation of $\Pi_i.\mathsf{Obf}\left(1^\lambda, [\mathbf{C}]_i\right)$ does not abort within t number of time steps re-assign $\Pi_i.\mathsf{Obf} = I$ and $\Pi_i.\mathsf{Eval} = UTM$, where I is an identity TM and UTM is an universal TM.
- For all $i \in [\lambda]$ and $i \neq \mathbf{i}$, execute $\Pi_i.\mathsf{Obf}(1^\lambda, [\mathbf{C}]_i)$ for at most $t = T(\lambda, \left|[\mathbf{C}]_i\right|)$ number of steps. Denote $\overline{[\mathbf{C}]_i}$ to be the result of computation. Denote ℓ to be the input length of $[\mathbf{C}]_i$. For every $i \in [n]$, sample λ^3 points $x_{1,i}, \ldots, x_{\lambda^3,i} \xleftarrow{\$} \{0,1\}^\ell$. Check if the following condition holds:

$$\bigwedge_{j=1}^{\lambda^3} \left([\mathbf{C}]_i(x_{j,i}) = \Pi_i.\mathsf{Eval}\left(\overline{[\mathbf{C}]_i}, x_{j,i}\right) \right) = 1 \tag{3}$$

  If for any $i \in [\lambda]$ the above condition does not hold, re-assign $\Pi_i.\mathsf{Obf} = I$ and $\Pi_i.\mathsf{Eval} = UTM$.
- Construct a string $x \in \{0,1\}^\lambda$ such that the i^{th} bit x_i is generated as:

$$x_i = \begin{cases} 1, & \text{if} \quad \Pi_i = \Pi'_i \\ 1, & \text{if} \quad \mathbf{i} = i \\ 0, & \text{otherwise} \end{cases}$$

- Execute $b \leftarrow \mathcal{A}(1^\lambda, x)$. Output b.

Suppose $\{[\mathbf{C}]_j\}_{j \neq \mathbf{i}, j \in [\lambda]}$ is derived from C_0 then the distribution of x in the end is identical to the distribution $\mathsf{Dist}_{\lambda, C_0, \mathbf{i}}$. Otherwise if $\{[\mathbf{C}]_j\}_{j \neq \mathbf{i}, j \in [\lambda]}$ is derived from C_1 then the distribution of x in the end is identical to $\mathsf{Dist}_{\lambda, C_1, \mathbf{i}}$. Thus, the distinguishability of $\mathsf{Dist}_{\lambda, C_0, \mathbf{i}}$ and $\mathsf{Dist}_{\lambda, C_1, \mathbf{i}}$ translates directly to the distinguishability of $\{[\mathbf{C}]_j\}_{j \neq \mathbf{i}, j \in [\lambda]}$ for the case of C_0 and C_1. This proves the sublemma. □

We now proceed to prove the main lemma. Recall that we are assured the existence of a secure IO candidate that is always correct. Let $\mathbf{i} \in \mathbb{Z}^{>0}$ be such that $\phi(\mathbf{i})$ represents the secure candidate. Let $\lambda \geq \mathbf{i}$. Consider two equivalent circuits $C_0, C_1 \in \mathcal{C}_\lambda$. That is, $|C_0| = |C_1|$ and for every $x \in \{0,1\}^\lambda$ we have $C_0(x) = C_1(x)$. Our goal is to show that $\Pi_{\mathsf{univ}}.\mathsf{Obf}(1^\lambda, C_0) \approx_c \Pi_{\mathsf{univ}}.\mathsf{Obf}(1^\lambda, C_1)$.

We define the following experiment. The following experiment, parameterized by (C_0, C_1), is same as $\Pi_{\mathsf{univ}}.\mathsf{Obf}(1^\lambda, C_0)$ except that the decision to choose which of the candidates to obfuscate the derived circuits $\{[\mathbf{C}]_i\}$ is made solely based on the circuit C_1.

$\underline{\mathsf{ExptObf}(1^\lambda, C_0, C_1, \mathbf{i})}$:

- Let $\phi(i) = (\Pi_i.\mathsf{Obf}, \Pi_i.\mathsf{Eval})$, for $i \in \{1, \ldots, \lambda\}$. Denote $\Pi_i = (\Pi_i.\mathsf{Obf}, \Pi_i.\mathsf{Eval})$.
- Compute the preprocessing step, $([\mathbf{C}]_1, \ldots, [\mathbf{C}]_n, aux) \leftarrow \mathsf{Preproc}(1^\lambda, 1^n, C_0)$ with $n = \lambda$.
- Sample x from $\mathsf{Dist}_{\lambda, C_1, \mathbf{i}}$, where $\mathbf{i} \in [\lambda]$. That is, x is sampled from the distribution Dist parameterized by $(\lambda, C_1, \mathbf{i})$.

- For every $i \in [\lambda]$ and $x_i = 0$, re-assign $\Pi_i.\mathsf{Obf} = I$ and $\Pi_i.\mathsf{Eval} = UTM$.
- Execute $\overline{[\mathbf{C}]_i} \leftarrow \Pi_i.\mathsf{Obf}(1^\lambda, [\mathbf{C}]_i)$ for at most $T(\lambda, |[\mathbf{C}]_i|)$ number of steps.
- Output $\overline{C} = (\overline{[\mathbf{C}]_1}, \ldots, \overline{[\mathbf{C}]_\lambda})$.

Consider the following claims.

Claim 3. *The distributions* $D_0 = \left\{\mathsf{ExptObf}\left(1^\lambda, C_b, C_b, \mathbf{i}\right)\right\}$ *and* $D_1 = \left\{\Pi_{\mathsf{univ}}.\mathsf{Obf}\left(1^\lambda, C_b\right)\right\}$ *are identical, for* $b \in \{0,1\}$.

The proof of the above claim follows directly from the description of $\mathsf{Dist}_{\lambda,C,\mathbf{i}}$.

Claim 4. *The distributions* $D_0 = \left\{\mathsf{ExptObf}\left(1^\lambda, C_b, C_0, \mathbf{i}\right)\right\}$ *and* $D_1 = \left\{\mathsf{ExptObf}(1^\lambda, C_b, C_1, \mathbf{i})\right\}$ *are computationally indistinguishable for* $b \in \{0,1\}$.

The proof of the above claim follows from the Candidate Indistinguishability Lemma (Lemma 1).

Claim 5. *The distributions* $D_0 = \left\{\mathsf{ExptObf}\left(1^\lambda, C_0, C_b, \mathbf{i}\right)\right\}$ *and* $D_1 = \left\{\mathsf{ExptObf}(1^\lambda, C_1, C_b, \mathbf{i})\right\}$ *are computationally indistinguishable for* $b \in \{0,1\}$.

Proof. We rely on the security (third bullet in Definition 6) of decomposable IO combiner to prove this claim. That is, the output of th IO combiner on two equivalent circuits are computationally indistinguishable. We only handle the case when $b = 0$, the other case symmetrically follows.

We denote $\mathcal{B}$ by the reduction that breaks the security of IO combiner. We denote the challenger of the IO combiner game to be Ch. $\mathcal{B}$ internally makes use of $\mathcal{A}$ where $\mathcal{A}$ is the PPT distinguisher that distinguishes the two distributions D_0 and D_1. $\mathcal{B}$ first computes $\phi(i) = (\Pi_i.\mathsf{Obf}, \Pi_i.\mathsf{Eval})$, for $i \in \{1, \ldots, \lambda\}$. Denote $\Pi_i = (\Pi_i.\mathsf{Obf}, \Pi_i.\mathsf{Eval})$. Then it samples x from $\mathsf{Dist}_{\lambda,C_0,\mathbf{i}}$, where $\mathbf{i} \in [\lambda]$. For every $i \in [\lambda]$ and $x_i = 0$, re-assign $\Pi_i.\mathsf{Obf} = I$ and $\Pi_i.\mathsf{Eval} = UTM$. It sends the candidates $\Pi_1, \ldots, \Pi_\lambda$ to Ch. It sends back $\overline{C} = (\overline{C_1}, \ldots, \overline{C_\lambda}, aux)$. Reduction $\mathcal{B}$ forwards $\overline{C}$ to $\mathcal{A}$.

If Ch uses C_0 to compute the obfuscated circuit then the input fed to $\mathcal{A}$ is distributed according to $\mathsf{ExptObf}\left(1^\lambda, C_0, C_0, \mathbf{i}\right)$. When Ch uses C_1, the input fed to $\mathcal{A}$ is distributed according to $\mathsf{ExptObf}\left(1^\lambda, C_1, C_0, \mathbf{i}\right)$. Furthermore, C_0 and C_1 are equivalent circuits. Thus, the computational indistinguishability of the obfuscations of C_0 and C_1 implies the computational indistinguishability of the two distributions. □

From Claims 3, 4, 5, it follows that $\Pi_{\mathsf{univ}}.\mathsf{Obf}(1^\lambda, C_0) \approx_c \Pi_{\mathsf{univ}}.\mathsf{Obf}(1^\lambda, C_1)$. In more detail,

$$\begin{aligned}
\Pi_{\mathsf{univ}}(1^\lambda, C_0) &\equiv \mathsf{ExptObf}\left(1^\lambda, C_0, C_0, \mathbf{i}\right) \text{ (from Claim 3)} \\
&\approx_c \mathsf{ExptObf}\left(1^\lambda, C_0, C_1, \mathbf{i}\right) \text{ (from Claim 4)} \\
&\approx_c \mathsf{ExptObf}\left(1^\lambda, C_1, C_1, \mathbf{i}\right) \text{ (from Claim 5)} \\
&\equiv \Pi_{\mathsf{univ}}(1^\lambda, C_1) \text{(from Claim 3)}
\end{aligned}$$

□

We have demonstrated that Π_{univ} satisfies both the correctness and security properties. This proves the theorem.

□

Step II: Approx. Correct to Exact (T, ϵ)-Universal Obfuscation. In Step I, we showed to construct a universal obfuscator that is $\left(1 - \frac{1}{\lambda}\right)$ correct. That is, for sufficiently large security parameter $\lambda \in \mathbb{N}$, every circuit $C \in \mathcal{C}_\lambda$, it holds that:

$$\Pr_{x \xleftarrow{\$} \{0,1\}^\lambda} \left[\Pi_{\mathsf{univ}}.C(x) = \mathsf{Eval}(\overline{C}, x) \ : \ \overline{C} \leftarrow \Pi_{\mathsf{univ}}.\mathsf{Obf}(1^\lambda, C) \right] \geq 1 - \frac{1}{\lambda}$$

We now apply the transformation of BV [BV16] to obtain a universal obfuscator that is exact (with overwhelming probability). In particular, we apply their transformation that is based on sub-exponential LWE assumption.

That is, for every $C \in \mathcal{C}_\lambda, x \in \{0,1\}^\lambda$, with high probability it holds that:

$$\Pr \left[\Pi_{\mathsf{univ}}.C(x) = \mathsf{Eval}(\overline{C}, x) \ : \ \overline{C} \leftarrow \Pi_{\mathsf{univ}}.\mathsf{Obf}(1^\lambda, C) \right] = 1$$

We state the formal theorem below.

Theorem 13. *Assuming learning with errors secure against adversaries running in time $2^{n^{\epsilon'}}$ and $(1 - 1/\lambda)$-correct (T, ϵ)-universal obfuscation, we have a (T, ϵ) -universal obfuscation that is exact (with overwhelming probability).*

Combining Step I and II $\Longrightarrow$ Main Result. Combining both the above steps and instantiating the decomposable IO combiner (Theorem 11) we get the following result:

Theorem 14. *Assuming LWE secure against adversaries running in time $2^{n^{\epsilon'}}$, there exists a (T, ϵ)-Universal Obfuscation with ϵ' being a function of ϵ.*

7 Witness Encryption Combiners

7.1 Definition of WE Combiner

We present the formal definition of a WE combiner below. The definition is similar to the definition of IO combiners. The task of the WE combiner is to take n candidates that are correct (in terms of encryption and decryption), and yield a scheme which is as secure as any one of the candidate schemes.

For a scheme Π we say that it is a *correct WE candidate* if it satisfies that correctness requirement of Definition 1 (item 1). We say that a candidate secure if it satisfies the security requirement (item 2) of Definition 1. We say that it is correct and secure if it satisfies both the requirements.

There are two PPT algorithms associated with an WE combiner, namely, $\mathsf{CombEnc}$ and $\mathsf{CombDec}$. Procedure $\mathsf{CombEnc}$ takes as input an instance x, a message m along with the description of multiple WE candidates and outputs a ciphertext. Procedure $\mathsf{CombDec}$ takes as input the ciphertext, a witness w, the description of the candidates and outputs the original message. Since the execution times of the candidates could potentially differ, we require the algorithms $\mathsf{CombEnc}$ and $\mathsf{CombDec}$ in addition to their usual inputs also take a time function T as input. T dictates an upper bound on the time required to execute all the candidates.

Syntax of WE Combiner. We define an WE combiner $\Pi_{\mathsf{comb}} = (\mathsf{CombEnc}, \mathsf{CombDec})$ for a language L.

- **Combiner of encryption algorithms,** $\mathsf{CT} \leftarrow \mathsf{CombEnc}(1^\lambda, x, m, \Pi_1, \ldots, \Pi_n, T)$: It takes as input security parameter λ, an instance x, a message m, description of WE candidates $\{\Pi_i\}_{i\in[n]}$, time function T and outputs a ciphertext.

- **Combiner of decryption algorithms,** $y \leftarrow \mathsf{CombDec}(\mathsf{CT}, w, \Pi_1, \ldots, \Pi_n, T)$: It takes as input a ciphertext CT, a witness for the instance x, descriptions of WE candidates $\{\Pi_i\}_{i\in[n]}$, time function T and outputs y.

We define the properties associated with a WE combiner scheme. There are two properties – correctness and security. We only consider the scenario where all the candidate WE schemes are (almost) perfectly correct but only one of them is secure.

Definition 9 (Secure WE combiner)**.** *Let $\Pi_1, \ldots, \Pi_n$ be n (almost) perfectly correct WE candidates for NP (that is all the schemes are correct, however all of them need not be secure). We say that $\Pi_{\mathsf{comb}} = (\mathsf{CombEnc}, \mathsf{CombDec})$ is a* **secure WE combiner** *if the following conditions are satisfied:*

- **Correctness.** *Consider the following process: (a) $\mathsf{CT} \leftarrow \mathsf{CombEnc}(1^\lambda, x, m, \Pi_1, \ldots, \Pi_n, T)$, (b) $y \leftarrow \mathsf{CombDec}(\mathsf{CT}, w, \Pi_1, \ldots, \Pi_n, T)$.*

  Then, $\mathsf{Pr}[y = m] \geq 1 - \mathsf{negl}(\lambda)$ over the randomness of $\mathsf{CombEnc}$.

- ***Security:** If for any $i \in [n]$ candidate Π_i is secure then, for any PPT adversary A and any polynomial $p(\cdot)$, there exists a negligible function $\mathsf{negl}(\cdot)$, such that for any $\lambda \in \mathbb{N}$, any $x \notin L$ and any two equal-length messages m_1 and m_2 such that $|x|, |m_1| \leq p(\lambda)$, we have that*

$$|\Pr[A(\mathsf{CombEnc}(1^\lambda, x, m_1, \Pi_1, \ldots, \Pi_n, T) = 1] - \Pr[A(\mathsf{CombEnc}(1^\lambda, x, m_2, \Pi_1, \ldots, \Pi_n, T)) = 1]| \leq \mathsf{negl}(\lambda).$$

Henceforth, we set the time function to be an a priori fixed polynomial. In our constructions presented next, we drop the parameter T which is input to the above algorithms.

7.2 Construction of WE Combiner

We give a construction of a WE combiner. Formally, we prove the following theorem.

Theorem 15. *If one-way functions exist, then there exists a secure WE combiner.*

The construction is given below. As described in Section 2.3, the main ingredient of the construction is a (perfectly) secure secret sharing scheme.

$\mathsf{CombEnc}(1^\lambda, x, m, \Pi_1, \ldots, \Pi_n)$**:** It takes as input security parameter λ, instance x, message m, description of candidates $\{\Pi_i = (\Pi_i.\mathsf{Enc}, \Pi_i.\mathsf{Dec})\}_{i\in[n]}$ and does the following.

1. **Secret share the message.** Choose n random strings $r_1, \ldots, r_n \in \{0,1\}^{|m|}$ such that $r_1 \oplus \ldots \oplus r_n = m$.

2. **Encrypt shares using candidates.** For $i \in [n]$, encrypt r_i using candidate Π_i: $y_i \leftarrow \Pi_i.\mathsf{Enc}(x, r_i)$.

3. Output $(y_1, \ldots, y_n)$.

$\underline{\mathsf{CombDec}(1^\lambda, \vec{y}, w, \Pi_1, \ldots, \Pi_n)}$**:** On input $\vec{y} = (y_1, \ldots, y_n)$, an input x with witness w, descriptions of candidates $\{\Pi_i\}_{i\in[n]}$ run the decryption candidates to obtain $r_i \leftarrow \Pi_i.\mathsf{Dec}(1^\lambda, y_i, w)$ for all $i \in [n]$. Compute $m \leftarrow r_1 \oplus \ldots \oplus r_n$ and output m.

Correctness: The correctness follows immediately from the scheme. For any $x \in L$ using the witness w we will get all r_i for $i \in [n]$ and from them we compute the correct message $m = r_1 \oplus \ldots \oplus r_n$.

Security: To prove security, assume that $x \notin L$ and let $i^* \in [n]$ be such that candidate Π_{i^*} is secure. Let m_0, m_1 be any two messages. Consider the following sequence of hybrids. Let H_0, parameterized by $(r_1, \ldots, r_n)$, be a distribution on the encryptions of m_0. That is, H_0 is a distribution over $(y_1, \ldots, y_n)$ where $y_i \leftarrow \Pi_i.\mathsf{Enc}(x, r_i)$ where r_i are random strings such that $r_1 \oplus \ldots \oplus r_n = m_0$. Then we define H_1, again parameterized by $(r_1, \ldots, r_n)$, to be a distribution on encryptions of the message $m_0 \oplus m_1 \oplus r_i$. That is, H_0 is a distribution over $y_{i^*} \leftarrow \Pi_{i^*}.\mathsf{Enc}(1^\lambda, x, r')$ where $r' = m_0 \oplus m_1 \oplus r_i$. From the security of Π_{i^*} we have that $\mathsf{H}_0 \approx \mathsf{H}_1$. Notice that

$$r_1 \oplus \ldots \oplus r_{i^*-1} \oplus r' \oplus r_{i^*+1} \ldots \oplus r_n = m_0 \oplus m_1 \oplus m_0 = m_1.$$

Moreover, the distribution of $r_1, \ldots, r_{i^*-1}, r', r_{i^*+1}, \ldots, r_n$ and the distribution $r_1, \ldots, r_n$ such that $r_1 \oplus \ldots \oplus r_n = m_1$ are identical. Therefore, if we define H_2 to be the distribution on the honest encryptions of the message m_1 (i.e., performed according to the scheme), we get that $\mathsf{H}_1 \equiv \mathsf{H}_2$. Thus we have that $\mathsf{H}_0 \approx \mathsf{H}_2$ which proves the security of the above scheme.

8 Universal Witness Encryption

We introduce the notion of universal witness encryption. We define a pair of Turing machines $\Pi_{\mathsf{univ}}.\mathsf{Enc}$ and $\Pi_{\mathsf{univ}}.\mathsf{Dec}$ to be a universal witness encryption if the existence of a secure WE candidate implies that $(\Pi_{\mathsf{univ}}.\mathsf{Enc}, \Pi_{\mathsf{univ}}.\mathsf{Dec})$ is also a secure WE candidate. Constructing a universal WE scheme means that we can turn the mere existence of a secure WE candidate (along with the existence of one-way functions) into an explicit construction. Formally:

Definition 10 (T-Universal Witness Encryption)**.** *We say that a pair of Turing machines* $\Pi_{\mathsf{univ}} = (\Pi_{\mathsf{univ}}.\mathsf{Enc}, \Pi_{\mathsf{univ}}.\mathsf{Dec})$ *is a* **universal witness encryption***, parametrized by T, if there exists an secure witness encryption scheme for* NP *with time function T then* Π_{univ} *is a witness encryption for* NP *with time function* $\mathrm{poly}(T)$*.*

We prove the following.

Theorem 16. *If one-way functions exist then there exists a T-universal witness encryption scheme.*

8.1 Construction of universal WE

The intuition behind the scheme presented below is described in the introduction. We now give the construction with full details. First, we give a witness-correctness amplifier for WE. That is, we show how to transform a WE where the decryption can be performed using only one witness into another WE scheme where the decryption can be performed using *any* valid witness. The next step is a *message-correctness amplification* step. We show how to transform a WE scheme that is correct only on a subset of messages (to be encrypted) into another WE scheme that is correct on *every* message with overwhelming probability. Once we have both the above steps, we then present the universal construction.

Step I: Witness-Correctness Amplifier for WE. Let L be an NP language associated with a relation R. First we define a transformation that converts any approximate WE candidate that might work differently on different witnesses, to a *witness-worst-case* scheme $\mathsf{w}\Pi$: a scheme in which all witnesses decrypt correctly with the same probability. We start with a definition of a witness encryption scheme that works on a single witness for any instance in the language.

Definition 11. *We say that Π is a witness encryption scheme with single witness correctness if we replace the (almost) perfect correctness requirement with the following one:*

- ***Single Witness Correctness:*** *For any security parameter λ, any $m \in \{0,1\}^*$, any $x \in L$ there exists $w \in \{0,1\}^*$ such that $(x,w) \in R$ and the following condition holds:*

$$\Pr[\Pi.\mathsf{Dec}(\Pi.\mathsf{Enc}(1^\lambda, x, m), w) = m] \geq 1 - 2^{-\lambda}.$$

Using this definition we can prove the following theorem.

Theorem 17. *Let Π be a WE scheme with single witness correctness, and assume one-way functions exist. Then, there exist a (secure) WE scheme, $\mathsf{w}\Pi$, with (almost) perfect correctness.*

Proof of Theorem 17. Suppose L be a language in NP. We augment L and define L_z with the relation R_z such that

$$(x,w) \in R_z \iff (x,w) \in R \vee \mathsf{PRG}(w) = z.$$

Let $\mathsf{PZK} = (\mathsf{PZK.Pre}, \mathsf{PZK.Verify}, \mathsf{PZK.Prove})$ be a zero-knowledge proof system with pre-processing (see Definition 2). Now given an instance x, let $(\sigma_V, \sigma_P) \leftarrow \mathsf{PZK.Pre}(1^\lambda)$ be the state of the verifier and prover respectively during the protocol up to the last round. We define a new language L_{z,σ_V} with a relation R_{z,σ_V} such that for witness π we have:

$$(x,\pi) \in R_{z,\sigma_V} \iff \mathsf{PZK.Verify}_{L_z}(\sigma_V, x, \pi) = 1.$$

The full details of the transformation in Figure 12.

Claim 6. *For large enough λ, for $z \leftarrow \{0,1\}^{2\lambda}$ chosen at random and for $(\sigma_V, \sigma_P) \leftarrow \mathsf{PZK.Pre}(1^\lambda)$ with all but negligible probability it holds that $L \equiv L_{z,\sigma_V}$.*

Proof. With extremely high probability we have that z is not in the image of the PRG, and hence $L = L_z$. From the correctness of the PZK we have that if $x \in L$ then there exists a proof π such $\mathsf{PZK.Verify}(\sigma_V, x, \pi) = 1$ and thus $x \in L_{z,\sigma_V}$. If $x \notin L$ then by the perfect soundness property of the PZK we get that there does not exist a string π such that $\mathsf{PZK.Verify}(\sigma_V, x, \pi) = 1$ and thus $x \notin L_{z,\sigma_V}$. □

The encryption and decryption algorithms of $\mathsf{w}\Pi$:

$\underline{\mathsf{w}\Pi.\mathsf{Enc}_L(1^\lambda, x, m)}$:

1. sample random $z \xleftarrow{\$} \{0,1\}^{2\lambda}$ and $(\sigma_V, \sigma_P) \leftarrow \mathsf{PZK}_{L_z}.\mathsf{Pre}(1^\lambda)$.
2. compute $\mathsf{CT} = \Pi.\mathsf{Enc}_{L_{z,\sigma_V}}(1^\lambda, x, m)$.
3. output $(z, \sigma_P, \mathsf{CT})$.

$\underline{\mathsf{w}\Pi.\mathsf{Dec}_L(1^\lambda, \mathsf{CT}', x, w)}$:

1. parse CT' as $(z, \sigma_P, \mathsf{CT})$.
2. compute $\pi = \mathsf{PZK}_{L_z}.\mathsf{Prove}(\sigma_P, x, w)$.
3. output $\Pi.\mathsf{Dec}_{L_{z,\sigma_V}}(\mathsf{CT}, \pi)$.

Figure 12: The worst-case scheme $\mathsf{w}\Pi$, parameterized by Π.

Claim 7. *Let $L \in \mathsf{NP}$ be a language with verifier V. Let Π be a candidate witness encryption scheme and let $\mathsf{w}\Pi$ be the witness-worst-case construction defined in Figure 12. Then for large enough λ, a message $m \in \{0,1\}^*$ the scheme $\mathsf{w}\Pi$ satisfies the following conditions:*

1. *For any $x \in L$ and w such that $(x, w) \in R$:*

$$\Pr[\mathsf{w}\Pi.\mathsf{Dec}(\mathsf{w}\Pi.\mathsf{Enc}(1^\lambda, x, m), w) = m] \geq \Pr[\Pi.\mathsf{Dec}(\Pi.\mathsf{Enc}(1^\lambda, x, m), w) = m] - \mathsf{negl}(\lambda).$$

2. *Worst-Case witness correctness: for any λ, $m \in \{0,1\}^*$, $x \in L$ and any w, w' such that $V(x, w) = 1$ and $V(x, w') = 1$ it holds that:*

$$|\Pr[\mathsf{w}\Pi.\mathsf{Dec}(\mathsf{w}\Pi.\mathsf{Enc}(1^\lambda, x, m), w) = m] - \Pr[\mathsf{w}\Pi.\mathsf{Dec}(\mathsf{w}\Pi.\mathsf{Enc}(1^\lambda, x, m), w') = m]| \geq \mathsf{negl}(\lambda).$$

3. *Security: for any λ, $m \in \{0,1\}^*$, $x \notin L$ it holds that if*

$$\Pi.\mathsf{Enc}(1^\lambda, x, m) \approx_c \Pi.\mathsf{Enc}(1^\lambda, x, 0^{|m|}),$$

then

$$\mathsf{w}\Pi.\mathsf{Enc}(1^\lambda, x, m) \approx_c \mathsf{w}\Pi.\mathsf{Enc}(1^\lambda, x, 0^{|m|})$$

(with a possible loss of $\mathsf{negl}(\lambda)$ in the indistinguishability probability).

Proof.

1. From Claim 6 we know that with high probability $L \equiv L_{z,\sigma_V}$. From the correctness of the PZK we get that for $\pi \leftarrow \mathsf{PZK}.\mathsf{Prove}(\sigma_p, x, w)$ it holds that $\mathsf{PZK}.\mathsf{Verify}_{L_z}(\sigma_V, x, \pi) = 1$, and thus if $\Pi.\mathsf{Dec}(\Pi.\mathsf{Enc}(1^\lambda, x, m), w) = m$ then also $\mathsf{w}\Pi.\mathsf{Dec}(\mathsf{w}\Pi.\mathsf{Enc}(1^\lambda, x, m), w) = m$.

2. Let w and w' be such that $(x,w) \in R$ and $(x,w') \in R$. From the zero-knowledge property of PZK, we have that:

$$\Pr\left[\mathsf{wII.Dec}(\mathsf{wII.Enc}(1^\lambda, x, m), w) = m\right] =$$

$$\Pr\left[\mathsf{WE.Dec}_{L_{z,\sigma_V}}(\mathsf{CT}, \pi) = m \;\middle|\; \begin{array}{l} z \leftarrow \{0,1\}^{2\lambda}, \\ \sigma_V, \sigma_P \leftarrow \mathsf{PZK.Pre}(1^\lambda), \\ \mathsf{CT} \leftarrow \mathsf{WE.Enc}_{L_{z,\sigma_V}}(x, m), \\ \pi \leftarrow \mathsf{PZK.Prove}_{L_z}(\sigma_P, x, w) \end{array}\right] =$$

$$\Pr\left[\mathsf{WE.Dec}_{L_{z,\sigma_V}}(\mathsf{CT}, \pi') = m \;\middle|\; \begin{array}{l} z \leftarrow \{0,1\}^{2\lambda}, \\ \sigma_V, \sigma_P \leftarrow \mathsf{PZK.Pre}(1^\lambda), \\ \mathsf{CT} \leftarrow \mathsf{WE.Enc}_{L_{z,\sigma_V}}(x, m), \\ \pi' \leftarrow \mathsf{PZK.Prove}_{L_z}(\sigma_P, x, w') \end{array}\right] \pm \mathsf{negl}(\lambda) =$$

$$\Pr\left[\mathsf{wII.Dec}(\mathsf{wII.Enc}(1^\lambda, x, m), w') = m\right] \pm \mathsf{negl}(\lambda).$$

3. From Claim 6 we have that $x \notin L_{z,\sigma_V}$ and thus by the security of WE the item follows.

□

The theorem is a direct corollary of the three items of Claim 7. The security follows from item 3. The correctness of the single witness follows from item 1. The correctness for any other witness follows from item 2. □

Step II: Message-Correctness Amplifier for WE. We describe an amplification transformation for messages. That is, we start with a WE scheme where the decryption errs on a subset of messages. We then transform it into one scheme where the decryption is correct on all the messages with overwhelming probability.

The transformation proceeds in two main steps: (1) a worst-case to average case message reduction and (2) amplification of the error probability. A similar amplification for IO is described in [KMN+14, Appendix B] (without the message worst-case reduction).

Definition 12. *We say that* Π *is a witness encryption scheme with approximate correctness if we replace the (almost) perfect correctness requirement with the following one:*

- ***Approximate Correctness:*** *There exists a constant* $\epsilon > 0$ *such that for any security parameter* λ*, any* $(x,w) \in R$ *it holds that*

$$\Pr_{m \in \{0,1\}^\lambda}[\Pi.\mathsf{Dec}(\Pi.\mathsf{Enc}(1^\lambda, x, m), w) = m] \geq 1/2 + \epsilon.$$

where the probability is over a random message m *and the internal randomness of* Π*.*

Let Π be a WE scheme. Let L be a language, x an instance, and w a message. Define a new scheme $\mathsf{ampWE}(\Pi) = (\mathsf{ampWE.Enc}, \mathsf{ampWE.Dec})$ as follow. Let n be a parameter that will be determined later.

Encryption: The encryption algorithm given a security parameter λ, an instance x and a message m:

1. For $i \in [n]$, sample $r_i \in \{0,1\}^{|m|}$ uniformly at random.
2. For $i \in [n]$, compute $c_i \leftarrow \Pi_L.\mathsf{Enc}(1^\lambda, x, r_i)$ (each with fresh randomness).
3. For $i \in [n]$, compute $r'_i \leftarrow m \oplus r_i$.
4. Output $(c_1, r'_1), \ldots, (c_n, r'_n)$.

Decryption: The decryption algorithm given n pairs $(c_1, r'_1), \ldots, (c_n, r'_n)$, and a witness w:

1. For all $i \in [n]$ compute $y_i \leftarrow \Pi.\mathsf{Dec}(1^\lambda, c_i) \oplus r'_i$.
2. Output $y \leftarrow \mathsf{majority}(y_1, \ldots, y_n)$.

Lemma 32. *Assume that Π is an approximate witness encryption scheme, then $\Pi' = \mathsf{ampWE}(\Pi)$ is an (almost) exact witness encryption scheme.*

Proof. Set $n = 4\epsilon^2\lambda$. Since Π is approximately correct, and since each r_i is completely random, we have that for all $i \in [n]$ it holds that

$$\Pr[\Pi.\mathsf{Dec}(\Pi.\mathsf{Enc}(1^\lambda, x, r_i), w) = r_i] \geq 1/2 + \epsilon.$$

Thus we get that $\Pr[y_i = m] \geq 1/2 + \epsilon$. By a standard Chernoff bound we get that the probability that more than half are decrypted incorrectly is:

$$\Pr[\mathsf{majority}(y_1, \ldots, y_n) \neq m] \leq \exp\left(-\frac{\epsilon^2}{2} \cdot \frac{n}{2}\right) \leq 2^{-\lambda}.$$

Security follows from a standard hybrid argument (see [KMN+14, Appendix B]). □

Given the witness-worst-case transformation and the amplification transformation we are ready to present the universal construction.

Universal construction of WE. The final construction is presented below. Let $\Pi_1, \ldots, \Pi_n$ be n candidate schemes, and let T be the time bound. The universal scheme works as follows.

$\underline{\Pi_{\mathsf{univ}}.\mathsf{Enc}(1^\lambda, x, m)}$: Takes as input security parameter λ, input x and message and executes the following steps:

1. Let $\phi(i) = (\Pi_i.\mathsf{Enc}, \Pi_i.\mathsf{Dec})$, for $i \in \{1, \ldots, n\}$ where $n = \lambda$. Denote $\Pi_i = (\Pi_i.\mathsf{Enc}, \Pi_i.\mathsf{Dec})$.
2. **Witness-worst-case transformation.** For all $i \in [\lambda]$, re-assign Π_i with the worst case transformation $\Pi_i \leftarrow \mathsf{wc}\Pi_i$ (see Figure 12).
3. **Eliminating Candidates with Large Runtimes.** For all $i \in [\lambda]$, execute $\Pi_i.\mathsf{Enc}(1^\lambda, x, r_i)$ where $r_i \xleftarrow{\$} \{0,1\}^{|m|}$ and for at most $t = T(\lambda, x)$ number of steps. For every $i \in [n]$, if the computation does not abort within t number of time steps re-assign $\Pi_i = I$, where I is an identity scheme.

4. **Eliminating Candidates with Imperfect Correctness.** For all $i \in [\lambda]$ do the following test λ times:
   - sample $r_i \xleftarrow{\$} \{0,1\}^{|m|}$.
   - sample $s \xleftarrow{\$} \{0,1\}^{\lambda}$, and compute $z \leftarrow \mathsf{PRG}(s)$.
   - compute $(\sigma_V, \sigma_P) \leftarrow \mathsf{PZK.Pre}(1^{\lambda})$.
   - compute $r'_i \leftarrow \Pi_{L_{z,\sigma_V}}.\mathsf{Dec}(1^{\lambda}, \Pi_{L_{z,\sigma_V}}.\mathsf{Enc}_i(1^{\lambda}, x, r_i), s)$.
   - if $r'_i \neq r_i$ then re-assign $\Pi_i = I$.

5. **Amplifying Correctness.** For all $i \in [\lambda]$, re-assign Π_i to be $\mathsf{ampWE}(\Pi_i, T)$, where ampWE is the correctness amplifier for WE (see Lemma 32).

6. **Combine candidates.** Use the combiner constructed in Section 7.2 on the candidate $\Pi_1, \ldots, \Pi_n$ to get a scheme $\Pi_{\mathsf{comb}} = (\Pi_{\mathsf{comb}}.\mathsf{CombEnc}, \Pi_{\mathsf{comb}}.\mathsf{CombDec})$, and output $\mathsf{CT} \leftarrow \Pi_{\mathsf{comb}}.\mathsf{CombEnc}(1^{\lambda}, x, m, \Pi_1, \ldots, \Pi_n)$.

$\underline{\Pi_{\mathsf{univ}}.\mathsf{Dec}(\mathsf{CT}, w)}$: Takes as a ciphertext CT, a witness w for x and executes:

1. Output $m \leftarrow \Pi_{\mathsf{comb}}.\mathsf{CombDec}(\mathsf{CT}, w, \Pi_1, \ldots, \Pi_n)$.

Proof of Theorem 16. We prove correctness and security of the universal scheme Π_{univ}.
Correctness: To prove correctness it suffices to show correctness for all candidates. That is, for all $i \in [n]$ we show that:

$$\Pr_r[\Pi_i.\mathsf{Dec}(\Pi_i.\mathsf{Enc}(x, r), w)] \geq 1 - \mathsf{negl}(\lambda).$$

Then, using a union bound we would get correctness for the entire scheme. Consider some candidate $\Pi = \Pi_i$ for $i \in [n]$ and denote by Π^j the candidate after step j. If at any point Π is re-assigned with the identity scheme then it will have perfect correctness. We consider the scheme after each step. For any witness w we denote the probability that it decrypts correctly by α_w. That is,

$$\alpha_w = \Pr_r[\Pi.\mathsf{Dec}(\Pi.\mathsf{Enc}(1^{\lambda}, x, r_i), w) = r].$$

After step (2) we know that this probability is roughly the same for any witness. Namely, using claim 7 item 2, we get that for any witness w' it holds that $\alpha_w \approx \alpha_{w'}$ (i.e., $\alpha_w = \alpha_{w'} \pm \mathsf{negl}(\lambda)$).

In step 4, we run the scheme on a different relation, namely we choose $z \leftarrow \mathsf{PRG}(s)$ instead of a random z. Denote by α'_w the same probability only under this new relation. From the security of the PRG we know that $\alpha_w \approx \alpha'_w$. Thus, we get that $\alpha_w \approx \alpha'_s$. We verify that α'_s is high, and otherwise discard the scheme and re-assign it with the identity. Assume that $\alpha'_s < 3/4$, then the probability that all checks verify is at most $(3/4)^{-\lambda} = \mathsf{negl}(\lambda)$. Thus, after step 4, we know that with all but negligible probability, we have that $\alpha'_s \geq 3/4$, and thus $\alpha_w \geq 3/4 - \mathsf{negl}(\lambda)$. Step 5 is an amplification step. Since $\alpha_w \geq 3/4 - \mathsf{negl}(\lambda)$ we get that after the amplification step $\alpha_w \geq 1 - \mathsf{negl}(\lambda)$. Finally, from the worst-case reduction of the amplification transformation we know that if α_w is high for a random message, then the corresponding probability is also high for the specific message m. Thus, correct holds for the candidate.

Security: Let $x \notin L$. The security should follow from the properties of the secure combiner performed at the last step. However, to apply it we need to prove that it is applicable here. That is we need to prove that there exists at least one secure candidate, and that the distribution on candidates given to the combiner is independent of the message.

First, we claim that there is an $i \in [n]$ such that Π_i is a correct and secure. Let $i \in [n]$ be the secure candidate at step 1. We need to show that the resulting candidate Π_i is correct and secure. The transformation any candidates goes though are the witness-worst-case transformation and the potential re-assignment to the identity scheme. From Claim 7 items 1 and 3 we get that the worst-case transformation remains its correctness and security.

Thus, we need to show that this candidate was not re-assigned to the identity at any step. It will not be re-assigned in step 3 since assume a bound on its running time. Since Π_i has (almost) prefect correctness, we get that after in step 4, the probability that any of the steps fails in negligibly small. Therefore, at not step will Π_i be re-assigned with the identity.

In steps 1 to 5, we change the candidates. We need to prove that the distribution of the resulting candidates given to the combiner is independent of the message. Notice that the choice of candidates is performed independently of the message, using random messages instead. The message is used only for the last step and thus we can apply the security of the combiner to get the overall security of Π_{univ}. □

9 Universal Schemes for Other Primitives

Universal Secret-Sharing We have constructed a combiner and a universal scheme for witness encryption, under the assumption of one-way functions. In [KNY14], Komargodski et al. have proved that witness encryption for NP along with one-way functions implies that existence of secret-sharing for NP. Intuitively, a secret-sharing scheme for an access structure in NP is defined as following: for the "qualified" subsets there is a witness attesting to this fact and given the witness it should be possible to reconstruct the secret. On the other hand, for the "unqualified" subsets there is no witness, and so it should not be possible to reconstruct the secret.

It is easy to see that, unconditionally, secret-sharing for NP implies witness encryption for NP. Combining these results we get a combiner and a universal scheme for secret sharing for NP. In particularly, given n candidate schemes for secret-sharing, where only one is correct and secure, we transform each one to a witness encryption scheme. We know that the correct and secure scheme will yield a secure witness encryption scheme. Then, applying the combiner for WE we get a correct and secure WE scheme. Finally, applying the construction of [KNY14] to this scheme we get a correct and secure secret-sharing scheme for NP.

Universal Witness PRF As an additional corollary for the universal WE scheme, we get a combiner and universal construction for witness PRF [Zha16]. Informally, a witness PRF for an NP language L is a PRF F such that anyone with a valid witness that $x \in L$ can compute $F(x)$ without the secret key, but for all $x \notin L$, $F(x)$ is computationally hidden without knowledge of the secret key.

The proof of a combiner and universal witness PRF is very similar to the proof of a universal WE scheme. The only modification needed is in combing the security of n witness PRF candidate. In the WE combiner, we secret-shared the message to the n candidate. For witness PRF, we want to combiner output to be pseudorandom, as long as one candidate is outputs a pseudorandom string.

Thus, we simply XOR all the outputs of the n candidates. It is easy to see that the final output will be pseudorandom. For boosting correctness, the same techniques of WE apply for witness as well. Notice, that since witness PRF by itself wimples the existence of one-way function, we get a universal witness PRF scheme with no additional assumptions.

Acknowledgements

We thank Yuval Ishai for helpful discussions and for bringing to our notice the problem of universal obfuscation. We additionally thank Abhishek Jain for useful discussions.

References

[AB15] Benny Applebaum and Zvika Brakerski. Obfuscating circuits via composite-order graded encoding. In *Theory of Cryptography*, pages 528–556. Springer, 2015.

[ACG+14] Prabhanjan Ananth, Nishanth Chandran, Vipul Goyal, Bhavana Kanukurthi, and Rafail Ostrovsky. Achieving privacy in verifiable computation with multiple servers–without fhe and without pre-processing. In *PKC*. 2014.

[AGIS14] Prabhanjan Ananth, Divya Gupta, Yuval Ishai, and Amit Sahai. Optimizing obfuscation: Avoiding barrington's theorem. In *ACM CCS*, 2014.

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In *CRYPTO*. 2015.

[AJS15] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Indistinguishability obfuscation from functional encryption for simple functions. Technical report, Cryptology ePrint Archive, Report 2015/730, 2015.

[App13] Benny Applebaum. Bootstrapping obfuscators via fast pseudorandom functions. *IACR Cryptology ePrint Archive*, 2013:699, 2013. To appear in Asiacrypt 2014.

[BBS98] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In *EUROCRYPT*, 1998.

[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*, 2001.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *PKC*. 2014.

[BGK+14] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In *EUROCRYPT*, 2014.

[BGL+15] Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Siddartha Telang. Succinct randomized encodings and their applications. In *STOC*, 2015.

[BMSZ16] Saikrishna Badrinarayanan, Eric Miles, Amit Sahai, and Mark Zhandry. Post-zeroizing obfuscation: The case of evasive circuits. In *EUROCRYPT*, 2016.

[BP15] Nir Bitansky and Omer Paneth. Zaps and non-interactive witness indistinguishability from indistinguishability obfuscation. In *TCC*, pages 401–427. Springer, 2015.

[BR14] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In *TCC*, 2014.

[BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In *FOCS*, 2015.

[BV16] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation: From approximate to exact. In *TCC*, 2016.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT*. 2013.

[CA81] G.R. Blakley C.A. Asmuth. An efficient algorithm for constructing a cryptosystem which is harder to break than two other cryptosystems. *Computers and Mathematics with Applications*, 1981.

[CDPR15] Ronald Cramer, Léo Ducas, Chris Peikert, and Oded Regev. Recovering short generators of principal ideals in cyclotomic rings. *IACR Cryptology ePrint Archive*, 2015:313, 2015.

[CGH+15] Jean-Sébastien Coron, Craig Gentry, Shai Halevi, Tancrede Lepoint, Hemanta K Maji, Eric Miles, Mariana Raykova, Amit Sahai, and Mehdi Tibouchi. Zeroizing without low-level zeroes: New mmap attacks and their limitations. In *Advances in Cryptology–CRYPTO 2015*, pages 247–266. Springer, 2015.

[CHL+15] Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehlé. Cryptanalysis of the multilinear map over the integers. In *EUROCRYPT*, 2015.

[CJL16] Jung Hee Cheon, Jinhyuck Jeong, and Changmin Lee. An algorithm for cspr problems and cryptanalysis of the ggh multilinear map without an encoding of zero. Technical report, Cryptology ePrint Archive, Report 2016/139, 2016.

[CLLT15] Jean-Sebastien Coron, Moon Sung Lee, Tancrede Lepoint, and Mehdi Tibouchi. Cryptanalysis of ggh15 multilinear maps. Cryptology ePrint Archive, Report 2015/1037, 2015. http://eprint.iacr.org/.

[CLT13] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In *CRYPTO*, 2013.

[CLTV15] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic circuits and applications. In *TCC*. 2015.

[CM15] Michael Clear and Ciaran McGoldrick. Multi-identity and multi-key leveled FHE from learning with errors. In *CRYPTO*, 2015.

[DMP88] Alfredo De-Santis, Silvio Micali, and Giuseppe Persiano. Non-interactive zero-knowledge with preprocessing. In *CRYPTO*, 1988.

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

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2013.

[GGH+13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013.

[GGH15] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In *TCC*. 2015.

[GGHZ16] Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Fully secure functional encryption without obfuscation. 2016.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM (JACM)*, 1986.

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

[GHV10] Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. i-hop homomorphic encryption and rerandomizable yao circuits. In *CRYPTO*. 2010.

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

[GK16] Shafi Goldwasser and Yael Tauman Kalai. Cryptographic assumptions: A position paper. In *TCC*, 2016.

[GLSW15] Craig Gentry, Allison B. Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. In *FOCS*, 2015.

[GLW14] Craig Gentry, Allison B. Lewko, and Brent Waters. Witness encryption from instance independent assumptions. In *CRYPTO*, 2014.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *STOC*, 1987.

[Göd31] Kurt Gödel. Über formal unentscheidbare sätze der principia mathematica und verwandter systeme i. *Monatshefte für mathematik und physik*, 1931.

[Gol01] Oded Goldreich. *The Foundations of Cryptography - Volume 1, Basic Techniques*. Cambridge University Press, 2001.

[GR07] Shafi Goldwasser and Guy N Rothblum. On best-possible obfuscation. In *TCC*. 2007.

[Her05] Amir Herzberg. On tolerant cryptographic constructions. In *CT-RSA*, 2005.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 1999.

[HJ15] Yupu Hu and Huiwen Jia. Cryptanalysis of ggh map. *IACR Cryptology ePrint Archive*, 2015:301, 2015.

[HKN+05] Danny Harnik, Joe Kilian, Moni Naor, Omer Reingold, and Alon Rosen. On robust combiners for oblivious transfer and other primitives. In *EUROCRYPT*, 2005.

[KMN+14] Ilan Komargodski, Tal Moran, Moni Naor, Rafael Pass, Alon Rosen, and Eylon Yogev. One-way functions and (im)perfect obfuscation. In *FOCS*, 2014.

[KNY14] Ilan Komargodski, Moni Naor, and Eylon Yogev. Secret-sharing for NP. In *ASIACRYPT*, 2014.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *ACM SIGSAC*, 2013.

[Lev87] Leonid A. Levin. One-way functions and pseudorandom generators. *Combinatorica*, 1987.

[LS90] Dror Lapidot and Adi Shamir. Publicly verifiable non-interactive zero-knowledge proofs. In *CRYPTO*, 1990.

[LTV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *STOC*, 2012.

[MSW14] Eric Miles, Amit Sahai, and Mor Weiss. Protecting obfuscation against arithmetic attacks. *IACR Cryptology ePrint Archive*, 2014:878, 2014.

[MSZ16] Eric Miles, Amit Sahai, and Mark Zhandry. Annihilation attacks for multilinear maps: Cryptanalysis of indistinguishability obfuscation over ggh13. Cryptology ePrint Archive, Report 2016/147, 2016. http://eprint.iacr.org/.

[MW16] Pratyay Mukherjee and Daniel Wichs. Two round MPC from LWE via multi-key FHE. In *EUROCRYPT*. 2016.

[Nao91] Moni Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 1991.

[PST14] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In *CRYPTO*, 2014.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *STOC*, 2014.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, 1986.

[Zha16] Mark Zhandry. How to avoid obfuscation using witness prfs. In *TCC*, 2016.

[Zim15] Joe Zimmerman. How to obfuscate programs directly. In *EUROCRYPT*, 2015.

A Status of IO Schemes

Since the first candidate construction of indistinguishability obfuscation [GGH+13b] based on multilinear maps (mmaps) was proposed, many mmap-based iO constructions that have followed suit. Recently, Cheon et al. [CHL+15] and many followup works have demonstrated that existing mmap schemes suffer from vulnerabilities. It is important to note that most these works have had no direct impact on the security of the iO candidates. Specifically:

- Most works on the cryptanalysis of mmaps have no impact on iO candidates because they require encodings (such as low-level encodings of zero) that do not arise in iO candidates [CHL+15, HJ15, CLLT15] or they only impact the parameter choices of mmap schemes [CJL16]. We note that the attacks of [CJL16] on GGH13 mmaps, for example, can be avoided by setting $n = \lambda \log^2 q$, or by choosing n to not be a power of 2.
- Some attacks yield only subexponential or quantum attacks [CDPR15].
- On the other hand, two recent works have shown the first polynomial-time attacks on some iO candidates: First, the work of [CGH+15] extended the cryptanalysis of [CHL+15] to apply to some iO candidates over CLT13 mmaps. Second, the work of [MSZ16] introduced a new class of polynomial-time attacks called Annihilating Attacks that applied to some iO candidates over GGH13 mmaps. However, as we illustrate in Figure 13, these known cryptanalytic techniques still leave many iO candidates without any known attacks.

  We elaborate further on the current state of affairs below. We expect more attacks to be found and the security situation of iO candidates to grow more complex as our understanding grows. This situation emphasizes the need for iO combiners, the subject of our work.

It is important to note that the security of these candidates (in most cases) depends on the class of functions being obfuscated. We classify the class of functions being obfuscated into many categories as described below.

- **Branching Programs (BP)**: This corresponds to the class of all poly-sized matrix branching programs. In this case, a matrix branching program is being directly obfuscated.

- NC^1: This corresponds to the class of NC^1 circuits. For obfuscators that natively obfuscate matrix branching programs, this corresponds to obfuscating the specific class of matrix branching programs obtained by applying Barrington's theorem to the class of NC^1 circuits.

- **General Circuits**: This category corresponds to general polynomial-size circuits. However, no iO candidates currently exist that "directly" obfuscate general circuits. Instead, the obfuscation of general circuits is achieved by obfuscating specific classes of "bootstrappable" functions: obfuscating such a bootstrappable function family suffices to obfuscate arbitrary class of polynomial-time functions. This is enabled by bootstrapping theorems. That is, in order to obfuscate a circuit C, we first obfuscate a circuit $C^* = g(C)$, a circuit derived from C. Then we apply bootstrapping theorem on C^* to obtain an obfuscation of C. We enumerate the different classes of bootstrappable functions known:

  - $\mathcal{F}_{GGH+}$: This corresponds to the class of functions defined in [GGH+13b]. It essentially consists of circuits that first perform proof checking and then compute the decryption of the input FHE ciphertext.

– $\mathcal{F}_{App}$: This class of functions, defined in[App13], corresponds to the class of functions that consists of computing randomized encoding of the functions to be obfuscated.

– $\mathcal{F}_{BGL+}$: This class of functions, defined in[BGL+15] (also observed by [AJS15]), is similar in spirit to [App13] except that the functions compute the randomized encoding of the functions "one piece at a time". The advantage of this, in comparison to [App13], the size of the circuits implementing these functions is independent of the size of the original circuit to be obfuscated.

– $\mathcal{F}_{GIS+}$: This was defined in [GIS+10] and consists of functions which at its core performs the following three steps: (a) decrypt the ciphertext, (b) compute a gate of the circuit being obfuscated and, (c) re-encrypt the ciphertext. This enjoys the same advantage as $\mathcal{F}_{BGL+}$ with the size of the functions being independent of the original circuit being obfuscated.

We enumerate the candidates of indistinguishability obfuscation proposed so far.

1. Garg et al. [GGH+13b]: This was the first candidate construction of indistinguishability obfuscation proposed. This candidate can be instantiated using both [GGH13a] and [CLT13] candidate multilinear maps.

   There are small classes of branching programs for which this candidate is known to be broken [CGH+15] when instantiated using CLT13 mmaps. However, no attacks are currently known for any other setting.

2. Brakerski and Rothblum [BR14]: This candidate can be instantiated using both [GGH13a] and [CLT13] candidate multilinear maps. There are small classes of branching programs for which this candidate is known to be broken when instantiated using GGH13 mmaps [MSZ16]. However, no attacks are currently known for any other setting.

3. Barak et al. [BGK+14]: This candidate can be instantiated using both [GGH13a] and [CLT13] candidate multilinear maps. Many future constructions of iO, notably [AGIS14, MSW14, BMSZ16] adopted this framework. There are two versions of [BGK+14] depending on how the final branching program is obfuscated.

   - *Single Input*: In this model, the final branching program is a matrix branching program consisting of a tuple where every entry corresponds to two matrices. On input x, the evaluation corresponds to choosing one of the two matrices from every entry, depending on x_i for some i, and multiplying all of them in order (say, from left to right).

     There are small classes of branching programs for which this candidate is known to be broken when instantiated using CLT13 mmaps [CGH+15], and when instantiated using GGH13 mmaps [MSZ16]. However, no attacks are currently known for any other setting.

   - *Dual Input*: In this model, the final branching program is a matrix branching program consisting of a tuple where every entry corresponding to *four* matrices. On input x, the evaluation corresponds to choosing one of the four matrices depending on the value of the pair (x_i, x_j) for some i, j with $i \neq j$. Once these matrices are chosen the matrices are multiplied in order, as before, to recover the output.

There are small classes of branching programs for which this candidate is known to be broken when instantiated using GGH13 mmaps [MSZ16]. However, no attacks are currently known for any other setting.

4. Pass et al. [PST14]: This can be instantiated using both [GGH13a] and [CLT13] candidate multilinear maps.

   There are small classes of branching programs for which this candidate is known to be broken when instantiated using GGH13 mmaps [MSZ16]. However, no attacks are currently known for any other setting. There are (different) classes of branching programs for which this candidate is known to be broken when instantiated using CLT13 mmaps [CGH+15]. However, no attacks are currently known for any other setting.

5. Gentry et al. [GLSW15]: This candidate can be instantiated using only [CLT13] candidate multilinear maps.

   There are small classes of branching programs for which this candidate is known to be broken when instantiated using CLT13 mmaps [CGH+15]. Furthermore, the multilinear subgroup elimination assumption used in [GLSW15] is known to be false over CLT13 mmaps. However, no attacks on the iO candidate itself are currently known for any other setting.

6. [Zim15, AB15]: These candidates can be instantiated using only [CLT13] candidate multilinear maps and apply directly to log-depth circuits. There are small classes of circuits for which this candidate is known to be broken when instantiated using CLT13 mmaps [CGH+15]. However, no attacks are currently known for any other setting.

7. iO_{GGH15}: This candidate was proposed by Gorbunov et al. [GGH15]. This is instantiated using a standard lattices-based multilinear maps candidate proposed in the same work. No attacks are known.

8. iO_{fe}: This candidate is obtained through the route of functional encryption. First, we obtain (sub-exponentially secure) functional encryption based on (sub-exponentially secure) [CLT13]-based multilinear map assumptions by using the work of Garg et al. [GGHZ16]. In the second step, we obtain iO from (sub-exponentially secure) functional encryption via the transformations put forward by the works [AJ15, BV15, AJS15]. No attacks are known.

We give a summary of the status of these candidates in Figure 13.

Explanation of entries in Figure 13: The rows indicate all the current known candidates of indistinguishability obfuscation. The row labeled by "Single-Input [BGK+14] `mmaps:[CLT13]`" corresponds to the (single-input version of) [BGK+14] candidate implemented using [CLT13] multilinear maps. The columns indicate the class of functions that are to be obfuscated. BP denotes the class of all poly-sized branching programs. NC^1 denotes the complexity class NC^1. For all candidates, except [Zim15, AB15], NC^1 is obfuscated via Barrington's theorem. The columns under "General Circuits via" corresponds to the class of functions obfuscating which suffices in order to obfuscate arbitrary polynomial-sized circuits. This is achieved by applying so called *bootstrapping theorems* that appeared in prior works [GGH+13b, BGL+15, AJS15]. A circle mark (◯) in $(i,j)^{th}$ entry indicates that currently there is no attack known on the i^{th} candidate for the j^{th} class of functions. 'N/A' indicates 'not applicable'. A cross mark (✕) in $(i,j)^{th}$ entry

indicates that the i^{th} candidate is broken when applied on the j^{th} class of functions.

	BP	NC^1	General Circuits via			
			$\mathcal{F}_{GGH+}$	$\mathcal{F}_{App}$	$\mathcal{F}_{BGL+}$	$\mathcal{F}_{GIS+}$
[GGH+13b] mmaps: [GGH13a]	◯	◯	◯	◯	◯	◯
[GGH+13b] mmaps: [CLT13]	×	◯	◯	◯	◯	◯
[BR14] mmaps: [GGH13a]	×	◯	◯	◯	◯	◯
[BR14] mmaps: [CLT13]	◯	◯	◯	◯	◯	◯
Single-Input [BGK+14] mmaps: [GGH13a]	×	◯	◯	◯	◯	◯
Dual-Input [BGK+14] mmaps: [GGH13a]	×	◯	◯	◯	◯	◯
Single-Input [BGK+14] mmaps: [CLT13]	×	◯	◯	◯	◯	◯
Dual-Input [BGK+14] mmaps: [CLT13]	◯	◯	◯	◯	◯	◯
[PST14] mmaps: [GGH13a]	×	◯	◯	◯	◯	◯
[PST14] mmaps: [CLT13]	×	◯	◯	◯	◯	◯
[GLSW15] mmaps: [CLT13]	×	◯	◯	◯	◯	◯
iO_{GGH15} mmaps: [GGH15]	◯	◯	◯	◯	◯	◯
[Zim15, AB15] mmaps: [CLT13]	N/A	×	◯	◯	◯	◯
iO_{fe} mmaps: [CLT13]	N/A	N/A	◯	◯	◯	◯

Figure 13: ◯ in $(i, j)^{th}$ entry denotes that no attacks are known on the i^{th} candidate when applied on the j^{th} class of functions. Similarly, × indicates that the i^{th} candidate is broken when applied to (a subset of) the j^{th} class of functions. N/A indicates that the entry is not applicable.