A Minor Note on Obtaining Simpler iO Constructions via Depleted Obfuscators

Răzvan Roşie*

Abstract. This paper puts forth a simpler construction for indistinguishability obfuscation (iO) for general circuits. The scheme is concocted from four main ingredients: (1) selectively indistinguishably-secure functional encryption for general circuits having its encryption procedure in complexity class NC^1; (2) universal circuits; (3) puncturable pseudorandom functions having evaluation in NC^1; (4) indistinguishably-secure affine-determinant programs, a notion proposed by works in submission that particularizes iO for specific circuit classes and acts as "depleted" obfuscators. The scheme can be used to build iO for all polynomial-sized circuits in a simplified way. Instantiations can be obtained from sub-exponentially secure learning with errors (LWE).

Keywords: affine determinant programs, branching programs, FE, iO

1 Introduction

Indistinguishability obfuscation (iO) [5] is a central goal in the cryptographic community. Its prime purpose is to make functionally equivalent circuits indistinguishable. Its plethora of applications includes functional encryption, searchable encryption or non-interactive key-exchange protocols [11]. iO can be realized from multilinear maps [15], multi-input functional encryption [18] or compact functional encryption[27]. Nowadays schemes achieving security under well-established assumptions exist [24].

Functional encryption and iO. Functional encryption (FE) provides targeted access over encrypted data. Using the public parameters (abbreviated mpk), any input inp taken from a specified domain can be encrypted as ciphertext CT. Using FE's secret key (abbreviated msk), a functional key – sk_f – can be issued for any function f represented as a polynomial-sized circuit $\mathscr{C}$. One recovers $\mathscr{C}(\mathsf{inp})$ whenever CT is decrypted under sk_f. The major security notion to be accomplished is indistinguishability: as long as $\mathscr{C}(\mathsf{m}_0) = \mathscr{C}(\mathsf{m}_1)$ for two different messages m_0 and m_1, it is hard for any computationally bounded adversary to distinguish if CT encrypts m_0 or m_1 given access to sk_f and mpk (and CT).

Indistinguishability obfuscation appears, at first sight, unrelated to functional encryption. Its interface has the following specification: consider two functionally equivalent circuits: $\mathscr{C}_0$ and $\mathscr{C}_1$, both implementing the same function f. An indistinguishability obfuscator iO takes as input one of them – say $\mathscr{C}_b$ – for a bit b sampled uniformly at random. It releases $\overline{\mathscr{C}}$, such that it is hard for any computationally bounded adversary to distinguish if $\overline{\mathscr{C}}$ was obtained from $\mathscr{C}_0$ or $\mathscr{C}_1$ (the indistinguishability property). We will use the term *depleted* obfuscator to refer to an iO obfuscator for very restricted subclasses of P.

1.1 Our Result

Placing our work in the Context of iO Schemes. This work follows, from a high level, the recipe put forth in [27]: an obfuscator is used to compute a functional ciphertext, and a functional key is issued to decrypt iO's outputs. There are, though, *major* differences: [27] builds a *compact* functional encryption (cFE) scheme using *generically* an exponentially-efficient obfuscator (XiO); the cFE is then used through a sequence of convoluted transforms to build iO. XiO is an obfuscator that is slightly smaller than a lookup table (the trivial obfuscator). Perhaps, at the time, the line of thought therein was focused on *provably* obtaining iO while focus on the realization of a less demanding primitive – the XiO obfuscator.

Herein, the energy is put on building an indistinguishable obfuscator for a very restricted (depleted) class of circuits. We depart from the usage of an XiO and go for a direct construction assuming the existence of: (1) FE with

* Lombard International, razvan.rosie@lombardinternational.com

encryption in NC^1 [19], universal circuits (Uc) [1], puncturable pseudorandom functions (pPRF) with evaluation in NC^1 and affine determinant programs (ADP).

The main idea is to generate a functional key for the Uc. Then, using a different type of obfuscator[2] – the ADP – we can produce functional ciphertexts for messages having their form: "$\mathscr{C}||\mathsf{inp}$" for some binary representation of circuit $\mathscr{C}$ and input inp. By the correctness of FE we get that:

$$\mathsf{FE.Dec}(\mathsf{FE.KGen}(\mathsf{msk},\mathsf{Uc}),\mathsf{FE.Enc}(\mathsf{mpk},\mathscr{C}||\mathsf{inp})) = \mathscr{C}(\mathsf{inp})\ . \tag{1}$$

The ADP (our depleted obfuscator) is used having the master public key of an FE scheme and a (puncturable) PRF key hardcoded. It will produce functional ciphertexts, to be decrypted under the functional key evaluating Uc. A preview of the inner working of our obfuscator is in Equation (2).

$$\begin{aligned}\mathsf{iO.Setup}(\mathscr{C}) &:= \Big(\mathsf{ADP.Setup}(\mathsf{mpk},\mathscr{C},\mathsf{k}),\ \mathsf{FE.KGen}(\mathsf{msk},\mathsf{Uc})\Big)\\ \mathsf{iO.Eval}(\mathsf{inp}) &:= \mathsf{FE.Dec}(\mathsf{sk}_{\mathsf{Uc}},\mathsf{ADP.Eval}(\mathsf{FE.Enc}(\mathsf{mpk},\mathscr{C}||\mathsf{inp};\mathsf{pPRF}(\mathsf{k},\mathsf{inp}))))\end{aligned} \tag{2}$$

Before exploring the main question – how to build such an ADP able to provide "fresh" FE ciphertexts while hiding $\mathscr{C}$ – we provide the intuition behind the indistinguishability proof for our iO obfuscator.

The proof for the obfuscator described in Equation (2) goes by hybridizing over all inputs (similar to [27]): considering two functionally equivalent $\mathscr{C}_0$ and $\mathscr{C}_1$, we will use the indistinguishability of ADP to switch to a setting where pPRF's key is replaced by one punctured in the hybrid game's input. Then, we use pPRF's security to replace the randomness used to produce the FE ciphertext in the current challenge point by true random coins. Next, we use the indistinguishability of the functional encryption for the current input, to replace the ciphertext encoding $\mathscr{C}_0||\mathsf{inp}^*$ with $\mathscr{C}_1||\mathsf{inp}^*$. Finally we switch back. Clearly, the number of hybrids is exponential in input length.

1.2 How to use ADPs to Obfuscate Specific Classes of Circuits

Affine determinant programs [8] were defined to target more efficient and direct obfuscators. The original construction was heuristic and is already broken for the general case using simple, specially-crafted counterexamples. Our work describes how to instantiate a different flavour of ADPs using *augmented* branching programs, thus incurring topological transforms on the underlying graph structure.

Consider a binary input of length n. An ADP is a set of $n+1$ square matrices having entries in $\mathbb{F}_2$. The first one $\mathbf{T}_0$ is referred to as the "base" matrix. Altogether they allow to encode and evaluate one function f representable as an NC^1 circuit. The evaluation procedure computes the determinant obtained by summing up the base matrix to subsets of matrices that are input dependent:

$$f(\mathsf{m}) = \det\left(\mathbf{T}_0 + \sum_{i=1}^{n} \mathsf{m}_i \cdot \mathbf{T}_i\right)\ , \tag{3}$$

where m_i is the i^{th} bit of m. The operation is done over $\mathbb{F}_2$.

A high-level view on ADPs. To build the matrices that form an ADP, one should consider a function in NC^1, as its branching program will have polynomial size[6]. The branching program consists of a start node, two terminal nodes denoted 0 and 1 and a set – variable in size – of regular nodes. The start node has two potential outgoing edges and no incoming edge. The terminal nodes have no or multiple incoming edges and no outgoing edges. The regular nodes receive at least an incoming edge and have two outgoing edges (potentially pointing to the same node). The crux point is that each node (except terminals) is associated (a.k.a. labelled) to some input bit value inp_i. If $\mathsf{inp}_i = 0$ one of the two outgoing edges is selected, when $\mathsf{inp}_i = 1$ the other one gets used. The selection (or settling) of one of the two possible arcs corresponding to each input bit induces an adjacency matrix where on each line, only one out of two possible entries are set to one. If we fix an input – say $\vec{0}$ – we get the core of the base matrix of the ADP from the adjacency matrix of the branching program. Let this core matrix be denoted $\mathbf{G}_0$.

[1] To recap, a universal circuit Uc is itself a circuit that takes as input a description of another circuit $\mathscr{C}$ computing some (abstract) function f as well as the input inp to $\mathscr{C}$ and returns the value of $\mathscr{C}(\mathsf{inp})$; Thus $\mathsf{Uc}(\mathscr{C},\mathsf{inp}) = \mathscr{C}(\mathsf{inp})$.

[2] We show how to build and prove its indistinguishability herein.

Next, we can obtain "difference" matrices: to evaluate input $1||0\ldots||0$, we subtract from the adjacency matrix that we obtained for $1||0\ldots||0$ the "base" matrix $\mathbf{G}_0$. Let this "difference" matrix be $\mathbf{G}_1 \leftarrow \mathbf{G}_{1||0\ldots||0} - \mathbf{G}_{0||0\ldots||0}$ (in general $\mathbf{G}_i$, i being the index of the bit set to 1). This constitutes the core of a *simplified* ADP.

After an intermediate simple post-processing step, a randomization phase is applied: left and right invertible matrices sampled over $\mathbb{F}_2^{m\times m}$ – denoted by $\mathbf{L}$ and $\mathbf{R}$ – are multiplied to each of the $n+1$ matrices.

1.3 The Usage of ADPs in Our Scheme

As mentioned before, the idea is to employ ADPs to generate FE ciphertexts. To this end, FE's encryption procedure itself must be in NC^1. The random coins used by FE.Enc are generated through a pPRF – which itself must be in NC^1. The circuit is described in Figure 1. The trick is to use a limiting variable ρ, originally set to the maximal value of input (and used in the security proof), while the circuit returns an FE ciphertext for the inp that will change during the proof.

$\mathscr{C}^{j}_{\mathsf{mpk},\mathsf{k},\mathfrak{C},\perp,\rho,c}(\mu||i)$:

if $\mu = 0$ return 0;
if $i < \rho$: return $\mathsf{FE}_j.\mathsf{Enc}\big(\mathsf{mpk}, \mathfrak{C}||i; \mathsf{pPRF}(\mathsf{k}, 1||i)\big)$
if $i = \rho$: return c
if $i > \rho$: return $\mathsf{FE}_j.\mathsf{Enc}\big(\mathsf{mpk}, \perp||i; \mathsf{pPRF}(\mathsf{k}, 1||i)\big)$

Fig. 1. ADP computes bitwise a functional ciphertext encrypting the circuit representation $\mathfrak{C}$ and input i, the randomness being obtained through the means of a puncturable pseudorandom function.

Given that each ADP outputs a single bit, we employ the usage of ℓ such circuits, where ℓ is the length of an FE ciphertext. This functional ciphertext, will, in turn, support decryption for functions of t bits length.

One of the major predicaments we face in our indistinguishability proof for ADPs is a different structure of circuits evaluating a pPRF under its *normal* and *punctured* keys[3]. Generally for pPRFs these evaluations circuits differ (see for instance the case of the GGM-based pPRF). To cope with this issue we use a trick that artificially expands the pPRF's input domain with one bit. The key is punctured in an input starting with 0, while always evaluating in an input starting with 1. Thus, we are guaranteed to be able to evaluate the pPRF on the entire original domain. In the proof, we will switch to the usage of a key puncture in point "$1||\mathsf{inp}^*$" for some challenge inp^*, while preserving the same topology of branching program, a crux point in the indistinguishability proof for the ADP, the latter acting as a "depleted obfuscator".

By reusing the previous argument, the strategy we put forth is to retain the same topology for the underlying BP. The only change occurs for a designated set of "sensitive" variables – the nimv – which can be regarded as hardcoded inputs. To this end our security proof will switch values for k, the limit ρ and $\mathscr{C}$'s representation, while preserving the branching program's structure.

Security considerations for ADPs: topology, behaviour and "nimvs". In order to prove the security of our iO-obfuscator, we use a nested level of hybrids. We iterate over the entire input space, and for each challenge input we switch from the encoding of the first circuit $\mathscr{C}_0$ to the representation of $\mathscr{C}_1$. For each challenge inp^*, we first rely on the indistinguishability of the ADP obfuscator to switch the key to one punctured in the current input. Then, we rely on pPRF's security to switch the randomness terms used to compute the challenge ciphertext c corresponding to inp^*. Once we are in a setting where fresh random coins are used, we rely on FE's security to switch to a setting where $\mathscr{C}_1$ is encoded. Finally, we have reverse hybrids that will undo the changes related to pPRF keys and decrease limit ρ.

[3] A puncturable PRF is a pseudorandom function with a normal evaluation mode using a key k and an input m, producing (pseudo-) random values y; a special evaluation mode uses a punctured key k^*, punctured in some point m^* and can compute all PRF values except for $\mathsf{PRF}(\mathsf{k}, \mathsf{m}^*)$.

It should be noted that values such as $\mathsf{k}, \mathscr{C}_0, \mathscr{C}_1$ or ρ are highly sensitive. Revealing them incurs a trivial distinguisher. One of the goals of IND-ADP indistinguishability is, indirectly, to protect such variables, denoted here as *non-input mutable variables*, or nimv. More detailed, one can think at the circuit in Figure 1 as taking 2-inputs: i and nimvs. We will denote by $|\mathsf{nimv}|$ the cumulative length of the binary description of all nimvs.

Remark 1. This work offers a view on ADP as being built from a "proto"-ADP having $|\mathsf{nimv}|+n+1$ matrices. In the setup phase, the nimv values are settled, and the actual base matrix is obtained by summing up the "proto"-ADP base matrix with the $|\mathsf{nimv}|$-matrices obtained by assigning values to nimv variables. In this way, we are guaranteed that the *topology* of the "proto"-branching program will remain the same even if the nimv variables are going to change (which is the case during the security proof). We will sometimes write that ADPs have "embedded" nimv values.

ADPs as Depleted Obfuscators. In our view, indistinguishability obfuscation is related to hiding structure for circuits. The goals are similar for ADPs: preserve the structure/topology while "shielding" the nimvs. Given the *lack* of average-case assumptions related to ADPs, our work considers a different approach – perfect security[4]:

$$\mathsf{ADP.Setup}\left(1^\lambda, \mathscr{C}_{\mathsf{nimv}_0}; R_0\right) = \mathsf{ADP.Setup}\left(1^\lambda, \mathscr{C}_{\mathsf{nimv}_1}; R_1\right) \tag{4}$$

The equation above states that for any random coins R_0 used during the ADP.Setup w.r.t. circuit $\mathscr{C}_0$, there is a unique term R_1 that can be used by ADP.Setup to produce the same program output, but with respect to $\mathscr{C}_1$. [7] puts forth a series of requirements on the function considered[5] which, if fulfilled, would have been sufficient to prove the security of IND-ADP. These conditions are to be introduced informally below, all but the 6th being unchanged. The same conditions were used in a context of a different primitive (in [3], but not implying iO). The new set will indirectly allow us to prove the security of iO. The conditions are introduced informally below, and explained afterwards (caveat: the circuit description has input length incremented by 1):

Theorem 1 (Informal). *Let $f : \{0,1\}^{|nimv|+n} \to \{0,1\}$ denote a binary function and let $\mathfrak{C}_{d,|nimv|+(n+1)}$ stand for a class of circuits of depth d with input length $|nimv| + (n+1)$, satisfying conditions (1) → (6) below:*

Condition 1 *Every $\mathscr{C} \in \mathfrak{C}_{d,|nimv|+(n+1)}$ implements the two-input function*

$$f : \{0,1\}^{|nimv|+n} \to \{0,1\} .$$

Condition 2 *A condition for $\mathfrak{C}_{d,|nimv|+(n+1)}$ to admit a* IND-ADP*-secure implementation is*

$$d \in O(\log_2(|nimv| + (n+1))) .$$

Condition 3 *For every $\mathscr{C} \in \mathfrak{C}_{d,|nimv|+(n+1)}$ implementing some $f : \{0,1\}^{|nimv|+n} \to \{0,1\}$, there exists a* PPT *algorithm $\mathscr{R}$ such that:*

$$\Pr\left[\mathscr{C}(|nimv|, \mathsf{inp}) = 1 \middle| (|nimv|, \mathsf{inp}) \leftarrow \mathscr{R}(1^\lambda, \mathscr{C})\right] > \frac{1}{\mathsf{poly}(|nimv| + (n+1))} .$$

Condition 4 *For every $\mathscr{C} \in \mathfrak{C}_{d,|nimv|+(n+1)}$ modelling some f, there exists a* PPT *algorithm $\mathscr{R}$ such that $\forall\ \mathsf{inp} \in \{0,1\}^{(n+1)}$,*

$$\Pr[nimv \neq nimv' \wedge \mathscr{C}(nimv, \mathsf{inp}) = \mathscr{C}(nimv', \mathsf{inp}) | (nimv, nimv') \leftarrow \mathscr{R}(1^\lambda, f)] > \frac{1}{\mathsf{poly}(|nimv| + (n+1))} .$$

Condition 5 *For every $\mathscr{C} \in \mathfrak{C}_{d,|nimv|+(n+1)}$ modelling some $f : \{0,1\}^{|nimv|+n} \to \{0,1\}$, there exists and is "easy to find" nimv such that:*

$$\mathscr{C}(nimv, b||\mathsf{inp}') := \begin{cases} 1 , & \text{if } b = 0. \\ f(nimv, \mathsf{inp}') , & \text{if } b = 1 \text{ and } \forall \mathsf{inp}' \in \{0,1\}^n \end{cases} .$$

[4] Note that this approach will **not** provide a perfectly secure obfuscator (which is impossible), the scheme relying on the security of FE and puncturable PRFs.

[5] E.g., it should be non-zero, a finding independently meeting an earlier result of [20].

Condition 6 *Considering the underlying branching program representation of some $\mathscr{C} \in \mathfrak{C}_{d,|\mathsf{nimv}|+(n+1))}$, on any local path, any index of a vertex depending on input is greater than any index of a vertex depending on* nimv.

Then (by Theorem 2, below) there is an IND-ADP*-secure* ADP *for the class $\mathfrak{C}_{d,|\mathsf{nimv}|+(n+1)}$. Moreover, assuming the existence of universal circuits, puncturable pseudorandom functions with evaluation in* NC^1 *and* FE *with encryption in* NC^1 *there is an indistinguishably-secure* iO *scheme for functions in class $\mathfrak{C}_{d',|\mathsf{nimv}|+(n+1)}$. Furthermore, the advantage of any adversary $\mathscr{A} := (\mathscr{A}_1, \mathscr{A}_2, \mathscr{A}_3)$ running in polynomial time against the security of the scheme is bounded as follows:*

$$\mathbf{Adv}^{\mathsf{iO}}_{\mathscr{A},\mathsf{cFE}}(\lambda) \leq 2^n \cdot \left((2\ell+1)\cdot \mathbf{Adv}^{\mathsf{IND\text{-}ADP}}_{\mathscr{A}_1,\mathsf{ADP}}(\lambda) + \mathbf{Adv}^{\mathsf{s\text{-}IND\text{-}FE\text{-}CPA}}_{\mathscr{A}_2,\mathsf{FE}}(\lambda) + 2\cdot \mathbf{Adv}^{\mathsf{puncture}}_{\mathscr{A}_3,\mathsf{pPRF}}(\lambda)\right)$$

where ℓ represents the output length of the FE *ciphertext, λ is the security parameter and* pPRF *stands for a puncturable* PRF.

We note that the first four requirements above are fulfilled by any circuit representation that is: (1) represents a bivariate function, (2) in NC^1, (3) represents a non-constant function and (4) is functionally equivalent to a circuit built under different nimvs. The fifth condition is needed in the proof (described below), while the most challenging part is achieving (6). In the forthcoming part we review transforms that produce augmented BPs respecting constraints (5) and (6) while allowing an indistinguishability proof to go through.

1.4 Perfect Security for Specific ADP Classes

The direct iO construction we introduced above requires ordering constraints for nodes occurring within the branching program corresponding to the circuit in Figure 1. Such requirements are used to prove the indistinguishability (in fact *perfect* security) for ADPs. Next, we state a result on perfectly-secure ADPs, present the topological transforms to get condition (6), then sketch the result's proof.

Theorem 2 (IND-ADP**-admissible circuits).** *Let $\mathfrak{C}_{d,|\mathsf{nimv}|+(n+1)}$ denote a class of circuits satisfying conditions (1) → (6) described in Theorem 1. Then, there exists an* ADP *that reaches perfect security with respect to every single $\mathscr{C}$ in the admissible class $\mathfrak{C}_{d,|\mathsf{nimv}|+(n+1)}$.*

By far, reordering nodes such that on any internal path, nodes depending on sensitive variables (nimvs) appear in front of nodes that are settled by inputs is the *most challenging* requirement. The "bare metal" transformation we put forth happens directly on BP's structure, as opposed to the higher level ones from [8]. We show the transforms, and after that motivate the constraints from Theorem 1 while introducing the indistinguishability proof. For clarity, given only the first two requirements mentioned in Theorem 1 is enough to obtain, through the transforms below the last two conditions (5) and (6). However, for the IND-ADP proof to go through, conditions (3) and (4) are necessary.
Consider the adjacency matrix $\mathbf{G}_0$ over $\mathbb{F}_2^{m\times m}$ corresponding to input $0||\ldots||0$, described above. Two transforms, taken from [3], are applied: augmentation with (i) *auxiliary* and (ii) *flare* nodes. The outcome – augmented branching programs – are slightly post-processed and then randomized with left/right randomization matrices $\mathbf{L}$ and $\mathbf{R}$. The latter have entries over $\mathbb{F}_2$ and are non-singular.

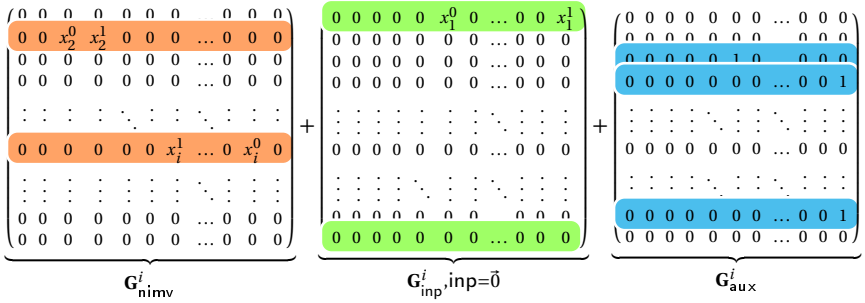
Auxiliary nodes.[3] Our goal of achieving perfect security meets a first obstacle: how to "isolate" the nimv-dependent vertices within BP from the remaining ones? When we use $\mathbf{L}$ and $\mathbf{R}$ to "randomize" the adjacency matrix, we want to "set apart" the columns of $\mathbf{L}$ used during multiplication with nimv-dependent entries of the adjacency matrix. Such a segregation of randomizing columns happens when no vertex within BP gets concomitantly direct incoming arcs from a nimv vertex and from an input vertex. If so, within any column corresponding to nodes that receive incoming arcs form nimv-dependent vertices, there will be no pair of entries set to 1. The transform put forth in [3] involves inserting "dummy" auxiliary intermediate nodes on every arc connecting a nimv node to its successor. In this way, when the augmented $\mathbf{G}_0$ is left-multiplied with $\mathbf{L}$, the sets of columns of $\mathbf{L}$ to be multiplied with the columns in $\mathbf{G}_0$ that depend on nimv and inp nodes will be *disjoint*.

Flare nodes. [3] Flare nodes are used to make sure that on every path within BP (already augmented with auxiliary nodes), vertices settled by nimv variables appear, intuitively, "above" nodes depending on inp. To delve into this,

consider a path originating in some nimv-dependent vertex. When we follow the arcs within this path, we can find some inp-dependent vertex v_{inp} having an outgoing arc that points to some other vertex v_{nimv}, settled by a nimv; the trick is to disrupt such paths by injecting "dummy" *flare* vertices. The newly injected flare node has two outgoing arcs: one points to terminal node 1; the other one points to v_{nimv}. By default $\mathbf{G}_0$ contains the arc of the "flare" node that points to terminal-node 1. Adding the (dummy) matrix $\mathbf{G}_1$ to $\mathbf{G}_0$, re-enables the normal flow. Thus, injecting flares gives a way to preserve a node ordering (the gist of the proof).

After the original BP was augmented, a post-processing step is performed for every $\{\mathbf{G}_i\}_{i=0}^{n}$. The last row and first column of each $\mathbf{G}_i$ are pulled out and a second diagonal is attached. Let $\overline{\mathbf{G}}_i$ be the resulting square matrix, and it can be viewed as consisting of rows associated (as an adjacency matrix) to nimvs (red), inputs (green) or auxiliary (blue). An example is below.

$$\overline{\mathbf{G}}_0 = \overline{\mathbf{G}}_{\mathsf{nimv}||0\ldots0} = \text{Second Diagonal+}$$



The split on the columns of the left matrix $\mathbf{L}$ is exemplified below.

$$\mathbf{L}^i = \underbrace{\begin{pmatrix} 0 & l_{1,2} & 0 & 0 & \ldots & l_{1,i} & \ldots & 0 & 0 \\ 0 & l_{2,2} & 0 & 0 & \ldots & l_{2,i} & \ldots & 0 & 0 \\ 0 & l_{3,2} & 0 & 0 & \ldots & l_{3,i} & \ldots & 0 & 0 \\ 0 & l_{4,2} & 0 & 0 & \ldots & l_{4,i} & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ 0 & l_{m,2} & 0 & 0 & \ldots & l_{m,i} & \ldots & 0 & 0 \end{pmatrix}}_{\mathbf{L}^i_{\mathsf{nimv}}} + \underbrace{\begin{pmatrix} l_{1,1} & 0 & 0 & 0 & \ldots & 0 & \ldots & 0 & l_{1,m} \\ l_{2,1} & 0 & 0 & 0 & \ldots & 0 & \ldots & 0 & l_{2,m} \\ l_{3,1} & 0 & 0 & 0 & \ldots & 0 & \ldots & 0 & l_{3,m} \\ l_{4,1} & 0 & 0 & 0 & \ldots & 0 & \ldots & 0 & l_{4,m} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ l_{m,1} & 0 & 0 & 0 & \ldots & 0 & \ldots & 0 & l_{m,m} \end{pmatrix}}_{\mathbf{L}^i_{\mathsf{inp}}} + \underbrace{\begin{pmatrix} 0 & 0 & l_{1,3} & l_{1,4} & \ldots & 0 & \ldots & l_{1,m-1} & 0 \\ 0 & 0 & l_{2,3} & l_{2,4} & \ldots & 0 & \ldots & l_{2,m-1} & 0 \\ 0 & 0 & l_{3,3} & l_{3,4} & \ldots & 0 & \ldots & l_{3,m-1} & 0 \\ 0 & 0 & l_{4,3} & l_{4,4} & \ldots & 0 & \ldots & l_{4,m-1} & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & l_{m,3} & l_{m,4} & \ldots & 0 & \ldots & l_{m,m} & 0 \end{pmatrix}}_{\mathbf{L}^i_{\mathsf{aux}}}$$

We show that a circuit family matching the conditions from Theorem 1 is a perfectly-secure (and thus IND-ADP-secure) ADP program (obfuscator).

Proof sketch for Theorem 2. First we use the *auxiliary nodes* transformation. Specifically, it is used to split the matrix $\mathbf{L}$ as follows:

$$\mathbf{L} = \mathbf{L}_{\mathsf{nimv}} + \mathbf{L}_{\mathsf{inp}} + \mathbf{L}_{\mathsf{aux}} \tag{5}$$

A similarly (although not symmetric) split can be done for $\mathbf{R}$, in rows set by nimvs ($\mathbf{R}_{\mathsf{nimv}}$) or not ($\widetilde{\mathbf{R}_{\mathsf{nimv}}}$):

$$\mathbf{R} = \mathbf{R}_{\mathsf{nimv}} + \widetilde{\mathbf{R}_{\mathsf{nimv}}}\ . \tag{6}$$

Using a "colouring" type of proof, it can be observed that:

1) $\mathbf{L}_{\mathsf{nimv}} \cdot \left(\mathbf{G}_{\mathsf{aux}} \cdot \widetilde{\mathbf{R}_{\mathsf{nimv}}}\right) = 0$. 2) For all $j \in [n]$: $\mathbf{L}_{\mathsf{nimv}} \cdot \left(\mathbf{G}_{\mathsf{inp}_j} \cdot \widetilde{\mathbf{R}_{\mathsf{nimv}}}\right) = 0$.
3) $\mathbf{L}_{\mathsf{aux}} \cdot \left(\mathbf{G}_{\mathsf{nimv}} \cdot \mathbf{R}_{\mathsf{nimv}}\right) = 0$. 4) For all $j \in [n]$: $\mathbf{L}_{\mathsf{aux}} \cdot \left(\mathbf{G}_{\mathsf{inp}_j} \cdot \widetilde{\mathbf{R}_{\mathsf{nimv}}}\right) = 0$.
5) For all $j \in [n]$: $\mathbf{L}_{\mathsf{inp}_j} \cdot \left(\mathbf{G}_{\mathsf{nimv}} \cdot \mathbf{R}_{\mathsf{nimv}}\right) = 0$.
6) For all $j \in [n]$: $\mathbf{L}_{\mathsf{inp}_j} \cdot \left(\mathbf{G}_{\mathsf{aux}} \cdot \widetilde{\mathbf{R}_{\mathsf{nimv}}}\right) = 0$.
7) For all $j \neq j'$: $\mathbf{L}_{\mathsf{inp}_{j'}} \cdot \left(\mathbf{G}_{\mathsf{inp}_j} \cdot \widetilde{\mathbf{R}_{\mathsf{nimv}}}\right) = 0$.

Using the observations above, one can rewrite the PS-ADP in Equation (4) as:

$$\begin{cases} \left(\mathbf{L}_{\mathsf{nimv}} + \mathbf{L}_{\mathsf{aux}} + \mathbf{L}_{\mathsf{inp}}\right) \cdot \overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}} \cdot \left(\mathbf{R}_{\mathsf{nimv}} + \widetilde{\mathbf{R}_{\mathsf{nimv}}}\right) = \left(\mathbf{L}'_{\mathsf{nimv}} + \mathbf{L}'_{\mathsf{aux}} + \mathbf{L}'_{\mathsf{inp}}\right) \cdot \overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}} \cdot \left(\mathbf{R}'_{\mathsf{nimv}} + \widetilde{\mathbf{R}_{\mathsf{nimv}}}'\right) \\ \quad \text{and} \\ \mathbf{L}_{\mathsf{inp}_j} \cdot \left(\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{j}} - \overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}\right) \cdot \widetilde{\mathbf{R}_{\mathsf{nimv}}}_j = \mathbf{L}'_{\mathsf{inp}_j} \cdot \left(\overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{j}} - \overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}}\right) \cdot \widetilde{\mathbf{R}_{\mathsf{nimv}}}'_j\ , \quad \forall j \in [n] \end{cases}$$

Because $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{j}} - \overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}} = \overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{j}} - \overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}}$, we can deliberately use the substitution $\mathbf{L} = \mathbf{L}'$ and $\widetilde{\mathbf{R}_{\mathsf{nimv}\,j}} = \widetilde{\mathbf{R}_{\mathsf{nimv}\,j}}'$ and remove the need to tackle all but the first equation above. We rewrite the first equation in the system above as:

$$\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}} \cdot \mathbf{A} + \mathbf{B} \cdot \overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}} = 0, \text{ where } \mathbf{B} \leftarrow \mathbf{I}_m \text{ and } \mathbf{A} \leftarrow \mathbf{I}_m + \overline{\mathbf{G}}^{-1}_{\mathsf{nimv}||\vec{0}} \cdot \mathbf{S}$$

and $\mathbf{S}$ is a matrix storing the differing nodes between nimv and nimv'.

We make use of the condition in Theorem 1 that $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$ is non-singular, else the base input $\vec{0}$ is changed to some other value (up to this point the first *four* requirements in Theorem 1 are used).

Next, the multiplication $\overline{\mathbf{G}}^{-1}_{\mathsf{nimv}||\vec{0}} \cdot \mathbf{S}$ is analysed. The idea is to use an invariant that will follow the arcs in the augmented BP backwards – "going up" and "going left" – proving that column j within $\overline{\mathbf{G}}^{-1}_{\mathsf{nimv}||\vec{0}}$ multiplied with $\mathbf{S}$ equates 0. This constraint is used to show $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}} \cdot \mathbf{A} + \mathbf{B} \cdot \overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}} = 0$. The invariant uses the topology of the matrix $\overline{\mathbf{G}}_0$ to derive a contradiction: we will assume that the aforementioned dot-product (line j in $\overline{\mathbf{G}}^{-1}_{\mathsf{nimv}||\vec{0}}$ with a column within $\mathbf{S}$) cannot be 0.

A "bad" event occurs when a row depending on input – j – within $\overline{\mathbf{G}}^{-1}_{\mathsf{nimv}'||\vec{0}}$ is multiplied over $\mathbb{F}_2$ by a column indexed by c in $\mathbf{S}$ the result being 1 (not 0 as desired). It is known that column c is included (as a sub-column) within $\overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}}$; since line j of $\overline{\mathbf{G}}^{-1}_{\mathsf{nimv}'||\vec{0}}$ is multiplied to a column indexed by $c \neq j$ of $\overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}}$, the result is 0 (basic algebra).

Following from what was previously remarked, we find that line j in $\overline{\mathbf{G}}^{-1}_{\mathsf{nimv}'||\vec{0}}$ must have an element set to 1 occurring to the left of j, and this element is multiplied with some value of 1 occurring in column c within $\mathbf{S}$. Else, it is not possible that the dot product of these two vectors is 1 over $\mathbb{F}_2$.

Suppose the entry set to 1 described above occurs in some position h. Take the column in $\overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}}$ that has 1 on position h. By recycling the argument above we end-up traversing the directed edges in the BP "backwards" up until a *flare* vertex stops from going more backwards. This is the way to derive the contradiction. That is, while the arcs are traversed upwards and the value of $c \leftarrow h$ gets updated per each iteration, we are guaranteed that every node that is visited is linked to a nimv. Put differently, is is always the case that $j \neq c$.

Given that we employ flare nodes in the augmented BP, the last nimv dependent node on the path is short on any incoming arcs; thus, line j of $\overline{\mathbf{G}}^{-1}_{\mathsf{nimv}'||\vec{0}}$ multiplied by column $j \neq c$ within $\overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}}$ is 1, which is impossible.

Therefore, indistinguishability is guaranteed unconditionally for classes of circuits obeying to requirements (1) to (6) in Theorem 1.

1.5 Finding Suitable FE schemes: AR17 and Decomposability

A secondary result regards the applicability of a specific *decomposable* functional encryption schemes in providing a poly-sized branching program, and then an ADP that satisfies constraints defined in Theorem 1. To this end we explore in Appendix B the circuit complexity of the encryption procedure for one scheme outputting a single bit that can be instantiated from LWE. A different scheme, that can be instantiated from RLWE, is described in Appendix C. Below, we provide some insights on tackling this second scheme. Readers may skip this part and come back to it when delving into Appendix C.

Remark 2. Let sFE stand for the functional encryption scheme defined in [2] supporting a single functional key derivation query. Then there exists a set of IND-ADP admissible circuit computing the ciphertexts of this scheme.

We start by providing a quick overview of the scheme, followed by its representation with respect to a Chinese Remainder Theorem (CRT) coefficient embedding, and how this representations eases our topological constraints.

Decomposable Functional Encryption. Several instantiations of functional encryption schemes exhibit an extra property, dubbed *decomposability*. Imagine that the plaintext m is split into Ω bits[6]. If the encryption algorithm obtains a ciphertext $\mathsf{CT} = \{\mathsf{CT}_1, \ldots, \mathsf{CT}_\Omega\}$ where each CT_i encrypts a single bit m_i of plaintext, in isolation from all $\mathsf{m}_{j \neq i}$, we say that the ciphertext is *decomposable*.

[6] In general, it may be split into chunks, but we consider digital *bits* for simplicity.

Such a construction, (we are only interested in the simper version targeting NC^1 circuits), has been put forward in the work of Agrawal and Rosen [2] assuming the hardness of the decisional RLWE problem [28]. Their construction resembles a levelled fully-homomorphic encryption scheme (FHE) [12] and encrypts each bit of the plaintext, independently, as follows:

- first multiplication level – the ciphertext consists of:

$$\mathsf{CT}_i^1 \leftarrow a \cdot s + p_0 \cdot e + \mathsf{m}_i \ \in \mathscr{R}_{p_1} \tag{7}$$

  where $\mathsf{m}_i \in \mathscr{R}_{p_0}$ and $s \leftarrow_{\$} \mathscr{R}_{p_1}$ while $e \leftarrow_{\chi} \mathscr{R}_{p_1}$; $a \in \mathscr{R}_{p_1}$ is provided through public parameters.
- for the second multiplication level, two ciphertexts are obtained

$$a' \cdot s + p_1 \cdot e' + \mathsf{CT}_i^1 \ \in \ \mathscr{R}_{p_2}$$

  and

$$a'' \cdot s + p_1 \cdot e'' + (\mathsf{CT}_i^1 \cdot s) \ \in \mathscr{R}_{p_2} \ .$$

  The computational pattern is repeated recursively up to d multiplication levels (d is the depth of the circuit representing f), and then for every bit of the input.
- between any two multiplication layers i and $i+1$, an addition layer is interleaved: it "duplicates" the ciphertexts in layer i and uses its modulus p_i. For simplicity in this work, we will entirely avoid describing multiplication layers.
- the public key (the "a"s) corresponding to the last layer are also the master public key mpk of a linear functional encryption scheme $\mathsf{Lin\text{-}FE}$.
- the decryptor will evaluate f obliviously over the ciphertext, layer by layer, finally obtaining

$$\mathsf{CT}_{f(x)} = f(x) \ + \ \text{noise} \ + \ (\mathsf{sk}_f\text{-dependent term}) \tag{8}$$

  and uses its functional key sk_f to recover $f(x)$ (after removing the noise).

As it can be easily observed (and also stated by its authors), such a scheme achieves *decomposability*. When looking to Figure 1, we see that the we need a circuit to compute each bit of the AR17 ciphertext. The beneficial aspect is the simple structure of AR17 ciphertext. Still, there are *two* major constraints that prevents us to limpidly think about branching programs that output AR17 ciphertexts:

- we work over a quotient ring of a polynomial ring, and performing arithmetic on this structure is not very intuitive, since we need to work with polynomial remainders.
- we use puncturable pseudorandom functions for NC^1 in a black box way, and we may need to delve into pPRFs instantiations as well.

In order to show that AR17's encryption procedure is ADP admissible, our main technique is as follows: we know that the quotient ring $\mathscr{R}_{p_1} := \mathscr{R}/p_1\mathscr{R}$, $\mathscr{R} := \mathbb{Z}[X]/(X^N+1)$ on which level 1 AR17 ciphertexts are defined, is isomorphic to a ring structure (herein referred to as the CRT embedding):

$$\mathscr{R}/p_1\mathscr{R} \cong \left(\mathscr{R}/\mathscr{I}_1 \times \ldots \times \mathscr{R}/\mathscr{I}_N\right) \tag{9}$$

where both the addition and multiplication of two elements is done coordinate-wise and $p_1\mathscr{R} := \cap_{i=1}^{N} \mathscr{I}_i$ and each $\mathscr{R}/\mathscr{I}_i$ is a field of prime characteristic p_1.

Step 1. We rewrite each level 1 ciphertext $a \cdot s + p_0 \cdot e + \mathsf{m}_i$ corresponding to m_i. Its explicit form written as:

$$\left(\sum_{j=0}^{N-1} a_{(j)} \cdot X^j\right) \cdot \left(\sum_{j=0}^{N-1} s_{(j)} \cdot X^j\right) + p_0 \cdot \left(\sum_{j=0}^{N-1} e_{(j)} \cdot X^j\right) + \mathsf{m}_i \tag{10}$$

becomes with respect to this CRT embedding:

$$\Big(\overline{a_{(0)}} \cdot \overline{s_{(0)}} + p_0 \cdot \overline{e_{(0)}} + \mathsf{m}_i, \ldots, \overline{a_{(j)}} \cdot \overline{s_{(j)}} + p_0 \cdot \overline{e_{(j)}} + \mathsf{m}_i, \ldots, \overline{a_{(N-1)}} \cdot \overline{s_{(N-1)}} + p_0 \cdot \overline{e_{(N-1)}} + \mathsf{m}_i\Big) \tag{11}$$

In Equation (11), we use the fact that a constant c that does not need to be reduced modulo any ideal $\mathscr{I}_i$ in Equation (9) is mapped to a constant element having c on all coordinates.

The benefits of the CRT representation are immediately observable: the multiplications and additions are performed component-wise, modulo q. Thus, it becomes easier to reason about branching programs producing ciphertexts with respect to the CRT embedding.

The next high-level step is to consider the branching program outputting one bit of some coordinate in the CRT representation. Essentially, the BP's output is set to one bit from the following computation:

$$\overline{a_{(j)}} \cdot \overline{s_{(j)}} + p_0 \cdot \overline{e_{(j)}} + \mathsf{m}_i \tag{12}$$

The values of each $\overline{a_{(j)}}$ and p_0 are public and they can be "hardcoded" in the branching program. On the other hand, we need to obtain the RLWE secret $\overline{s_{(j)}}$ and the noise element $\overline{e_{(j)}}$.

To compute $\overline{s_{(j)}}$, we employ directly the puncturable PRF in NC^1 and restrict its output to $\mathbf{F}_q$. To compute the noise, things get more complex:

$$\overline{e_{(j)}} = \sum_{k=0}^{N-1} e_{(k)} \cdot \theta^{k \cdot (2j-1)}\,, \tag{13}$$

where θ is the $2N$ root of unity. The reason is that noise is not normally distributed with respect to the CRT embedding.

We will get each term $e_{(j)}$ by calling the puncturable PRF on input i and then extracting (for simplicity) a Binomial term, rather than a Gaussian one.

To sum things up, Equation (12) can be rewritten as:

$$g(i) + \Big(\sum_{k=0}^{N-1} p_0 \cdot h_k(i)\Big) + \mathsf{m}_i \tag{14}$$

Once the elements are manipulated in this pseudo-linear form, we present our BP-related technique to tackle conditions 5 and 6 from Theorem 1.

Step 2. We want to ensure that the BP representation has, on any path containing nimv-dependent nodes, the vertices depending on nimvs "on top of" the nodes depending on input. The main remark we make herein is related to a *locality* property: we observe the BP corresponding to Equation (14) can be visualized as a branching program simulating addition with "carry" (and in the end the "mod" operand), where the additions correspond to the terms involving the function g and h. Instead of having a large augmented branching program, we can compose multiple, smaller, "inner" branching programs and reduce the size of the "outer" branching program.

As stated in the previous subsections, in order to handle node ordering (Condition 6), we introduce flare nodes depending on input bit 1 between any two gadget inner BPs used by the outer BP: considering its base matrix $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$, we introduce between every two connected summand inner BPs a "flare" node depending on input 1 that points to the terminal node 1. This acts as a "switch[7]" in the BP representation, as it interrupts the flow from the start node to the terminal node, between additions performed in Equation (14).

When the program is evaluated in some input stating with 1, the flow is enabled, as we switch back to the normal evaluation by adding the complementary lines to $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$.

Assuming familiarity with the proof in Section 1.3, we remark that if the "bad" event happens – when we multiply the line depending on input j in the $\overline{\mathbf{G}}^{-1}_{\mathsf{nimv}'||\vec{0}}$ with some column c in $\mathbf{S}$ and we follow the arcs "upwards" – this

[7] Like an electrical switch.

switch will prevent us for going up. A bit more detailed if the inner product between line j and column c is different by zero, we consider the product between line j and column $c \neq j$ of $\overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}}$: this product has to be 0 but we know that there is one entry set to 1 (due to the product with the column of $\mathbf{S}$ "included" in $\overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}}$). So we follow the BP arcs in reverse order, up to the point we hit the interruption by the flare node, which leaves us with the event that line j of $\overline{\mathbf{G}}^{-1}_{\mathsf{nimv}'||\vec{0}}$ multiplied with column $c' \neq j$ of $\overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}}$ is 1. Because we reach the contradiction, the inner product between line j and the original column c of $\mathbf{S}$ has to be 0. Thus, we can ensure that the indistinguishability property holds given that nodes are ordered locally as desired (nodes depending on nimvs occur "on top of" nodes depending on inputs). This is clearly advantageous, as we do not need a single ordering for the entire BP.

Then we turn to the inner BP corresponding to functions like g, h_k, which use puncturable pPRFs and Binomial/Gaussian noise extractors. The gist is to reuse the same idea, i.e. to introduce nodes depending on the first input bit. Such nodes act as switches and provide a space-efficient alternative compared to full node reordering [8]. To introduce flare nodes in the inner BP corresponding to g, observe that one has to interrupt only the local paths breaking Condition 6.

Here, we point out that such an approach of augmenting and composing inner BPs representing ciphertexts having their form described by Equation (14) may constitute a more space-efficient alternative to obtaining the full branching program and after that augment it with auxiliary/flare nodes.

Step 3. Everything that has been presented up to this stage was in relation with ADP representation for level 1 ciphertexts in AR17. The next step is to extend the approach to higher level ciphertexts.

Fortunately, the idea is relatively straightforward, just that the higher the level, the more complex Equation (14) becomes. This is because it needs to cope with the multiplication of functions similar to g. We present this extension, in full, as Appendix C for this work.

How to read this paper. Section 2 introduces the standard notations and definitions. Section 2.2 reviews the construction of randomized encodings from branching programs. In Section 3 we introduce our scheme and prove the iO based on ADPs' indistinguishability (Section 4 and detailed in Appendix D), while in Section 5 we review the conditions for getting perfect security. In Section 6 we provide a detailed look into ADPs satisfying conditions formalized in Section 5 and provide a partially different proof from [3] that they achieve indistinguishability. Appendices A and B analyse the circuit complexity of a specific pPRF's evaluation, respectively FE's encryption.

2 Background

Notation. An algorithm is equivalent to a Turing machine and receives the security parameter denoted by $\lambda \in \mathbb{N}^*$ in unary representation (denoted by 1^λ). Unless mentioned, an algorithm herein is randomized, and in many cases we use PPT algorithms: their runtime is "probabilistic polynomial-time" in the security parameter. Given an algorithm $\mathscr{A}$ running $\mathscr{A}$ on input(s) $(1^\lambda, x_1, \ldots)$ with uniform random coins r and assigning the output(s) to $(y_1, \ldots)$ is defined as $(y_1, \ldots) \leftarrow_\$ \mathscr{A}(1^\lambda, x_1, \ldots; r)$. If $\mathscr{A}$ has access to an oracle $\mathscr{O}$, we write $\mathscr{A}^{\mathscr{O}}$. For $k \in \mathbb{N}^*$, we define $[k] := \{1, \ldots, k\}$; $|S|$ is the cardinality of a finite set S; the action of sampling an uniformly at random element x from X by $x \leftarrow_\$ X$. Bold lowercase variables – $\mathbf{w}$ – represent column vectors and bold upercase matrices – $\mathbf{A}$. A subscript $\mathbf{A}_{i,j}$ indicates an entry in the matrix. A real-valued function $\text{NEGL}(\lambda)$ is negligible if $\text{NEGL}(\lambda) \in \mathscr{O}(\lambda^{-\omega(1)})$. $||$ stands for concatenation. Circuits are used to represent (abstract) functions. Unless mentioned, n stands for the input length of the circuit, s for its size and d for its depth.

2.1 Basic Definitions

Definition 1 (Learning with Errors[29]). *Given $q = q(\lambda) \geq 2$ in $\mathbb{N}^*$ and an error distribution $\chi = \chi(\lambda)$ defined on $\mathbb{Z}_q$, the learning-with-errors problem asks to distinguish the following distributions: $\{(\mathbf{A}, \mathbf{A}^\top \cdot \mathbf{s} + \mathbf{e})\}$ and $\{(\mathbf{A}, \mathbf{u})\}$, where $\mathbf{A} \leftarrow_\$ \mathbb{Z}_q^{n \times m}, \mathbf{s} \leftarrow_\$ \mathbb{Z}_q^n, \mathbf{e} \leftarrow_\chi \mathbb{Z}_q^m, \mathbf{u} \leftarrow_\$ \mathbb{Z}_q^m$.*

[8] Node reordering is always possible at the cost of having an exponential size for BP.

Definition 2 (Puncturable PRFs [30]). *A puncturable pseudorandom function* pPRF *is a set of* PPT *procedures* $(\mathsf{pPRF.Setup}, \mathsf{pPRF.Eval}, \mathsf{pPRF.Puncture})$*:*

$K_{\mathsf{pPRF}} \leftarrow_{\$} \mathsf{pPRF.Setup}(1^{\lambda})$*: samples* K_{pPRF} *uniformly at random over keyspace.*
$K^{*}_{\mathsf{pPRF}} \leftarrow \mathsf{pPRF.Puncture}(K_{\mathsf{pPRF}}, \mathsf{m}^{*})$ *: given a point* m^{*} *from the input space, and* K_{pPRF}*, returns the punctured key* K^{*}_{pPRF}*.*
$Y \leftarrow \mathsf{pPRF.Eval}(K_{\mathsf{pPRF}}, \mathsf{m})$*: returns the output of the pseudorandom function.*

Correctness: *given any* $\mathsf{m}^{*} \in \mathcal{M}$*, any* $K_{\mathsf{pPRF}} \in \mathcal{K}$ *and any* $\mathsf{m} \neq \mathsf{m}^{*} \in \mathcal{M}$*:* $\mathsf{pPRF.Eval}(K_{\mathsf{pPRF}}, \mathsf{m}) = \mathsf{pPRF.Eval}(K^{*}_{\mathsf{pPRF}}, \mathsf{m})$*, where* $K^{*}_{\mathsf{pPRF}} \leftarrow \mathsf{pPRF.Puncture}(K_{\mathsf{pPRF}}, \mathsf{m}^{*})$*. Moreover, the distribution of* pPRF*'s outputs indistinguishable (computationally) from the uniform one. Even when punctured key* K^{*}_{pPRF} *is revealed, no* PPT *adversary can distinguish between* $\mathsf{pPRF.Eval}(K_{\mathsf{pPRF}}, \mathsf{m}^{*})$ *and* $R \leftarrow_{\$} \mathcal{R}$*.*

Definition 3 (Functional Encryption - Public Key Setting[10]). *A public-key functional encryption scheme* FE *defined for a set of functions* $\{\mathcal{F}_{\lambda}\}_{\lambda \in N} = \{f : \mathcal{M}_{\lambda} \rightarrow Y_{\lambda}\}$ *is a tuple of* PPT *algorithms* $(\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$*:*

- $(\mathsf{msk}, \mathsf{mpk}) \leftarrow_{\$} \mathsf{FE.Setup}(1^{\lambda})$ *: for the unary representation of the security parameter* λ*, a pair of master secret/public keys is released.*
- $\mathsf{sk}_f \leftarrow_{\$} \mathsf{FE.KGen}(\mathsf{msk}, f)$*: for the master secret key and a function* f *taken as input, the key-derivation method releases a corresponding* sk_f*.*
- $\mathsf{CT} \leftarrow_{\$} \mathsf{FE.Enc}(\mathsf{mpk}, \mathsf{m})$*: the encryption method releases a plaintext* m *corresponding to* mpk*.*
- $\mathsf{FE.Dec}(\mathsf{CT}, \mathsf{sk}_f)$*: for a ciphertext* CT *and a functional key* sk_f*, either a valid message* $f(\mathsf{m})$ *is released or a an error symbol* $\perp$*, if decryption fails.*

We say that FE *satisfies correctness if for all* $f : \mathcal{M}_{\lambda} \rightarrow Y_{\lambda}$*:*

$$\Pr\left[y = f(M) \;\middle|\; \begin{array}{l} (\mathsf{msk}, \mathsf{mpk}) \leftarrow_{\$} \mathsf{FE.Setup}(1^{\lambda}) \wedge \mathsf{sk}_f \leftarrow_{\$} \mathsf{FE.KGen}(\mathsf{msk}, f) \wedge \\ \mathsf{CT} \leftarrow_{\$} \mathsf{FE.Enc}(\mathsf{mpk}, \mathsf{m}) \wedge y \leftarrow \mathsf{FE.Dec}(\mathsf{CT}, \mathsf{sk}_f) \end{array} \right] = 1 - \mathrm{NEGL}(\lambda) \, .$$

We call FE *semantically secure if there exists a stateful* PPT *simulator* $\mathcal{S}$ *s.t. for any* PPT *adversary* $\mathcal{A}$*,* $\mathbf{Adv}^{\mathsf{IND\text{-}FE\text{-}CPA}}_{\mathcal{A},\mathsf{FE}}(\lambda) := \left| \Pr[\mathsf{IND\text{-}FE\text{-}CPA}^{\mathcal{A}}_{\mathsf{FE}}(\lambda) = 1] - \frac{1}{2} \right|$ *is negligible; the game* $\mathsf{IND\text{-}FE\text{-}CPA}$ *is presented in Figure 2 (left).*

Definition 4 (Indistinguishability Obfuscation[27]). *A* PPT *algorithm* iO *is an indistinguishability obfuscator for a class* $\{\mathcal{C}_{\lambda}\}_{\lambda \in \mathbb{N}^{*}}$ *if the followings hold:*

- ***Correctness****:* $\Pr\left[\forall x \in \mathcal{D}, \mathcal{C}(x) = \overline{\mathbf{C}}(x) | \overline{\mathbf{C}} \leftarrow_{\$} \mathsf{iO}(\mathcal{C})\right] = 1$ *.*
- ***Indistinguishability****: for* $\mathcal{D}$ *the input domain of the circuits* C*, the following quantity is negligible:*

$$\left| \Pr\left[b = b' \;\middle|\; \begin{array}{l} \forall C_1, C_2 \in \{C\}_{\lambda} \wedge \forall x \in \mathcal{D} : C_1(x) = C_2(x) \wedge \\ b \leftarrow_{\$} \{0,1\} \;\wedge\; \overline{\mathbf{C}} \leftarrow_{\$} \mathsf{iO}(\mathcal{C}_b) \wedge b' \leftarrow_{\$} \mathcal{A}(1^{\lambda}, \overline{\mathbf{C}}, \mathcal{C}_0, \mathcal{C}_1) \end{array} \right] - \frac{1}{2} \right|$$

2.2 Direct ADPs from Randomized Encodings

Definition 5 (Affine Determinant Programs[8]). *Given the input length* n *and dimension* m*, an affine determinant program over* $\mathbb{F}_p$ *has two algorithms:*

$\mathsf{Prog} \leftarrow_{\$} \mathsf{ADP.Setup}(1^{\lambda}, \mathcal{C})$*: the* Setup *is a randomized algorithm such that given a circuit description* $\mathcal{C}$ *of some function* $f : \{0,1\}^{n} \rightarrow \{0,1\}$*, outputs a set of* $n+1$ *matrices as the ADP program:* $\mathsf{Prog} := (\mathbf{T}_0, \mathbf{T}_1, \ldots, \mathbf{T}_n) \in \mathbb{F}_p^{m \times m}$ *.*
$b \leftarrow \mathsf{ADP.Eval}(\mathsf{Prog}, \mathsf{inp})$*: is a deterministic procedure, that given the program* Prog *and some input* m*, returns a binary value* b*, defined as:*

$$b := \det\left(\mathbf{T}_0 + \sum_{i=1}^{n} \mathsf{inp}_i \cdot \mathbf{T}_i \right) .$$

$b \leftarrow_\$ \{0,1\}$
$\mathtt{L} \leftarrow \emptyset$
$(\mathsf{m}_0, \mathsf{m}_1, \text{state}) \leftarrow_\$ \mathscr{A}^{\mathsf{KGenO}_{\mathsf{msk}}(\cdot), \mathsf{FE.EncO}_{\mathsf{msk}}(\cdot)}(1^\lambda)$
$\mathsf{CT}^* \leftarrow_\$ \mathsf{Enc}(\mathsf{msk}, \mathsf{m}_b)$
$b' \leftarrow_\$ \mathscr{A}^{\mathsf{KGenO}_{\mathsf{msk}}(\cdot), \mathsf{EncO}_{\mathsf{msk}}(\cdot)}(1^\lambda; \text{state})$
if $\exists f \in \mathtt{L}$ s.t. $f(\mathsf{m}_0) \neq f(\mathsf{m}_1)$:
  return 0
return $b = b'$

Proc. $\mathsf{KGenO}_{\mathsf{msk}}(f)$:
$\mathtt{L} \leftarrow \mathtt{L} \cup \{f\}$
$\mathsf{sk}_f \leftarrow_\$ \mathsf{FE.KGen}(\mathsf{msk}, f)$
return sk_f

$\mathsf{PRF}^{\mathscr{A}}_{\mathsf{PRF}}(\lambda)$:
$b \leftarrow_\$ \{0,1\}; \mathtt{L} \leftarrow \emptyset$
$K \leftarrow_\$ \mathsf{Setup}(1^\lambda)$
$b' \leftarrow_\$ \mathscr{A}^{\mathsf{Prf}(\cdot)}(1^\lambda)$
return $(b' \stackrel{?}{=} b)$

Proc. $\mathsf{Prf}(\mathsf{m})$:
if $\mathsf{m} \in \mathtt{L}$ **then return** $\mathtt{L}[\mathsf{m}]$
$Y \leftarrow \mathsf{PRF}(K, \mathsf{m})$
if $b = 0$ **then** $Y \leftarrow_\$ \{0,1\}^{|Y|}$
$\mathtt{L} \leftarrow \mathtt{L} \cup \{(\mathsf{m}, Y)\}$
return Y

Fig. 2. Security for pseudorandom functions (right), as well as FE security (left).

Correctness: $\forall \mathsf{m} \in \{0,1\}^n$, $\Pr\left[\mathscr{C}(\mathsf{m}) = \mathsf{Prog}(\mathsf{m}) | \mathsf{Prog} \leftarrow_\$ \mathsf{ADP.Setup}(\lambda, \mathscr{C})\right] = 1$
Security: *We say that an* ADP *is* IND-ADP *secure with respect to a class of circuits* $\mathscr{C}_\lambda$*, if* $\forall (\mathscr{C}_1, \mathscr{C}_2) \in \mathscr{C}_\lambda \times \mathscr{C}_\lambda$ *such that* $\forall \mathsf{m} \in \{0,1\}^\lambda \mathscr{C}_1(\mathsf{m}) = \mathscr{C}_2(\mathsf{m})$*, the following quantity is negligible:*

$$\left| \Pr\left[b \leftarrow_\$ \mathscr{A}(1^\lambda, \mathsf{Prog}) \;\middle|\; b \leftarrow_\$ \{0,1\} \wedge \mathsf{Prog} \leftarrow_\$ \mathsf{ADP.Setup}(1^\lambda, \mathscr{C}_b) \right] - \frac{1}{2} \right|$$

Randomized Encodings through Branching Programs. This part covers the implementation of randomized encodings [23] from branching programs, and recaps the description from a work in submission[3].

A BP puts forth a method to (sequentially) compute an abstract function (represented as a circuit). Constructing branching programs for circuits is straightforward and largely described in literature [26]. In this work we use only single-bit output functions, but we can view non-boolean functions as concatenations of boolean ones. The branching program itself is (close to) a digraph such that each node has two potential outgoing arcs (except for terminal nodes). Each bit within input is linked to a corresponding node and based on input's value, one out of two corresponding arcs is followed until a terminal node – 0 or 1 – is visited. The value of the terminal node is the function's output for the provided input (that determined a path in digraph). Barrington provides a proof in [6] that the depth of the circuit representation of f is linked to the size of the corresponding branching program.

Set $\mathbf{G}_\mathsf{m}$ to be the adjacency matrix for the branching program corresponding to some $f : \{0,1\}^{|\mathsf{m}|} \rightarrow \{0,1\}$. Set the entries in the main diagonal to 1. Note that every row has at most one extra 1 apart from the 1 occurring on the main diagonal. Let $\overline{\mathbf{G}}_\mathsf{m}$ stand for the matrix post-processed by eliminating the first column and the last row within $\mathbf{G}_\mathsf{m}$. In [22] it is shown that $f(\mathsf{m}) = \mathbf{det}(\overline{\mathbf{G}}_\mathsf{m})$. Furthermore, matrices $\mathbf{R}_l$ and $\mathbf{R}_r$ having a special form exist, and the following relation holds:

$$\mathbf{R}_l \cdot \overline{\mathbf{G}}_\mathsf{m} \cdot \mathbf{R}_r = \left(\begin{array}{c|c} \vec{\mathbf{0}} & f(\mathsf{m}) \\ \hline -\mathbf{I} & \vec{\mathbf{0}} \end{array}\right) = \begin{pmatrix} 0 & 0 & \dots & 0 & f(\mathsf{m}) \\ -1 & 0 & \dots & 0 & 0 \\ 0 & -1 & \dots & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \dots & -1 & 0 \end{pmatrix} = \overline{\mathbf{G}}_{f(\mathsf{m})} \in \mathbb{F}_p^{m \times m}$$

This representation of $f(\mathsf{m})$, as a product of two matrices $\mathbf{R}_l$ and $\mathbf{R}_r$, is advantageous when considering randomized encodings' simulation security. Notable, the value $f(\mathsf{m})$ is given to the simulator while it can simulate a product of (1) either full-ranked matrices or (2) of rank $m-1$ matrices. Thus, such a representation is a direct, natural randomized encoding. During the decoding phase of the randomized encoding, the determinant of $\mathbf{R}_l \cdot \overline{\mathbf{G}}_\mathsf{m} \cdot \mathbf{R}_r$ is obtained, allowing to get $f(\mathsf{m})$. Correctness follows given that both $\mathbf{R}_l, \mathbf{R}_r$ are non-singular.

We consider $\mathbf{R}_r, \mathbf{R}_l \in \mathbb{F}_p^{m\times m}$ having the subsequent pattern:

$$\mathbf{R}_l = \begin{pmatrix} 1 & \$ & \$ & \dots & \$ & \$ \\ 0 & 1 & \$ & \dots & \$ & \$ \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & \$ \\ 0 & 0 & 0 & \dots & 0 & 1 \end{pmatrix}, \mathbf{R}_r = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 & \$ \\ 0 & 1 & 0 & \dots & 0 & \$ \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & \$ \\ 0 & 0 & 0 & \dots & 0 & 1 \end{pmatrix}.$$

To generalize the previous observation, one can use different distributions for $\mathbf{R}_l, \mathbf{R}_r$. Put differently, consider $\mathbf{L} \in \mathbb{F}_p^{m\times m}$ and $\mathbf{R} \in \mathbb{F}_p^{m\times m}$ being sampled uniformly at random over the space of non-singular matrices. We can write them as: $\mathbf{R} \leftarrow \mathbf{R}_r \cdot \mathbf{R}'$ and $\mathbf{L} \leftarrow \mathbf{L}' \cdot \mathbf{R}_l$. Note that:

$$\mathbf{L} \cdot \overline{\mathbf{G}}_{\mathsf{m}} \cdot \mathbf{R} = (\mathbf{L}' \cdot \mathbf{R}_l) \cdot \overline{\mathbf{G}}_{\mathsf{m}} \cdot (\mathbf{R}_r \cdot \mathbf{R}) = \mathbf{L}' \cdot (\mathbf{R}_l \cdot \overline{\mathbf{G}}_{\mathsf{m}} \cdot \mathbf{R}_r) \cdot \mathbf{R} = \mathbf{L}' \cdot \overline{\mathbf{G}}_{f(\mathsf{m})} \cdot \mathbf{R}'$$

Since the matrices $\mathbf{L}'$ and $\mathbf{R}'$ have full-rank, $\det\left(\mathbf{L} \cdot \overline{\mathbf{G}}_{\mathsf{m}} \cdot \mathbf{R}\right) = \det\left(\overline{\mathbf{G}}_{\mathsf{m}}\right) = f(\mathsf{m})$. Also, observe that all entries in the resulting $m \times m$ matrix

$$\mathbf{T}_{\mathsf{m}} \leftarrow \mathbf{L} \cdot \overline{\mathbf{G}}_{\mathsf{m}} \cdot \mathbf{R},$$

can be written as a sum of degree-three monomials. [22] shows that when decomposing every entry within $\mathbf{T}_{i,j}$ in monomials, no monomial depends on more than one input bit of m. Moreover, each monomial has one component occurring stemming from $\mathbf{L}$ and $\mathbf{R}$.

Augmenting NC^1 BPs for Keyed Functions The next part (Section 3) describes how ADPs can be used to instantiate iO. We will not use ADPs directly as described before. We will first apply transformations at the branching program level resulting in an "augmented" branching program, taken from [3] and described herein. The nuts and bolts of "augmented ADP" are used only in the indistinguishability proof for ADPs (taken from [3] and provided in Section 6), while Section 3 needs only the interface of ADPs – so they can regarded as black-box objects enjoying indistinguishability (Definition 5).

Nimvs and Augmenting BPs. [3] puts forth a method to *augment* a branching program with a set of intermediate nodes. This step is done without changing the underlying function's behaviour. The reason for doing so is to isolate the "sensitive" variables – called *non-input mutable variables* or nimv. For the problem of building iO, nimv should be imagined as pPRFs keys, m, ρ or c, or any other variables on which ADPs may differ while preserving their functionality. Hiding such variables is instrumental in constructing iO.

Consider the branching program BP that implements a bivariate function $f(\mathsf{nimv}, \mathsf{m})$ that is represented as a circuit by $\mathscr{C}(\mathsf{nimv}||\mathsf{m})$; BP's directed acyclic graph representation has two complementary sets of nodes: the first contains the vertices depending on nimv, the second contains vertices linked to input (the message m). Any other non-terminal vertex – if any – that is not linked to input is added to the first set. Moreover, the nimvs are hardcoded (i.e. fixed) to some values. This hardcoding step implies that all nodes depending on nimv in BP are assigned a binary value.

The augmented branching program is simply a BP that has an additional ensemble of nodes injected into its original graph. We add *auxiliary* and *flare* nodes, intuition being discussed in Section 1.

Auxiliary nodes. The first transform proposed in [3] concerns auxiliary nodes. Given v a vertex settled by bit in some nimv variable's binary representation, and let u stand for another vertex s.t. an arc $v \rightarrow u$ exists in the graph representation of the BP. The digraph's structure is changed (augmented) by injecting an auxiliary node α between v and u, such that v is no longer directly connected to u. The path between the two variables becomes $v \rightarrow \alpha \rightarrow u$.

Definition 6 (Augmented branching programs for circuits in NC^1 with auxiliary nodes [3]). *Let BP be the branching program corresponding to some circuit $\mathscr{C}_{nimv} \in \mathsf{NC}^1$ that embeds nimv. Let $\mathscr{V}$ denote the set of vertices settled by nimv. For each vertex $v \in \mathscr{V}$ let u be a vertex such that there exists an arc from v to u. A branching program augmented with auxiliary nodes ABP is defined by extending the BP graph and introducing an intermediate vertex α on the path between any node v depending on nimv and any child vertex u.*

It can be noted that using auxiliary nodes to augment a branching program conserves its correctness. Over $\mathbb{F}_2$ computing the determinant (in order to evaluate the ADP) is in fact a sum of $m!$ permutations. Given that a path originating in the start vertex and ending in the terminal node 1 exists, it must be the case that this path as well as the 1s placed on the second diagonal will force one of the $m!$ sums appearing in the the explicit determinant formulation to be 1.

Related to the size of the augmented branching program, [3] notes that the size of ABP is upper bounded by $3 \times |\mathsf{BP}|$. This happens because every nimv-dependent node injects two other nodes, the number of nodes being at most triple.

The primary advantage offered by auxiliary nodes is a way to decouple the nimv dependent rows (or columns) appearing within the randomizing matrices $\mathbf{R}$ (or $\mathbf{L}$), from the other nodes. To see this, when the post-processed augmented matrix $\overline{\mathbf{G}}$ (obtained from ABP) gets multiplied with $\mathbf{R}$, the rows of $\mathbf{R}$ triggered by nimv-depending variables are isolated from the rows within $\mathbf{R}$ that depend on input inp. Somehow similarly, the columns of $\mathbf{L}$ can be partitioned in three disjoint sets: columns that are triggered by nimv, the inp or the auxiliary variables (there is an *asymmetry* to the splitting of rows in the randomizing matrix $\mathbf{R}$ – we only split its lines in two disjoint sets).

Flare vertices. The second step introduced in [3] is to inject *flare* vertices. Their purpose is to handle node ordering constraints on each of BP's paths. [3] considers the following problem: (1) take BP's structure augmented with auxiliary vertices; (2) choose any vertex that depends on nimvs; (3) choose any path that starts in the nimv-dependent vertex from step (2); (4) enforce that on the path chosen in (3), no vertices that depend on inp-dependent vertices appear "before" vertices that depend on nimvs. Equivalently, on any local path, all low-indexed vertices must be linked to nimvs (similarly for auxiliary nodes) and all high-indexed nodes must be linked to inp dependent nodes. This is done by injecting artificial vertices that redirect to the terminal node 1, whenever the condition stated before above fails. To restore the "normal evaluation mode", we add complementary arcs when summing up the matrix $\overline{\mathbf{G}}_1$ (that corresponds to the first input bit) to the base matrix $\overline{\mathbf{G}}_0$. This observation is made precise in what follows.

Take any path within the BP, and consider 3 non-empty lists of vertices $\mathtt{L}_1$, $\mathtt{L}_2$, $\mathtt{L}_3$. $\mathtt{L}_1$ includes only vertices linked to nimvs or auxiliary vertices, $\mathtt{L}_2$ includes only vertices linked to inputs, and $\mathtt{L}_3$ includes only vertices linked to nimvs or auxiliary vertices like $\mathtt{L}_1$. Assume that there is an arc between any two pair of neighbour vertices occurring in $\mathtt{L}_1$ (resp. $\mathtt{L}_2$ and $\mathtt{L}_3$); furthermore assume that arc between the last vertex in $\mathtt{L}_1$ and the first vertex in $\mathtt{L}_2$ exists; assume that an arc between the last vertex in $\mathtt{L}_2$ and the first vertex in $\mathtt{L}_3$ (thus a "path") exists. [3] introduces a natural ordering relation of vertices within BP: $v_i \prec v_j$ holds iff $i < j$. This ordering relationship is extended to paths and such that $\mathtt{L}_1 \prec \mathtt{L}_2$ iff $\max\{\mathsf{index}(v) : v \in \mathtt{L}_1\} < \min\{\mathsf{index}(v) : v \in \mathtt{L}_2\}$. According to the ordering defined above, every vertex in $\mathtt{L}_1$ has a lower index than any vertex in $\mathtt{L}_2$, and this transitive relations is kept amongst $\mathtt{L}_2$ and $\mathtt{L}_3$. [3] "decouples" vertices from $\mathtt{L}_2$ from the vertices in $\mathtt{L}_3$. They introduce flares, after each "ending" vertex from $\mathtt{L}_2$. An "ending" vertex $v \in \mathtt{L}_2$ is a vertex that has an arc to some $v' \in \mathtt{L}_3$. Flare nodes fl^1, fl^0 depending on input bit in position 1 are introduced s.t. the following arcs exist: $v \to fl^0 \to 1$ and $v \to fl^1 \to v'$. In layman terms, fl acts as an "electrical" switch: whenever the first bit in inp is 1, fl^1 enables the normal BP evaluation flow; whenever the first bit in inp is 0, fl^0 redirects the flow to the terminal node 1 – thus skipping the real function evaluation.

Definition 7 (Augmented branching programs for circuits in NC^1 with flare nodes [3]). *Let* BP *be the branching program corresponding to some circuit $\mathscr{C}_{nimv} \in \mathsf{NC}^1$ that embeds nimv. Let $\prec$ denote a node ordering relation. Let L_1, L_3 denote lists of increasingly ordered vertices settled by nimv and auxiliary nodes. Let L_2 be an ordered list of nodes that are settled by inputs. Let $L_1 \prec L_2 \prec L_3$ such that the relation is applied vertex-wise. For any node v in L_2 such that $v \to v'$ and $v' \notin L_2$ introduce a node fl^0 such that $v \to fl^0 \to 1$. Introduce $fl^1 \to v'$. Let the node fl^b be triggered by the value b of the first input bit.*

Finally, the first input bit will be linked to a dummy node. Therefore, the arity of ADP is artificially increase by 1. When the ADP is evaluated such that the first, artificial, input bit is 0 the output is 1; when changing the first input bit to 1 we use the normal evaluation mode.

Related to the size, similarly to auxiliary nodes, it can be noted that adding flares preserves the polynomial size of the BP, the increase can be loosely upper bounded by twice the size of the ABP already augmented with auxiliary nodes.

3 Our New Indistinguishability Obfuscator

Definition 8. *Let $\mathscr{C}_{d,g} : \{0,1\}^n \to \{0,1\}^t$ stand for a class of circuits having depth d, size g, input length n and output length t. Let $\mathsf{Uc}^{d,g}_{d',g'}$ stand for a universal circuit supporting the evaluation of circuits of depth $\leq d$ and gates $\leq g$ and itself having depth d' and g' gates. Let FE denote a functional encryption scheme for a class of circuits $\mathscr{C}_{d',g'} : \{0,1\}^{n+\mathsf{poly}(d',g')} \to \{0,1\}^t$ having depth d' and gates g'. Let ℓ stand for the size of a ciphertext of FE. Assume the encryption procedure of FE can be described by an NC^1 circuit. Let pPRF stand for a puncturable pseudorandom function keyed by k and having its length of output matching the length of FE's encryption randomness term. Let ADP denote an affine determinant program supporting circuits matching the depth and width of FE's encryption procedure. The following construction represents an iO-obfuscator for circuits in class $\mathscr{C}_{d,g}$.*

$\underline{\mathsf{iO.Setup}(1^\lambda, \mathscr{C})}$**:** *Let m stand for the representation of $\mathscr{C}$ as a binary string. Consider the decomposition of an FE ciphertext into ℓ components. For every $j \in [\ell]$, consider the circuit described in Figure 3.*

$\mathscr{C}^j_{\mathsf{mpk},\mathsf{k},\mathsf{m},\perp,\rho,c}(b||i)$:
if $b = 0$: return 0
if $i < \rho$: return $\mathsf{FE}_j.\mathsf{Enc}\left(\mathsf{mpk}, \mathsf{m}||i; \mathsf{pPRF}(\mathsf{k}, 1||i)\right)$
if $i = \rho$: return c
if $i > \rho$: return $\mathsf{FE}_j.\mathsf{Enc}\left(\mathsf{mpk}, \perp||i; \mathsf{pPRF}(\mathsf{k}, 1||i)\right)$

Fig. 3. The circuit outputs (bits of) a functional encryption (FE) ciphertext. ρ is a limit set to $2^n - 1$. The main input range is $\{0,1\}^n$, artificially extended by 1.

1. *Sample* $(\mathsf{msk}, \mathsf{mpk}) \leftarrow_\$ \mathsf{FE.Setup}(1^{g'}, 1^{d'})$. *Set* $\rho \leftarrow 2^n - 1$.
2. *Compute* $c \leftarrow_\$ \mathsf{FE.Enc}(\mathsf{mpk}, \mathsf{m}||2^n - 1)$.
3. *Sample a* pPRF *key* $\mathsf{k}_{\mathsf{pPRF}}$ *and puncture it in a random point* $0||\$$.
4. *For each* j *in* $[\ell]$, *run* $\mathsf{ADP.Setup}\left(1^\lambda, \mathscr{C}^j_{\mathsf{mpk},\mathsf{k}_{\mathsf{pPRF}},\mathsf{m},\perp,\rho,c}\right)$ *and obtain* ADP^j.
5. *Run* $\mathsf{FE.KGen}(\mathsf{msk}, \mathsf{Uc})$ *and obtain* $\mathsf{sk}_{\mathsf{Uc}}$.
6. *Return* $\left(\mathsf{sk}_{\mathsf{Uc}}, \{\mathsf{ADP}^j\}_{j\in[\ell]}\right)$.

$\underline{\mathsf{iO.Eval}(\mathsf{inp})}$**:**

1. *For each* j *in* $[\ell]$, *evaluate* $\mathsf{ADP}^j.\mathsf{Eval}(\mathsf{inp})$ *and obtain* CT^j.
2. *Run* $\mathsf{FE.Dec}(\mathsf{sk}_{\mathsf{Uc}}, \mathsf{CT}^1||\ldots||\mathsf{CT}^\ell)$.

Proposition 1 (Correctness). *The obfuscator in Definition 8 is correct.*

Proof (Proposition 1).

$$\begin{aligned}
\mathsf{iO.Eval}(\mathsf{inp}) &= \mathsf{FE.Dec}\left(\mathsf{sk}_{\mathsf{Uc}}, \mathsf{ADP}^1.\mathsf{Eval}(\mathsf{inp})||\ldots||\mathsf{ADP}^\ell.\mathsf{Eval}(\mathsf{inp}) = ||_{i=1}^{\ell}\mathsf{ADP}^i.\mathsf{Eval}(\mathsf{inp})\right) \\
&= \mathsf{FE.Dec}\left(\mathsf{sk}_{\mathsf{Uc}}, ||_{i=1}^{\ell}\mathsf{ADP}^i.\mathsf{Eval}(\mathsf{FE.Enc}(\mathsf{mpk}, \mathscr{C}||\mathsf{inp}; \mathsf{pPRF}(\mathsf{k}, \mathsf{inp})))\right) \\
&= \mathsf{FE.Dec}\left(\mathsf{sk}_{\mathsf{Uc}}, \mathsf{FE.Enc}(\mathsf{mpk}, \mathscr{C}||\mathsf{inp}; \mathsf{pPRF}(\mathsf{k}, \mathsf{inp}))\right) \\
&= \mathsf{Uc}(\mathscr{C}||\mathsf{inp}) = \mathscr{C}(\mathsf{inp}) = f(\mathsf{inp}) .
\end{aligned}$$

4 Security of Our iO FE Scheme

The proof is akin to the one published in [27]. Its structure can be summarized as follows: we hybridize over every single input inp taken over $\{0,1\}^n$ (which incurs an exponential security loss in n). For each input, we will: i) use

the indistinguishability of ADP to switch the pPRF key to one[9] punctured in 1||inp; ii) then, based on pPRF indistinguishability, we switch the pPRF's value in the punctured point to true randomness[10]; iii) then based on FE's indistinguishability we switch to an FE ciphertext corresponding to $\mathsf{m}_1||\mathsf{inp}$, where m_1 is the binary representation of $\mathscr{C}_1$; iv) we then revert the changes, and switch the random coins back to the ones generated by the pPRF under a key punctured in inp; v) finally, we use the ADP's indistinguishability to switch back the punctured key and also decrease the limit ρ.

Theorem 3 (iO-Security). *The obfuscator presented in Definition 8 reaches indistinguishability, as per Definition 4.*

Proof (Theorem 3). The proof follows through a standard hybrid argument. We present the hybrid games and then provide the reductions that justify the transitions between these distributions.

Game_0**:** the real game corresponding to $\mathscr{C}_0$ encoded as m_0. Identical to $\mathsf{Game}_{1,0}$.
$\mathsf{Game}^0_{i,0}$**:** identical to $\mathsf{Game}_{i,0}$. In forthcoming games, i indexes elements of set $[2^n-1]$.
$\mathsf{Game}^j_{i,0}$**:** for all $j \in [\ell]$ using ADP's indistingishability, replace the pPRF key k (punctured in 0||\$) with a key punctured in $1||(2^n-i)$.
$\mathsf{Game}_{i,1}$**:** based on pPRF's security in the punctured point, replace the actual pPRF's value used to compute FE's ciphertext c in the challenge point with an actual true randomly sampled value r;
$\mathsf{Game}_{i,2}$**:** based on FE's indistinguishability, replace the FE ciphertext c corresponding to the challenge point with a ciphertext encrypting $\mathsf{m}_1 := \mathscr{C}_1$.
$\mathsf{Game}_{i,3}$**:** is the inverse of $\mathsf{Game}_{i,1}$.
$\mathsf{Game}^j_{i,4}$**:** is the inverse of $\mathsf{Game}_{i,0}$ (the punctured key is replaced by k, and the limit ρ is decreased to $2^n-(i+1)$.
$\mathsf{Game}^\ell_{2^n-1,4}$**:** is the last game in the sequence, where the ciphertext corresponds to $\mathsf{m}_1 := \mathscr{C}_1$ and the limit ρ is 0.
$\mathsf{Game}_{\mathbf{final}}$**:** based on ADP's indistinguishability, we switch the places between m_0 and m_1 and replace ρ's value of 0 with 2^n-1.

We prove the transitions between each pair of consecutive hybrids.
$\mathsf{Game}^{j-1}_{i,0} \to \mathsf{Game}^j_{i,0}$: for any $j \in [\ell]$, in $\mathsf{Game}^{j-1}_{i,0}$ the circuit will compute the j^{th}-bit of the FE ciphertexts using k, a punctured key punctured in 0||\$; while in $\mathsf{Game}^j_{i,0}$, a punctured key punctured in $1||(2^n-i)$ is going to be used. The two circuits are equivalent due to the functional preservation under punctured keys of a puncturable PRF as well as due to the fact that in the challenge point, the ciphertext is computed using the normal key. Formally, the reduction algorithm $\mathscr{B}$ crafts the two functionally-equivalent circuits, sends them to the ADP challenger, and receives an ADP-obfuscated version of one of them. The received version stands for the j^{th} ADP that is sent to the adversary $\mathscr{A}_1$. The adversary $\mathscr{A}_1$ impersonates an iO distinguisher, and returns its guess b (that is next returned by $\mathscr{B}$ to the security experiment). Clearly, the simulation is perfect and the winning advantage against the hybrid transition is identical to the advantage against ADP's indistinguishability.
$\mathsf{Game}^\ell_{i,0} \to \mathsf{Game}_{i,1}$: in this setting, the punctured key used by the ℓ circuits is punctured in the input point $1||(2^n-i)$. The hardcoded ciphertext corresponding to $\rho = 2^n-i$ cannot be computed without the challenge value received from the pPRF game. Our reduction $\mathscr{B}$ takes from the pPRF game: i) a key punctured in $1||(2^n-i)$ and ii) a value corresponding to point $1||(2^n-i)$ which is either sampled uniformly at random or generated using the normal pPRF key. The reduction $\mathscr{B}$ uses challenge value to build the FE ciphertext in ρ and the punctured key in $1||(2^n-i)$ to further build all ADPs, which is perfectly possible. The set of ADPs as well as the FE functional key are then sent to the adversary $\mathscr{A}$ that outputs a guess b, which is $\mathscr{B}$'s guess. As for the previous case, the reduction is tight. Note that in adversary's view, both iO obfuscators are functionally equivalent, as decrypting FE ciphertexts generated under different random coins will provide the same result.
$\mathsf{Game}_{i,1} \to \mathsf{Game}_{i,2}$: we use the indistinguishability of the functional encryption to switch the encoded message, which should correspond to $\mathscr{C}_1$ for the challenge input. The reduction $\mathscr{B}$ will receive the mpk, $\mathsf{sk}_{\mathsf{Uc}}$, will construct

[9] This happens once for all of the $[\ell]$ circuits.
[10] Note that the output corresponding to the challenge inp is an FE ciphertext encrypting $\mathscr{C}_0||\mathsf{inp}$.

the challenge ciphertext corresponding to $\rho = 2^n - i$. It will sample on its own the pPRF keys and build the ADPs. The resulting obfuscator represents the input to the adversary. Clearly, if the adversary distinguishes with non-negligible probability, the same will do $\mathscr{B}$.

$\mathsf{Game}^{\ell}_{i,2} \rightarrow \mathsf{Game}_{i,3}$: is virtually the inverse of the transition between $\mathsf{Game}^{\ell}_{i,0}$ and $\mathsf{Game}_{i,1}$.

$\mathsf{Game}_{i,3} \rightarrow \mathsf{Game}^{0}_{i,4}$: these games are identical.

$\mathsf{Game}^{j-1}_{i,4} \rightarrow \mathsf{Game}^{j}_{i,4}$: is almost the inverse of the transition between $\mathsf{Game}^{j-1}_{i,0}$ and $\mathsf{Game}^{j}_{i,0}$, up to the difference that the limit ρ is decreased by 1 and a ciphertext corresponding to m_0 is computed for this position $\rho - 1$. Clearly the two settings are identical, as for inputs $(\rho - 1, \rho)$ the first ADP outputs the ciphertexts corresponding to $(\mathsf{m}_0, \mathsf{m}_1)$, while the ADPs in the other setting outputs ciphertexts corresponding exactly to the same pair $(\mathsf{m}_0, \mathsf{m}_1)$.

In the last but one game, the circuit will have $\rho = 0$. We do an extra game hop and will output only ciphertexts corresponding to m_1 (and thus $\mathscr{C}_1$). We add the extra hybrid that switches ρ back to $2^n - 1$ and inverses m_0 and m_1. Clearly this follows from ADP indistinguishability.

We apply the union bound and conclude that the advantage of any PPT adversary $\mathscr{A} = (\mathscr{A}_1, \mathscr{A}_2, \mathscr{A}_3)$ is upper bounded by: $\mathbf{Adv}^{\mathsf{iO}}_{\mathscr{A},\mathsf{cFE}}(\lambda) \leq (2^n - 1) \cdot \Big((2\ell + 1) \cdot \mathbf{Adv}^{\mathsf{IND\text{-}ADP}}_{\mathscr{A}_1,\mathsf{ADP}}(\lambda) + \mathbf{Adv}^{\mathsf{s\text{-}IND\text{-}FE\text{-}CPA}}_{\mathscr{A}_2,\mathsf{FE}}(\lambda) + 2 \cdot \mathbf{Adv}^{\mathsf{puncture}}_{\mathscr{A}_3,\mathsf{pPRF}}(\lambda) \Big) + \mathbf{Adv}^{\mathsf{IND\text{-}ADP}}_{\mathscr{A}_1,\mathsf{ADP}}(\lambda)$. This completes the proof. □

4.1 Instantiation from Learning With Errors

Somehow expected, the primitives used herein are realizable from sub-exponentially secure Learning with Errors assumption. Universal circuits able to evaluate circuits of the same depth exists unconditionally. Puncturable PRFs can be realized from LWE, by referring to a work of Canetti and Chen that builds single-key constrained PRFs[14] (and thus puncturable PRFs) where the evaluation is in NC^1.

Sub-exponentially secure LWE suffices to instantiate succinct single-key functional encryption [19].

5 Sufficiency Conditions for IND-Secure ADP

This part is included as appendix from [3], in order to be able to verify the IND-ADP security of the affine determinant programs. In terms of novelty, this part is devoid of any content.

It is by now clear the link between the security of affine determinant programs and that of indistinguishability obfuscators[5]. Namely, an ADP reaching IND-ADP security provides a direct iO.

We turn our attention on strengthening the security at the expense of shrinking the supported class of circuits. As stated in introduction, we aim for perfect security for the classes of circuits comprising the one put forth in Definition 4.

Definition 9 (Perfectly-Secure ADPs). *Let $\mathscr{C}_0$ and $\mathscr{C}_1$ denote two functionally equivalent, but internally different circuits. $\mathscr{C}_0$ and $\mathscr{C}_1$ admit* ADP *implementations that are perfectly-secure if for any randomness term R_0, there exists a uniquely determined value R_1 such that:*

$$\mathsf{ADP.Setup}(1^{\lambda}, \mathscr{C}_0; R_0) = \mathsf{ADP.Setup}(1^{\lambda}, \mathscr{C}_1; R_1) \, .$$

Stated differently, we can obtain the *same* implementation for two equivalent functions, using two different sets of randomness terms. We can relax the definition above in a statistical sense, requiring the previous condition to hold in front of a statistical adversary.

Definition 10 (Statistically-Secure PS-ADPs). *We say that two different circuits $\mathscr{C}_0$ and $\mathscr{C}_1$ admit* ADP *implementations that are colliding-secure if for any R_0, there exists a randomness term R_1 such that:*

$$\mathsf{ADP.Setup}(1^{\lambda}, \mathscr{C}_0; R_0) \approx_s \mathsf{ADP.Setup}(1^{\lambda}, \mathscr{C}_1; R_1) \, .$$

Proposition 2 (PS-ADP $\Longrightarrow$ IND-ADP)**.** *Let $\mathscr{C}_0$ and $\mathscr{C}_1$ be two circuits that admit an* PS-ADP *secure implementation. Then, they also admit an* IND-ADP*-secure implementation.*

Proof (Proposition 2). The IND-ADP secure implementation is identical to the PS-ADP implementation. By PS-ADP-security, the advantage of any adversary in distinguishing the two settings is 0. Hence, any adversary against the IND-ADP game will have advantage 0 in winning. □

5.1 Constraints on Classes of ADPs

Below, we provide a summary of a set of requirements for circuits in order to admit PS-ADP-secure implementation. As a caveat, we do not claim that the conditions put forth herein are also necessary. Stated differently, there may be other ways to obtain PS-ADP secure instantiations for various classes of functions and by bypassing the conditions stated below. Our approach is to provide a list of conditions, motivate them as compelling as possible, although for some the motivation stems from the proof, and make various remarks by connecting to known findings.

Condition 1: naturally, a primordial conditions is related to the depth of circuits. For circuits to admit polynomial-sized branching program, the class of supported circuits should be $\mathsf{L/poly}$. For brevity, we work with NC^1.

Condition 7 *Let $\mathfrak{C}_{d,|nimv|+(n+1)}$ stand a class of circuits of depth d with input length $|nimv|+(n+1)$. A condition for $\mathfrak{C}_{d,|nimv|+(n+1)}$ to admit an* PS-ADP*-secure implementation is $d \in O(\log_2(|nimv|+(n+1)))$.*

Condition 2: the main set of applications that are envisioned are built on multivariate functions. As mentioned in Section 2.2, the functions we deal with have inputs and nimv variables. Thus, we can always regard them as functions over "$\mathsf{nimv}||\mathsf{inp}$". Hence, the function that is modelled by some circuit can be described as: $f : \{0,1\}^{|\mathsf{nimv}|+n} \rightarrow \{0,1\}$. As a technicality, the input space of the circuit representation of f is doubled in order to cope with the dummy *flare* nodes.

Condition 8 *Let $\mathfrak{C}_{d,|nimv|+(n+1)}$ stand for a class of circuits of depth d and input length $|nimv|+(n+1)$. A condition for $\mathfrak{C}_{d,|nimv|+(n+1)}$ to admit an* PS-ADP*-secure implementation is that every $\mathscr{C} \in \mathfrak{C}_{d,|nimv|+(n+1)}$ implements a two input function $f : \{0,1\}^{|nimv|+n} \rightarrow \{0,1\}$.*

Condition 3: the proof given in Section 6 requires us to consider exclusively non-zero-constant functions. The main trigger behind this requirement is the need to use invertible matrices in the proof. Clearly, if the function would be 0 on all inputs, all the determinants evaluated via ADPs on all inputs will be 0, no candidate matrix will be invertible and the proof will not go through.

Condition 9 *Let $\mathfrak{C}_{d,|nimv|+(n+1)}$ stand for a class of circuits of depth d and input length $|nimv|+(n+1)$. A condition for $\mathfrak{C}_{d,|nimv|+(n+1)}$ to admit a* PS-ADP*-secure implementation is that: for every $\mathscr{C} \in \mathfrak{C}_{d,|nimv|+(n+1)}$ implementing some f, there exists a* PPT *algorithm $\mathscr{R}$ such that:*

$$\Pr\left[\mathscr{C}(|nimv|,\mathsf{inp}) = 1 | (|nimv|,\mathsf{inp}) \leftarrow \mathscr{R}(1^{\lambda},\mathscr{C})\right] > \frac{1}{\mathsf{poly}(|nimv|+(n+1))} . \tag{15}$$

This should be interpreted as: there exists an "efficient" PPT procedure that can find some set of nimv together with some input, such that $f(\mathsf{nimv},\mathsf{inp}) = 1$. We **emphasize** that $\mathscr{R}$ is **not** asked to sample uniformly at random the inp point, nor the variables nimv. We show below that there exists functions not fulfilling this property.

Proposition 3 (Separation for Condition 3). *Assume the existence of claw-free permutations. Let f_0 and f_1 be such permutations that are claw free. Define f to be 1 as long as $f_1(x) = f_2(y)$ and 0 otherwise, for some (x,y) representing the input to f. This function does not fulfil the requirement above.*

Proof. Finding a claw is difficult by definition of claw-free permutations. □

Remark 3 (Is statistical iO *possible?).* Goldwasser and Rothblum [20] show that achieving iO that is statistically secure is impossible for general function. Their result constructs an unsatisfiable SAT formula which is then obfuscated. The unsatisfiable formula corresponds to an all-zero function, which is captured by the condition above.

Condition 4: this condition asks for an efficient algorithm $\mathscr{R}$ that can generate a pair $(\mathsf{nimv},\mathsf{nimv}')$ such that $f(\mathsf{nimv},X) = f(\mathsf{nimv}',X)$.

Condition 10 *Let $\mathfrak{C}_{d,|nimv|+(n+1)}$ stand for a class of circuits of depth d and input length $|nimv|+(n+1)$. A condition for $\mathfrak{C}_{d,|nimv|+(n+1)}$ to admit a* PS-ADP*-secure implementation is that: for every $\mathscr{C} \in \mathfrak{C}_{d,|nimv|+(n+1)}$ modelling some f, there exists a* PPT *algorithm $\mathscr{R}$ such that $\forall\ \mathsf{inp} \in \mathscr{X}$,*

$$\Pr[nimv \neq nimv' \wedge \mathscr{C}(nimv, \mathsf{inp}) = \mathscr{C}(nimv', \mathsf{inp}) | (nimv, nimv') \leftarrow \mathscr{R}(1^{\lambda}, f)] > \frac{1}{\mathsf{poly}(|nimv| + (n+1)} \ .$$

Remark 4. As for the previous condition, we stress that $\mathscr{R}$ is not required to sample the keys uniformly at random.

Condition 5: The proof provided in Section 6 asks that the the first *line* in the modified adjacency matrix $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$ of the BP of f has of the form $(0,\ldots,0,1)$.
From a **high-level** point of view, this condition can be expressed as:

$$f(\mathsf{nimv}, 0||****) = 1 \ .$$

Condition 11 *Let $\mathfrak{C}_{d,|nimv|+(n+1)}$ denote a class of circuits of depth d and input length $|nimv|+(n+1)$. A condition for $\mathfrak{C}_{d,|nimv|+(n+1)}$ to admit a* PS-ADP*-secure implementation is that: for every $\mathscr{C} \in \mathfrak{C}_{d,|nimv|+(n+1)}$ modelling some f, there exists and is "easy to find" nimv such that:*

$$\mathscr{C}(nimv, b||\mathsf{inp}') := \begin{cases} 1\ , & \text{if } b = 0. \\ f(nimv, \mathsf{inp})\ , \text{if } b = 1 \ \text{ and } \ \forall \mathsf{inp}' \in \{0,1\}^n \end{cases}$$

Condition 6: The last condition is related to the ordering of nodes depending on nimv and inp, which means a condition on the topology of BP. Let $\mathscr{D}$ be the set of nodes having their values settled by nimvs or auxiliary nodes. Let $\mathscr{I}$ be the set of nodes depending on inputs (including flares). Let $\mathscr{P}$ denote a path originating in some $v \in \mathscr{D}$. The condition states that if $u, v \in \mathscr{P}$ and $u \in \mathscr{D}$ and $v \in \mathscr{I}$, then $u \prec v$. In rough terms, it must be the case the nodes depending on input –i.e. nodes in $\mathscr{I}$ – should occur "below" the ones in $\mathscr{D}$, on any path, where $\mathscr{I}$ includes flare nodes.

Condition 12 *Let $\mathfrak{C}_{d,|nimv|+(n+1)}$ stand for a class of circuits of depth d with input length $|nimv| + (n+1)$. A condition for $\mathfrak{C}_{d,|nimv|+(n+1)}$ to admit an* PS-ADP*-secure implementation is that the index of any vertex depending on input is greater than the index of any vertex depending on $|nimv|$, on any local path starting from some $|nimv|$-dependent vertex and including an input-dependent vertex.*

Definition 11 (Admissible Class for ADPs via BPs). *Let $\mathfrak{C}_{\lambda,d,|nimv|+n}$ denote a class of circuits. We call this class admissible if the requirements (1),(2),(3),(4),(5) and (6) stated above hold.*

5.2 The Claim for Perfectly Secure ADPs

Theorem 4 (IND-ADP **Security).** *Let $\mathfrak{C}_{d,|nimv|+(n+1)}$ be a class of ADP-admissible circuits as per Definition 11. Let $\mathscr{C}_{nimv} \in \mathfrak{C}_{d,|nimv|+(n+1)}$ be any circuit in $\mathfrak{C}_{d,|nimv|+(n+1)}$ having the nimv-dependent part of input fixed. Then, there exists an* ADP *that reaches* PS-ADP*-security with respect to $\mathscr{C}_{nimv}$.*

6 Proof of Theorem 4

In this part, we prove Theorem 4. We do this through a sequence of lemmata, while showing the necessity of each of the aforementioned conditions in our proof.

We show perfect security. Translated in layman's terms, we have to handle two affine determinant programs, obtained under different variables that we have to hide – nimv and nimv′ – such that:

$$\mathsf{ADP.Setup}(1^{\lambda}, \mathscr{C}_{\mathsf{nimv}}; \mathbf{L}, \mathbf{R}) = \mathsf{ADP.Setup}(1^{\lambda}, \mathscr{C}_{nimv'}; \mathbf{L}', \mathbf{R}') \tag{16}$$

This part is about the nuts and bolts of ADPs. Henceforth, we digress into Equation (16). An ADP for a circuit with input length α is nothing more than a set of $\alpha+1$ matrices, randomized by two randomization matrices, $\mathbf{L}$ and $\mathbf{R}$.

We remind that in our notation, the input length is itself $n+1$, n being the input length for a functions f, $n+1$ being the input length of its circuit representation that is going to be obfuscated. Thus, our ADPs will have each one base matrix and $n+1$ inp-dependent matrices. The evaluation passes only when the first matrix is added to the sum and the determinant applied on the resultant sum matrix. Explicitly, Equation (16) becomes:

$$\mathscr{S}_1 := \begin{cases} \mathbf{L}\cdot\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}\cdot\mathbf{R} = \mathbf{L}'\cdot\overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}}\cdot\mathbf{R}' \\ \mathbf{L}\cdot\left(\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{1}}-\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}\right)\cdot\mathbf{R} = \mathbf{L}'\cdot\left(\overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{1}}-\overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}}\right)\cdot\mathbf{R}' \\ \mathbf{L}\cdot\left(\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{2}}-\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}\right)\cdot\mathbf{R} = \mathbf{L}'\cdot\left(\overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{2}}-\overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}}\right)\cdot\mathbf{R}' \\ \quad\vdots \qquad\qquad\qquad\qquad \vdots \\ \mathbf{L}\cdot\left(\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{n+1}}-\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}\right)\cdot\mathbf{R} = \mathbf{L}'\cdot\left(\overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{n+1}}-\overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}}\right)\cdot\mathbf{R}' \end{cases} \tag{17}$$

where $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{i}}$ denotes a matrix having its input populated by the i^{th} vector of the n+1 canonical basis (i.e. 1 in position i, remaining entries 0).

Remark 5. Our goal is to solve the system, and obtain a general form for $\mathbf{L}'$ and $\mathbf{R}'$, in terms of $\mathbf{L}$, $\mathbf{R}$, nimv and nimv'.

First, we make a simplifying substitution in the system above. We set $\mathbf{L}=\mathbf{L}'$ and obtain the following simplified form of $\mathscr{S}_1$:

$$\mathscr{S}_2 := \begin{cases} \overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}\cdot\mathbf{R} = \overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}}\cdot\mathbf{R}' \\ \left(\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{1}}-\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}\right)\cdot\mathbf{R} = \left(\overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{1}}-\overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}}\right)\cdot\mathbf{R}' \\ \left(\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{2}}-\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}\right)\cdot\mathbf{R} = \left(\overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{2}}-\overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}}\right)\cdot\mathbf{R}' \\ \quad\vdots \qquad\qquad\qquad\qquad \vdots \\ \left(\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{n+1}}-\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}\right)\cdot\mathbf{R} = \mathbf{L}'\cdot\left(\overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{n+1}}-\overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}}\right)\cdot\mathbf{R}' \end{cases} \tag{18}$$

In order to reach the aforementioned goal of finding a randomness under which the system can be interpreted as encoding either nimv or nimv', we need decomposition properties for $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$ and $\mathbf{R}$. We define the decomposition of $\overline{\mathbf{G}}_{\mathsf{nimv}||\mathsf{inp}}$ and $\mathbf{R}$ in Section 6.1, analyse their properties in Section 6.2, and go back to solving Equation (18) in Section 6.3.

6.1 Row Decomposition of $\overline{\mathbf{G}}_{\mathsf{nimv}||\mathsf{inp}}$ and $\mathbf{R}$

Decomposition of $\overline{\mathbf{G}}_{\mathsf{nimv}||\mathsf{inp}}$. Each of the m lines in $\overline{\mathbf{G}}_{\mathsf{nimv}||\mathsf{inp}}$ depends on either: (1) the embedded secret variables nimv, (2) the input inp, or (3) an auxiliary node a. Moreover, each line of $\overline{\mathbf{G}}_{\mathsf{nimv}||\mathsf{inp}}$ contains at most one node that is set to 1 if and only if the corresponding bit of nimv or inp is set to 1. Rows depending on auxiliary nodes introduced in Section 2.2 have exactly *one, fixed, known* entry set to 1. Rows depending on flare nodes are included in the inp-dependent set.

For line j in $\overline{\mathbf{G}}_{\mathsf{nimv}||\mathsf{inp}}$ let x_j^1 denote the node whose value is set to 1 iff the input bit corresponding to position $j$is 1, and x_j^0 is set to 1 iff the input is bit corresponding to position j is 0. For completeness, we note that one of the two terminal nodes is removed (when the first column and last line are removed: when $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$ is obtained from $\mathbf{G}_{\mathsf{nimv}||\vec{0}}$). The unremoved terminal node will be included in the inp-dependent nodes, but it will play no role further (its out-degree is zero).

Below, a pictorial perspective is presented for the rows of $\overline{\mathbf{G}}_{\mathsf{nimv}||\mathsf{inp}}$, and later adapted for $\mathbf{R}$[11]. Lines triggered by inp are depicted in green, the ones triggered by nimv are in red, and the lines corresponding to auxiliary nodes are in blue. We formally describe the decomposition of matrices and the associated properties in Proposition 5.

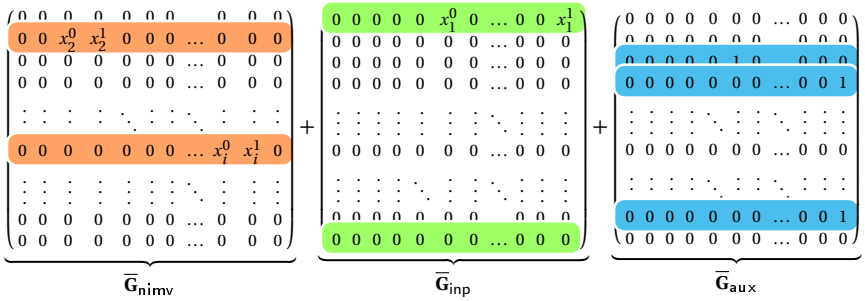
[11] In an extended version, $\mathbf{L}^i$ can be decomposed similarly, which would correspond to a different choice than $\mathbf{L}\neq\mathbf{L}'$.

Consider the following example:

$$\overline{\mathbf{G}}_{\mathsf{nimv||inp}} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & x_1^0 & 0 & \dots & 0 & 0 & x_1^1 \\ \mathbf{1} & 0 & x_2^0 & x_2^1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & \dots & x_i^0 & x_i^1 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & \mathbf{1} & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & \mathbf{1} & 0 \end{pmatrix} \tag{19}$$

We rewrite the matrix $\overline{\mathbf{G}}_{\mathsf{nimv||inp}}$ as a sum of the second diagonal – $\mathbf{J}$ – and of three matrices: one triggered by nimv, one by inp and one being constant (depending on the auxiliary nodes):

$$\overline{\mathbf{G}}_{\mathsf{nimv||inp}} = \mathbf{J} +$$



This intuition is formalized through the following relation:

Definition 12 (Decomposition of $\overline{\mathbf{G}}_{\mathsf{nimv||inp}}$).

$$\overline{\mathbf{G}}_{nimv||\mathsf{inp}} = \mathbf{J} + \overline{\mathbf{G}}_{nimv} + \overline{\mathbf{G}}_{\mathsf{inp}} + \overline{\mathbf{G}}_{aux} \,. \tag{20}$$

Proposition 4. *Let $\mathfrak{N}, \mathfrak{I}, \mathfrak{A}$, denote respectively the set of indexes of non-zero lines of $\overline{\mathbf{G}}_{nimv}, \overline{\mathbf{G}}_{\mathsf{inp}}, \overline{\mathbf{G}}_{aux}$. The following sets are empty: $\mathfrak{N} \cap \mathfrak{I}, \mathfrak{I} \cap \mathfrak{A}$ and $\mathfrak{A} \cap \mathfrak{N}$.*

Proof. Proposition 4 To show this, remember that each node in the BP of $\overline{\mathbf{G}}_{\mathsf{nimv||inp}}$ is labelled by an input (for inp-settled nodes in $\mathfrak{I}$), or by a a bit in the binary representation of nimvs (for nodes in $\mathfrak{N}$), or is simply set to point to a predefined successor, as for auxiliary nodes. The adjacency matrix of this BP will allocate one row for each type of node and hence the sets above are disjoint. □

Input-decomposition of $\overline{\mathbf{G}}_{\mathsf{inp}}$. As seen above $\overline{\mathbf{G}}_{\mathsf{inp}}$ is the sub-matrix of $\overline{\mathbf{G}}$ inheriting each row that corresponds to a node labelled by an input variable x_{inp}. and all the other rows being zero. We will further decompose $\overline{\mathbf{G}}_{\mathsf{inp}}$ into $n+1$ sub-matrices, such that the following equality relation holds:

$$\overline{\mathbf{G}}_{\mathsf{inp}} = \sum_{j=1}^{n+1} \overline{\mathbf{G}}_{\mathsf{inp}_j} \tag{21}$$

and furthermore, if we consider the set of indexes of the non-zero lines in $\overline{\mathbf{G}}_{\mathsf{inp}_j}$ as $\mathfrak{I}_j$, we will have that:

$$\mathfrak{I}_j \cap \mathfrak{I}_j = 0 \,, \quad \forall j \neq k \,. \tag{22}$$

Decomposition of R. By multiplying $\overline{\mathbf{G}}_{\mathsf{nimv}||\mathsf{inp}}$ on the right with $\mathbf{R}$, we observe that each row of $\overline{\mathbf{G}}_{\mathsf{nimv}||\mathsf{inp}} \cdot \mathbf{R}$ depends at most on a *single* input bit x_i, which is set by either inp or nimv (exclusively). This suggests splitting the rows in $\mathbf{R}^i$ in *two* complementary sets: one orthogonal to $\mathbf{G}_{\mathsf{nimv}}$ denoted by $\widetilde{\mathbf{R}_{\mathsf{nimv}}}$ and the remaining part denoted by $\mathbf{R}_{\mathsf{nimv}}$.

$$\mathbf{R} = \underbrace{\begin{pmatrix} 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & & 0 \\ r_{2,1} & r_{2,2} & r_{2,3} & \dots & r_{2,m} \\ r_{4,1} & r_{4,2} & r_{4,3} & \dots & r_{4,m} \\ 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & & 0 \\ r_{m-1,1} & r_{m-1,2} & r_{m-1,3} & \dots & r_{m-1,m} \\ 0 & 0 & 0 & \dots & 0 \end{pmatrix}}_{\mathbf{R}^i_{\mathsf{nimv}}} + \underbrace{\begin{pmatrix} r_{1,1} & r_{1,2} & r_{1,3} & & r_{1,m} \\ r_{2,1} & r_{2,2} & r_{2,3} & \dots & r_{2,m} \\ 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & & 0 \\ r_{8,1} & r_{8,2} & r_{8,3} & \dots & r_{8,m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ r_{m-1,1} & r_{m-1,2} & r_{m-1,3} & \dots & r_{m-1,m} \\ 0 & 0 & 0 & & 0 \\ r_{m,1} & r_{m,2} & r_{m,3} & \dots & r_{m,m} \end{pmatrix}}_{\widetilde{\mathbf{R}_{\mathsf{nimv}}}^i}$$

Observe that this representation is antisymmetric, as $\overline{\mathbf{G}}_{\mathsf{nimv}||\mathsf{inp}}$ is split in three.

Formally, we capture the intuition presented above under the following notation:

Definition 13 (Decomposition of R).

$$\mathbf{R} = \mathbf{R}_{nimv} + \widetilde{\mathbf{R}_{nimv}} \,. \tag{23}$$

Remark 6. The lines triggered by nimv in $\overline{\mathbf{G}}_{\mathsf{nimv}||\mathsf{inp}}$ are to be multiplied with the entries in $\mathbf{R}_{\mathsf{nimv}}$, while *all* the remaining entries (i.e. blue and green) in $\overline{\mathbf{G}}_{\mathsf{nimv}||\mathsf{inp}}$ are to be multiplied with $\widetilde{\mathbf{R}_{\mathsf{nimv}}}$.

Input-decomposition of $\widetilde{\mathbf{R}_{\mathsf{nimv}}}$. $\widetilde{\mathbf{R}_{\mathsf{nimv}}}$ can be decomposed as well.

Definition 14. *We define* $\widetilde{\mathbf{R}_{nimv\,j}}$ *as* $\widetilde{\mathbf{R}_{nimv\,j}} \leftarrow \overline{\mathbf{G}}_{\mathsf{inp}\,j} \cdot \widetilde{\mathbf{R}_{nimv}}$.

Remark 7. As opposed to the input decomposition of $\overline{\mathbf{G}}_{\mathsf{nimv}}$, the decomposition of $\widetilde{\mathbf{R}_{\mathsf{nimv}}}$ does **not** result in a "disjoint" set of matrices. If we consider $\mathfrak{I}_j$ the set of indexes of rows within $\widetilde{\mathbf{R}_{\mathsf{nimv}\,j}}$, then it may be the case that there exists indices j,k such that $\mathfrak{I}_k \cap \mathfrak{I}_j \neq 0$. This relation can be visualized easily as follows: some nodes labelled by inputs j,k within the branching program may point to the same inp-dependent node.

6.2 Properties of Decomposition

As shown previously in the decomposition of $\mathbf{R}$, the matrix $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}} \cdot \mathbf{R}$ (note that each $\mathsf{inp}_j := 0$ in $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$) has each row depending on nimv or on some position j of input. Consider the following remarks:

Remark 8. The product of $\overline{\mathbf{G}}_{\mathsf{nimv}}$ and $\widetilde{\mathbf{R}_{\mathsf{nimv}}}$ is the zero matrix. This happens because each entry of $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$ settled by nimv is multiplied exclusively with entries from $\mathbf{R}_{\mathsf{nimv}}$ (a result formalized below). Each entry in $\overline{\mathbf{G}}_{\mathsf{inp}}$ is multiplied exclusively with rows of $\widetilde{\mathbf{R}_{\mathsf{nimv}}}$. Each entry in $\overline{\mathbf{G}}_{\mathsf{aux}}$ is multiplied exclusively with rows in $\widetilde{\mathbf{R}_{\mathsf{nimv}}}$. This happens because there is no direct arc among auxiliary nodes.

Using these notations, observe that the following condition holds:

$$\begin{aligned} \overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}} \cdot \mathbf{R} &= \left(\mathbf{J} + \overline{\mathbf{G}}_{\mathsf{nimv}} + \overline{\mathbf{G}}_{\mathsf{aux}} + \sum_{j=1}^{n+1} \overline{\mathbf{G}}_{\mathsf{inp}_j} \right) \cdot \left(\mathbf{R}_{\mathsf{nimv}} + \widetilde{\mathbf{R}_{\mathsf{nimv}}} \right) \\ &= \mathbf{J} \cdot \mathbf{R} + \overline{\mathbf{G}}_{\mathsf{nimv}} \cdot \mathbf{R}_{\mathsf{nimv}} + \overline{\mathbf{G}}_{\mathsf{aux}} \cdot \widetilde{\mathbf{R}_{\mathsf{nimv}}} + \sum_{j=1}^{n+1} \overline{\mathbf{G}}_{\mathsf{inp}_j} \cdot \widetilde{\mathbf{R}_{\mathsf{nimv}}} \end{aligned} \tag{24}$$

In Equation (24), we denote by $\overline{\mathbf{G}}_{\mathsf{inp}_j}$ the submatrix of $\overline{\mathbf{G}}_{\mathsf{inp}}$ having its rows labelled by inp_j. Such a decomposition can be done similarly for the left matrix $\mathbf{L}$, however, we avoid it as we set $\mathbf{L} = \mathbf{L}'$ for this work.

Proposition 5 (Decompositions' properties). *Let* $\mathbf{R}, \overline{\mathbf{G}}$ *be defined as in Equation* (20)*, Equation* (23)*, Equation* (5)*. The following relations hold:*

1) $\overline{\mathbf{G}}_{nimv} \cdot \widetilde{\mathbf{R}_{nimv}} = 0$.
2) $\overline{\mathbf{G}}_{aux} \cdot \mathbf{R}_{nimv} = 0$.
3) For all $j \in [n]$*:* $\overline{\mathbf{G}}_{\mathsf{inp}_j} \cdot \mathbf{R}_{nimv} = 0$.

Proof (Proposition 5). The proof relies on the colouring properties of the matrices described in Equation (20), Equation (23), Equation (5).

Consider the first relation: $\widetilde{\mathbf{R}_{\mathsf{nimv}}}$ is a sub-matrix of $\mathbf{R}$ which has all entries set to 0, except for rows $j \in \mathfrak{IA}$, which are identical to the corresponding rows of $\mathbf{R}$ indexed by j. On the other hand $\overline{\mathbf{G}}_{\mathsf{nimv}}$ is the sub-matrix of $\overline{\mathbf{G}}$ having within its every non-zero rows, one element set to 1. Row α in $\overline{\mathbf{G}}$, corresponding to some vertex α (labelled by some bit in nimv), has an entry set to 1 in position β, where $\mathsf{succ}(\alpha) = \beta$. On the other hand, we know that $\beta \notin \mathfrak{IA}$, as

$$\mathfrak{IA} = \{\gamma : \gamma = \mathsf{succ}(\delta) \wedge (\delta \in \mathfrak{A} \vee \delta \in \mathfrak{I})\} . \tag{25}$$

Due to the fact that we used auxiliary nodes, no successor of an input/auxiliary-labelled node can point to a successor of a nimv-node.

The other two cases are done similarly: the successors of $\overline{\mathbf{G}}_{\mathsf{aux}}$ are only in $\mathfrak{A}$, and thus not in $\mathfrak{N}$, so they product of $\overline{\mathbf{G}}_{\mathsf{aux}} \cdot \mathbf{R}_{\mathsf{nimv}} = 0$, as each non-zero row in $\overline{\mathbf{G}}_{\mathsf{aux}}$ has a 1 in position β and $\beta \notin \mathfrak{N}$. A similar argument is applies for $\overline{\mathbf{G}}_{\mathsf{inp}_j}$, having successors in $\mathfrak{I}$. □

6.3 ADPs for Admission Circuit Classes are IND-Secure

In fact, we show a very strong guarantee: given two sets of variables nimv and nimv′, and considering two functionally equivalent, but different, circuits namely $\mathscr{C}_{\mathsf{nimv}}(\mathsf{m}) = \mathscr{C}_{\mathsf{nimv}'}(\mathsf{m}), \forall \mathsf{m} \in \mathscr{M}$ – we show that there exists randomness terms $(\mathbf{L}, \mathbf{R})$ and $(\mathbf{L}', \mathbf{R}')$ such that $\mathsf{ADP.Setup}(\mathscr{C}^i_{\mathsf{nimv}}; (\mathbf{L}, \mathbf{R})) = \mathsf{ADP.Setup}(\mathscr{C}^i_{\mathsf{nimv}'}; (\mathbf{L}', \mathbf{R}'))$. In some sense, an adversary cannot guess to whom it corresponds the ADP: to nimv or to nimv′.

As expected, we can prove such a strong property only for some *very* restricted classes of circuits. Interestingly, the proof we give is computationally simple, using only basic algebraic properties.

Theorem 5 (IND-ADP **programs for Figure 3).** *Let* pPRF *denote a puncturable* PRF *admitting circuits in* NC^1*, and let* FE *denote a functional encryption scheme admitting a circuit representation of its encryption scheme in* NC^1*. Let* $\mathscr{C}^i$ *denote the circuit described in Figure 3. Let* nimv *and* nimv′ *be two sets of non-input dependent mutable variables. Then, the distributions of* $\mathsf{ADP.Setup}(1^\lambda, \mathscr{C}^i_{nimv})$ *and* $\mathsf{ADP.Setup}(1^\lambda, \mathscr{C}^i_{nimv'})$ *are identical.*

Proof (Theorem 5). We show that for any two sets of variables nimv and nimv′ we have that $\mathsf{ADP.Setup}(1^\lambda, \mathscr{C}_{\mathsf{nimv}}) = \mathsf{ADP.Setup}(1^\lambda, \mathscr{C}_{\mathsf{nimv}'})$. We know from Equation (18) that:

$$\mathscr{S}_2 := \begin{cases} \overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}} \cdot \mathbf{R} = \overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}} \cdot \mathbf{R}' \\ \left(\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{1}} - \overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}\right) \cdot \mathbf{R} = \left(\overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{1}} - \overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}}\right) \cdot \mathbf{R}' \\ \left(\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{2}} - \overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}\right) \cdot \mathbf{R} = \left(\overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{2}} - \overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}}\right) \cdot \mathbf{R}' \\ \qquad \vdots \qquad\qquad\qquad\qquad \vdots \\ \left(\overline{\mathbf{G}}_{\mathsf{nimv}||n\vec{+}1} - \overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}\right) \cdot \mathbf{R} = \mathbf{L}' \cdot \left(\overline{\mathbf{G}}_{\mathsf{nimv}'||n\vec{+}1} - \overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}}\right) \cdot \mathbf{R}' \end{cases} \tag{26}$$

Corroborating this with the decomposition of $\overline{\mathbf{G}}_{\mathsf{nimv}||\mathsf{inp}}$ and $\mathbf{R}$ obtained in Section 6.1, we get:

$$\mathscr{S}_3 := \begin{cases} \overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}} \cdot \left(\mathbf{R}_{\mathsf{nimv}} + \widetilde{\mathbf{R}_{\mathsf{nimv}}}\right) = \overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}} \cdot \left(\mathbf{R}'_{\mathsf{nimv}} + \widetilde{\mathbf{R}_{\mathsf{nimv}}}'\right) \\ \quad \text{and} \\ \forall j \in [n+1] : \quad \left(\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{j}} - \overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}\right) \cdot \widetilde{\mathbf{R}_{\mathsf{nimv}}}_j = \left(\overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{j}} - \overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}}\right) \cdot \widetilde{\mathbf{R}_{\mathsf{nimv}}}'_j \end{cases} \tag{27}$$

Step 1 - Simplifying the System of Equations. In the equation above we set:

$$\forall j \in [n+1]: \ \widetilde{\mathbf{R}_{\mathsf{nimv}\,j}} = \widetilde{\mathbf{R}_{\mathsf{nimv}\,j}}' \ . \tag{28}$$

This is the choice we make to simplify the proof, by eliminating the need to act on the second part of the system above independently (this results by replacing Equation (28) into Equation (27)). We consider the equation above, simplified, once again, in a more compact form:

$$\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}} \cdot \mathbf{R} = \overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}} \cdot \mathbf{R}' \iff \overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}} \cdot \left(\mathbf{R} \cdot (\mathbf{R}')^{-1}\right) = \overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}} \iff \overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}} \cdot \mathbf{A} + \overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}} = 0 \ . \tag{29}$$

We note that $\mathbf{A} = \mathbf{R} \cdot (\mathbf{R}')^{-1}$ which gives us the following relation:

$$\mathbf{A} \cdot \mathbf{R}' = \mathbf{R} \ . \tag{30}$$

Step 2 - Constraints on A. Looking at Equation (30) and correlating it with the fact that $\widetilde{\mathbf{R}_{\mathsf{nimv}\,j}} = \widetilde{\mathbf{R}_{\mathsf{nimv}\,j}}', \forall j \in [|\mathsf{inp}|]$, we conclude that we want the rows of $\mathbf{R}$ and $\mathbf{R}'$ depending on inp to be equal. Such a satisfying assignment comes from setting the relevant lines (those depending on inp) of $\mathbf{A}$, to be the ones of the identity matrix $\mathbf{I}_m$ (the lines of $\mathbf{I}_m$ corresponding to inp). We formalize this observation within the following claim.

Proposition 6 (Step 2). *Let $\mathbf{A}, \mathbf{R}, \mathbf{R}'$ denote matrices over $\mathbb{F}_2^{m \times m}$ s.t. $\mathbf{A} \cdot \mathbf{R}' = \mathbf{R}$. For each input position $j \in [n+1]$, let $\mathfrak{W}_j \subseteq [m]$ be defined as $\mathfrak{W}_j = \{k : k \in [m] \wedge \mathit{label}(k) = \mathsf{inp}_j\}$ and let $\widetilde{\mathbf{R}_{\mathsf{nimv}\,j}}$ be a sub-matrix of both $\mathbf{R}$ and $\mathbf{R}'$ consisting of rows indexed(labelled) by $k \in \mathfrak{W}_j$. Then, each row $k \in \mathfrak{W}_j$ of $\mathbf{A}$ has 1 in position k and all other entries 0.*

Proof. Let $\mathbf{a}_k$ denote row k of $\mathbf{A}$, where $k \in \mathfrak{W}_j$. Fix

$$\mathbf{a}_k = (0, 0, \ldots, 1, 0, 0, \ldots, 0) \ ,$$

and observe that $\mathbf{a}_k \cdot \mathbf{R}' = \mathbf{r}_k$, where $\mathbf{r}_k$ is the k^{th} line of $\mathbf{R}'$ and of $\mathbf{R}$. □

Step 3 - Simplification. Now, we are left to show that in Equation (29) there is always an $\mathbf{A}$ such that $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}} \cdot \mathbf{A} + \mathbf{I}_m \cdot \overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}} = 0$, which reduces to

$$\mathbf{A} \leftarrow \mathbf{I}_m + \overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}^{-1} \cdot \mathbf{S} \tag{31}$$

where $\mathbf{S} \leftarrow \overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}} - \overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$. Note that in Equation (31), we assume the existence of a nimv that makes $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$ invertible (thus matching Condition 9).

Step 4 - Why Augmentation of BPs Matters. In order to tackle Equation (31), observe that $\mathbf{S}$ is a very sparse matrix. In essence, it consists only of red lines, each line containing exactly two entries set to 1 – corresponding to the elements that are set by the bits in nimvs:

$$\mathbf{S} \leftarrow \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & & 0 & 0 & 0 \\ 0 & 0 & 1^0 & 1^1 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 1^0 & 0 & \ldots & 0 & 1^1 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 \end{pmatrix} \tag{32}$$

Proposition 7 (Step 4). *Any column of $\mathbf{S}$ contains at most one entry set to 1.*

Proof (Proposition 7). This follows from the usage of auxiliary nodes, where we direct each node depending on nimv to a dummy node having its *out*- degree set to 1. Assume by contradiction that there exists some column c having two entries set to 1. It must be the case that two nimv dependent vertices point to the same node. However, by the auxiliary nodes transform, each nimv dependent node, depending on its value, point to its own dummy nodes, which are not shared with other nimv dependent nodes. Another way to look into this is to observe that nimv-dependent lines are coloured in red will imply a linear combination over the lines of $\mathbf{R}_{\mathsf{nimv}}$, and are not mixed with the green lines forming $\widetilde{\mathbf{R}_{\mathsf{nimv}}}$. □

Step 5 - Shape of A, R′ and R. We turn back to populating the matrix **A** described in Equation (31), subject to the constraint from Proposition 6. We want that line k in **A** (that is triggered by some bit j in inp, with $k \in \mathfrak{W}_j$) to be the same as the line k of $\mathbf{I}_m$ – i.e. to contain 1 only in position k (as formalized in Proposition 6).

Roughly speaking, from the previous statement we can infer that line k in $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}^{-1}$ multiplied with all columns in **S** must be 0. Otherwise $\mathbf{A} = \mathbf{I}_m + \overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}^{-1} \cdot \mathbf{S}$ will not have its line k with 1 in position k, as required by Proposition 6. Using this observation we show the following lemma:

Lemma 1. *Let* $\mathbf{A} \leftarrow \mathbf{I}_m + \overline{\mathbf{G}}_{nimv||\vec{0}}^{-1} \cdot \mathbf{S}$, *where* $\mathbf{S} \leftarrow \overline{\mathbf{G}}_{nimv'||\vec{0}} - \overline{\mathbf{G}}_{nimv||\vec{0}}$ *defined as above. Then, it can be shown that:*

$$\mathbf{g}_k \cdot \mathbf{S} = 0 \tag{33}$$

where $\mathbf{g}_k \in \mathbb{F}_2^{1\times m}$ *is the* k^{th} *line of* $\overline{\mathbf{G}}_{nimv||\vec{0}}^{-1} \in \mathbb{F}_2^{m\times m}$, $k \in \mathfrak{W}$ *and* $\mathfrak{W} = \bigcup_{j=1}^{n+1} \{k : k \in [m] \wedge label(k) = \mathsf{inp}_k\}$.

Proof (Lemma 1). **S** is a very sparse matrix (as seen above), and each column contains at most a single 1. Let $\mathbf{g}_k$ denote the k^{th} row of $\overline{\mathbf{G}}^{-1}_{\mathsf{nimv}||\vec{0}}$ and let **h** stand for some column of **S**. Assume there exists an index $\ell \in [m]$ such that **h** has 1 in position ℓ for some **h** a non-zero column of **S**.

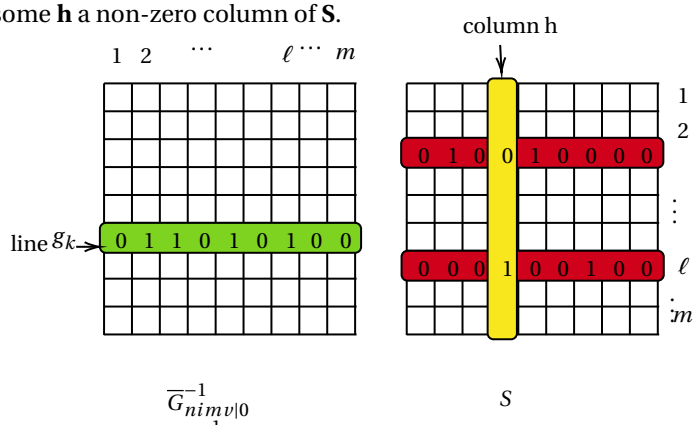


Fig. 4. An overview of the multiplication of $\overline{\mathbf{G}}^{-1}_{\mathsf{nimv}||\vec{0}}$ with **S**. Lines coloured in green have their indexes k belonging to $\mathfrak{W}$.

We work with elementary algebra and we distinguish two cases:

Case 1. If the element in position ℓ in $\mathbf{g}_k (k \in \mathfrak{W})$ is already 0, then nothing has to be done, as this line multiplied with column **h** of **S** will give a zero, and things work as expected.

Case 2 (or why the flares matter). If the element in position ℓ in $\mathbf{g}_k$ of $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}^{-1}$ is 1, then the multiplication with column of **S** seems problematic. We show below a contradiction based on flare nodes (Condition 12), implying that entry ℓ in $\mathbf{g}_k$ cannot be 1, for any position ℓ that is set to 1 in any non-zero column **h** of **S**.

We rely on flares' action on the topology of the circuit for this thing not to happen. If $\mathbf{g}_k$ has 1 in position ℓ, we show that either the very first line of $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$ is $(0, 0, \ldots, 1)$ (Condition 11), or we will reach a contradiction based on the behaviour of a flare vertex. To see this, we consider the relation $\overline{\mathbf{G}}^{-1}_{\mathsf{nimv}||\vec{0}} \cdot \overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}} = \mathbf{I}_m$. Observe that $\mathbf{g}_k$ (column k in $\overline{\mathbf{G}}^{-1}_{\mathsf{nimv}||\vec{0}}$) will get multiplied with columns of $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$, and in all but one cases the result must be 0.

First, remember that on line ℓ, **S** has two entries set to 1 (one corresponding to $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$, the other to $\overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}}$). Second, assume that the entry $\mathbf{S}_{\ell,h} = 1$ will also be part of $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$ (the other case does not need to be treated once we show that $\mathbf{g}_\ell = 0$). Assume for contradiction that $\mathbf{g}_{k,\ell} \cdot \mathbf{S}_{\ell,h} \neq 0$.

The trick is to observe that when we multiply $\mathbf{g}_k$ with some column of $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$ that depends on nimv – that is the column **h** having 1 in position ℓ and an extra 1 due to the second diagonal – we must obtain 0, otherwise $\overline{\mathbf{G}}^{-1}_{\mathsf{nimv}||\vec{0}} \cdot \overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}} = \mathbf{I}_m$ does not hold. Observe also that $\ell \neq h$, as k indexes an inp dependent node while h is the index of an aux node.

Subcase "Go left". Now we multiply $\mathbf{g}_k$ with the column of $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$ that has 1 in position ℓ, but **due to the second diagonal**[12]. It must be the case the product between $\mathbf{g}_j$ and this current column is 0. The reason is due

[12] Pictorially we selected a column of $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$ which is placed to the left of the original starting column.

to the property o the multiplication with inverse, and in particular because $k \neq$ pivot: k is the index of an inp-dependent node and the column pivot is indexed by a nimv-node. Thus the product of line $\mathbf{g}_k$ with column h must be 0.

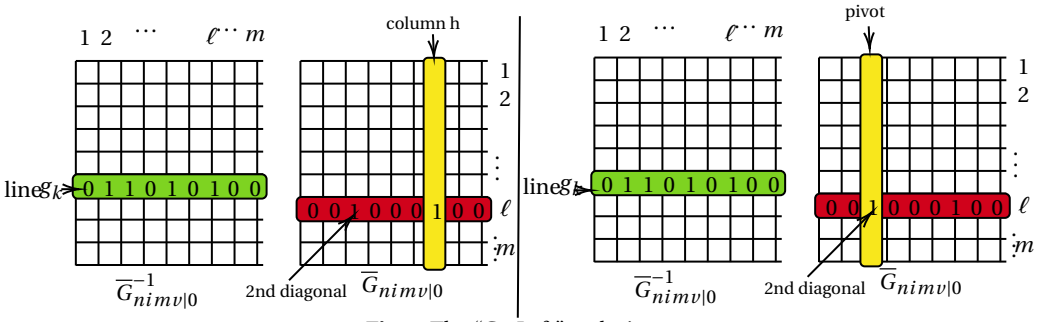


Fig. 5. The "Go Left" technique.

"Go" up. Since the previous product is 0 it must be the case that the current column (pivot in Figure 6,left) has (at least) a 1 in its upper part (as elements below the 2nd diagonal are zero), and $\mathbf{g}_k$ has a 1 in its left part (i.e. left to position ℓ), such that their product is 1, otherwise the assumption is contradicted.

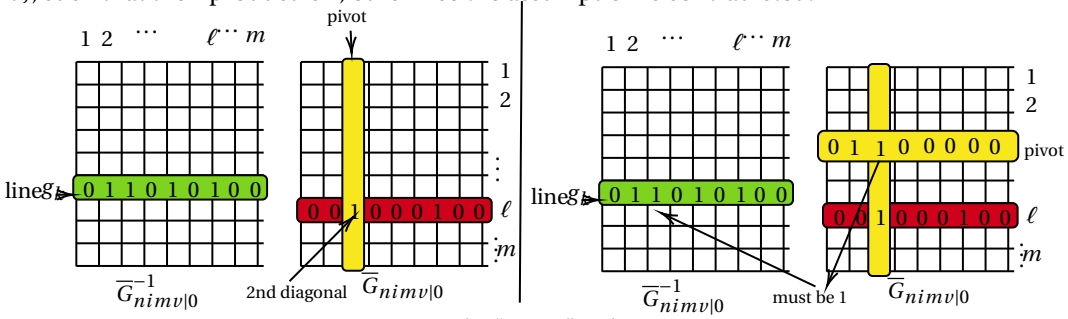


Fig. 6. The "Go Up" technique.

The previous two steps are repeated, maintaining an invariant that enforces the left part (with respect to the current index) of $\mathbf{g}_k$ to contain an entry set to 1 in some index say pivot. It is easy to see that by using this technique we traverse the branching program. At the end of the traversal, we will reach either a flare node or the leftmost column of $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$. As we traverse upwards, we $\mathbf{g}_k$ will never have to be multiplied with itself. The reason is that $\mathbf{g}_k$ is a column indexed by a bit in inp, while when traversing, up and left, the columns will be indexed by nimv or aux nodes (as flares interrupt paths).

In case that by maintaining the invariant, we reach a "flare" node *fl*, we remind the fact that for the base matrix, the first input node is set to 0. Thus, the flare "redirects" to the terminal node 1. That is, it interrupts the current path, and in this case, we get a contradiction as $\mathbf{g}_k$ multiplied with $\mathbf{h}$ obtained from the latest iteration is 1, while the basic algebraic property require this product to be 0 (due to the multiplication of a matrix with its inverse).

In case we reach the leftmost column of $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$, which has its entries $(0,1,0,\ldots)^\top$, while $\mathbf{g}_k$ will be $(0,1,\ldots)$ or $(1,1,\ldots)$. The inner product of these two quantities is 1, as opposed to the expected 0. This settles a topology for $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$, related to the very first line.

In order to achieve the constrains enforced by the subcase above, one can set the matrix $\left(\overline{\mathbf{G}}^1_{\mathsf{inp}_1} - \overline{\mathbf{G}}^0_{\mathsf{inp}_1}\right)$ corresponding to inp_1 to have the first line set to $(1,0,0,\ldots,1)$. Thus, when this matrix is added to $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$, one can correctly evaluate the augmented branching program. □

As $\mathbf{R}$ can always be sampled, and the same holds for $\mathbf{L}$, we have a satisfying assignment for Equation (27). We can easily show the statistical distance between these distributions is 0, as be have a bijection between the choices for $(\mathbf{L},\mathbf{R})$ and $\left(\mathbf{L}',\mathbf{R}'\right)$, notably with $\mathbf{L}=\mathbf{L}'$ and $\mathbf{R} = (\mathbf{I}_m + \overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}{}^{-1} \cdot (\overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}} - \overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}})) \cdot \mathbf{R}'$. Thus, it follows that the two distributions are identical and therefore the implementation is IND-ADP-secure. □

References

1. Shweta Agrawal. Stronger security for reusable garbled circuits, general definitions and attacks. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 3–35. Springer, Heidelberg, August 2017.
2. Shweta Agrawal and Alon Rosen. Functional encryption for bounded collusions, revisited. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 173–205. Springer, Heidelberg, November 2017.
3. Undisclossed Authors. Personal communication (in submission), 2023.
4. Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 719–737. Springer, Heidelberg, April 2012.
5. Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Heidelberg, August 2001.
6. David A Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc1. *Journal of Computer and System Sciences*, 38(1):150–164, 1989.
7. Jim Barthel and Răzvan Roşie. Nike from affine determinant programs. In *Provable and Practical Security: 15th International Conference, ProvSec 2021, Guangzhou, China, November 5–8, 2021, Proceedings*, page 98–115. Springer-Verlag, 2021.
8. James Bartusek, Yuval Ishai, Aayush Jain, Fermi Ma, Amit Sahai, and Mark Zhandry. Affine determinant programs: A framework for obfuscation and witness encryption. In Thomas Vidick, editor, *ITCS 2020*, volume 151, pages 82:1–82:39. LIPIcs, January 2020.
9. Dan Boneh, Sam Kim, and Hart William Montgomery. Private puncturable PRFs from standard lattice assumptions. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 415–445. Springer, Heidelberg, April / May 2017.
10. Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 253–273. Springer, Heidelberg, March 2011.
11. Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 480–499. Springer, Heidelberg, August 2014.
12. Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 505–524. Springer, Heidelberg, August 2011.
13. Zvika Brakerski and Vinod Vaikuntanathan. Constrained key-homomorphic PRFs from standard lattice assumptions - or: How to secretly embed a circuit in your PRF. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 1–30. Springer, Heidelberg, March 2015.
14. Ran Canetti and Yilei Chen. Constraint-hiding constrained PRFs for NC^1 from LWE. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 446–476. Springer, Heidelberg, April / May 2017.
15. Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 1–17. Springer, Heidelberg, May 2013.
16. Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.
17. Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Heidelberg, August 2013.
18. Shafi Goldwasser, S. Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 578–602. Springer, Heidelberg, May 2014.
19. Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 555–564. ACM Press, June 2013.
20. Shafi Goldwasser and Guy N. Rothblum. On best-possible obfuscation. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 194–213. Springer, Heidelberg, February 2007.
21. Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 545–554. ACM Press, June 2013.
22. Yuval Ishai. Secure computation and its diverse applications (invited talk). In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, page 90. Springer, Heidelberg, February 2010.

23. Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan Eidenbenz, and Ricardo Conejo, editors, *ICALP 2002*, volume 2380 of *LNCS*, pages 244–256. Springer, Heidelberg, July 2002.
24. Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 60–73. ACM, 2021.
25. Samuel Jaques, Hart Montgomery, and Arnab Roy. Time-release cryptography from minimal circuit assumptions. Cryptology ePrint Archive, Paper 2020/755, 2020. https://eprint.iacr.org/2020/755.
26. Donald E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 1: Bitwise Tricks and Techniques; Binary Decision Diagrams*. 2009.
27. Huijia Lin, Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation with non-trivial efficiency. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part II*, volume 9615 of *LNCS*, pages 447–462. Springer, Heidelberg, March 2016.
28. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-LWE cryptography. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 35–54. Springer, Heidelberg, May 2013.
29. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.
30. Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014.
31. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

A Puncturable PRFs with Evaluations in NC^1

In this part, we provide evidence that a particular version of the constrained PRF from [13] – namely the "toy" puncturable PRF informally introduced by Boneh, Kim and Montgomery in [9, Section 1] – admits an NC^1 circuit representation of the evaluation function. This informal scheme is chosen for space reasons, and also for simplicity (avoiding the usage of universal circuits in its description).

The notations that are used in this section are as follows: $|\mathsf{inp}|$ stands for the length of the input string, λ stands for the security parameter, n and m stand for the dimensions of the matrix used in the construction, q stand for the LWE modulus.

$\underline{\mathsf{pPRF.Setup}(1^\lambda, 1^{|\mathsf{inp}|})}$**:** given the unary description of the security parameter λ, proceed as follows

1. Sample a (column) vector:
$$\mathbf{s} \leftarrow\!\!\$\ \mathbb{Z}_q^n \ .$$
2. Sample $|\mathsf{inp}|$ matrices $\mathbf{B}_i$ of dimensions $n \times m$ uniformly at random over $\mathbb{Z}_q^{n\times m}$, for all $i \in [|\mathsf{inp}|]$. That is:
$$\mathbf{B}_i \leftarrow\!\!\$\ \mathbb{Z}_q^{n\times m} \ .$$
3. Sample 2 matrices $\mathbf{A}_0, \mathbf{A}_1$ as before:
$$(\mathbf{A}_0, \mathbf{A}_1, \mathbf{D}) \leftarrow\!\!\$\ \left(\mathbb{Z}_q^{n\times m}, \mathbb{Z}_q^{n\times m}, \mathbb{Z}_q^{n\times m}\right) \ .$$
4. Set as secret key:
$$\mathsf{k} \leftarrow \left(\mathbf{s},\ \mathbf{B}_1,\ \ldots,\ \mathbf{B}_{|\mathsf{inp}|},\ \mathbf{A}_0,\ \mathbf{A}_1, \mathbf{D}\right) \ .$$

$\underline{\mathsf{pPRF.Eval}(\mathsf{k}, \mathsf{inp})}$**:** to evaluate input inp under pPRF key k, proceed as follows

1. Use the $\mathsf{PK}_{\mathsf{Eval}}$ evaluation algorithm from [13] (detailed below) in order to publicly compute a matrix $\mathbf{A}_{\mathsf{eq}}$.
2. Compute: $Y \leftarrow \mathbf{s}^\top \cdot \mathbf{A}_{\mathsf{eq}} \cdot \mathbf{G}^{-1}(\mathbf{D})$.
3. Return $\lfloor Y \rceil$, where $\lfloor \cdot \rceil$ is a rounding function: $\lfloor \cdot \rceil : \mathbb{Z}_q \to \mathbb{Z}_p$ that maps $x \to \lfloor x \cdot (p/q) \rceil$, i.e. the argument x is multiplied with p/q and the result is rounded (over reals).

$\underline{\mathsf{pPRF.Puncture}(1^\lambda, \mathsf{k}, \mathsf{inp}^*)}$**:**

1. Return the punctured key for $\mathsf{inp}^* = \left(\mathsf{inp}_1^*, \ldots, \mathsf{inp}_{|\mathsf{inp}|}^*\right) \in \{0,1\}^{|\mathsf{inp}|}$ as

$$\mathsf{k}^* \leftarrow \left(\mathsf{inp}^*, \mathbf{B}_1, \ \ldots, \ \mathbf{B}_{|\mathsf{inp}|}, \mathbf{A}_0, \mathbf{A}_1, \mathbf{D}, \left\{\mathbf{s}^\top \cdot (\mathbf{A}_k + k \cdot \mathbf{G}) + \mathbf{e}_k\right\}_{k\in\{0,1\}}, \left\{\mathbf{s}^\top \cdot \left(\mathbf{B}_k + \mathsf{inp}_k^* \cdot \mathbf{G}\right) + \mathbf{e}_k^*\right\}_{k=1}^{|\mathsf{inp}^*|}\right) .$$

2. Return k^*.

$\underline{\mathsf{pPRF.PuncEval}(1^\lambda, \mathsf{k}^*, \mathsf{inp})}$**:** To evaluate in the punctured point:

1. Compute the encoding evaluation (detailed below) over the punctured key and obtain Y:

$$Y \leftarrow \left(\underbrace{\mathbf{s}^\top \cdot \left(\mathbf{A}_{\mathsf{eq}} + \mathsf{eq}(\mathsf{inp}^*, \mathsf{inp}) \cdot \mathbf{G}\right) + \mathbf{e}'}_{\mathsf{CT}_{\mathsf{Eval}}} \right) \cdot \mathbf{G}^{-1}(\mathbf{D}) .$$

2. Return $\lfloor Y \rceil$ where $\lfloor \cdot \rceil$ is the same rounding function used by the normal evaluation.

Observe that when $\mathsf{eq}(\mathsf{inp}^*, \mathsf{inp}) = 0$, the value Y computed in punctured evaluation is in fact $\mathbf{s}^\top \cdot \mathbf{A}_{\mathsf{eq}} + \mathbf{e}'$. The correctness and security are based on the constrained PRF scheme from [13], hence we ignore them herein. We focus on the runtime analysis of the punctured evaluation algorithm. In doing so, we need the public and the encoding evaluation algorithm.

A.1 The Encoding Evaluation Algorithm for the pPRF in [13]

Consider a circuit composed from the universal set of gates: AND and NOT.

AND Gates: let $g_{u,v,w}$ be and AND gate, where u and v denote the input wires while w denotes the output. Let $\mathbf{y}_u = \mathbf{s}^\top \cdot (\mathbf{A} + x_u \cdot \mathbf{G}) + \mathbf{u}$ and $\mathbf{y}_v = \mathbf{s}^\top \cdot (\mathbf{A} + x_v \cdot \mathbf{G}) + \mathbf{v}$ where x_u and x_v denote the value of wires u and v corresponding to same input. The evaluation over encodings returns:

$$\mathbf{y}_w \leftarrow x_u \cdot \mathbf{y}_v - \mathbf{y}_u \cdot \mathbf{G}^{-1}(\mathbf{A}_v) . \tag{34}$$

which will be a valid encoding corresponding to the value of w.

NOT Gates: we reuse similar notations for gates as per the previous case, with $g_{u,w}$ being a not gate and input wire is u, and $\mathbf{y}_0$ is an encoding corresponding to the value 0:

$$\mathbf{y}_w \leftarrow \mathbf{y}_0 - \mathbf{y}_u . \tag{35}$$

A.2 The Security Proof

Lemma 2 (pPRF**'s Security).** *The puncturable PRF construction above is a secure according to Definition 2.*

Proof (Lemma 2). Let $\mathscr{A}$ stand for a PPT adversary against pPRF's indistinguishability experiment. Let $t = \mathsf{poly}(\lambda)$ be the number of inputs made by $\mathscr{A}$. Let $B = \alpha \cdot q \cdot \omega(\sqrt{(}\log(\lambda)))$ stand for the LWE bound. We follow almost *ad literam* the proof in [13]. However, we target the easier notion of selective security, rather than adaptive (although a convoluted adaptive proof can be derived similarly to [13] at the cost of using admissible hash functions).

The Real Game – Game_0**.** The adversary commits to a challenge input inp^*. The game samples a key punctured in the input point, and returns it. Then, the challenge value for the punctured input point is answered by returning, with the same probability, a random element ($b = 1$) sampled over the codomain, or the real evaluation of the pPRF under the normal key ($b = 0$).

Game_1 In this game we change the distribution of the public parameters. Let $\{\mathbf{A}'_i\}_{i\in\{0,1\}}$ and $\{\mathbf{B}'_i\}_{i\in[k]}$ stands for the pulic matrices sampled according to the distribution in Game_0. We obtain $\{\mathbf{A}_i\}_{i\in\{0,1\}}$ and $\{\mathbf{B}_i\}_{i\in[\mathsf{inp}]}$ Namely:

$$\begin{aligned} \mathbf{A}_i &\leftarrow \mathbf{A}'_i - b \cdot \mathbf{G} \ \text{ for } i \in \{0,1\} \\ \mathbf{B}_i &\leftarrow \mathbf{B}'_i - x_i^* \cdot \mathbf{G} \ \text{ for } i \in |\mathsf{inp}| \end{aligned} \tag{36}$$

We note that this distribution isidentical to the distribution in the previous game, given the matrices are sampled uniformly at random.

Game_2 In this game, we replace the vector component of the punctured key with uniformly sampled vectors. The reduction stems on LWE. The LWE experiment provides us with tuples ofthe form $(\mathbf{X}, \mathbf{s}\cdot\mathbf{X}+\mathbf{e})$ (x)or with random tuples $(\mathbf{X},\mathbf{r})$ for some $\mathbf{r}$ sampled uniformly at random. We will make use of a specific LWE term which will be either: $(\mathbf{D}, \mathbf{s}^\top\cdot\mathbf{D}+\mathbf{e}_d)$ or $(\mathbf{D},\mathbf{d})$. Then, observe that the elements have the form:

$$\begin{aligned}\mathbf{a}_i &\leftarrow \mathbf{s}^\top\cdot(\mathbf{X}'_i - b\cdot\mathbf{G} + b\cdot\mathbf{G}) + \mathbf{e}_{\mathbf{X},i} = \mathbf{s}^\top\cdot\mathbf{X}'_i + \mathbf{e}_{\mathbf{X},i} \text{ for } i\in\{0,1\}\\ \mathbf{b}_i &\leftarrow \mathbf{s}^\top\cdot(\mathbf{Y}'_i - x^*_i\cdot\mathbf{G} + x^*_i\cdot\mathbf{G}) + \mathbf{e}_{\mathbf{Y},i} = \mathbf{s}^\top\cdot\mathbf{Y}'_i + \mathbf{e}_{\mathbf{Y},i} \text{ for } i\in|\mathsf{inp}|\end{aligned} \tag{37}$$

These values correspond to LWE tuples. Note that in order to simulate the punctured key, we also need to release the pulic matrices, as they are needed to evaluate AND operation over the ciphertext. The interesting part is to show the selective security of the puncturable pseudorandom function in the challenge point inp^*. The challenger must simulate: $\left(\mathbf{s}^\top\cdot\left(\mathbf{A}_{\mathsf{eq}}+\mathbf{G}\right)+\mathbf{e}\right)$. Note that it is easy for the challenger to compute: $\left(\mathbf{s}^\top\cdot\left(\mathbf{A}_{\mathsf{eq}}\right)+\mathbf{e}\right)$ which can be done through the ciphertext evaluation of the element in Equation (37) (i.e. of the LWE tuples). To cope with the challenge value corresponding to inp^*, the challenger employs the additional matrix $\mathbf{D}$ and obtains:

$$\begin{aligned}\left(\mathbf{s}^\top\cdot\left(\mathbf{A}_{\mathsf{eq}}\right)+\mathbf{e}\right)\cdot\mathbf{G}^{-1}(\mathbf{D}) + \left(\mathbf{s}^\top\cdot\mathbf{D}+\mathbf{e}_d\right) &= \left(\mathbf{s}^\top\cdot\left(\mathbf{A}_{\mathsf{eq}}\right)+\mathbf{e}\right)\cdot\mathbf{G}^{-1}(\mathbf{D}) + \left(\mathbf{s}^\top\cdot\mathbf{G}\cdot\mathbf{G}^{-1}(\mathbf{D})+\mathbf{e}_d\right)\\ &= \left(\mathbf{s}^\top\cdot\left(\mathbf{A}_{\mathsf{eq}}+\mathbf{G}\right)+\mathbf{e}\right)\cdot\mathbf{G}^{-1}(\mathbf{D})+\mathbf{e}_d\end{aligned}$$

We note that once the rounding function is applied elementwise, the small noise will be esentially modulated, given that $\mathbf{e}_{d,i} < q/p$. Thus under this restriction, the challenge value is correct. It is easy to see that for randomly sampled values, the simulated Y^* in the pPRF game is uniform. The final part of the proof involving th rounding is similar to the one in [13]. □

A.3 Punctured Evaluation's Parallel Complexity

Here we scrutinize the parallel efficiency of the gate evaluation corresponding to the equality function:

$$\mathsf{eq}(\mathsf{inp}^*,\mathsf{inp}) := \begin{cases}1, \text{ if } \mathsf{inp}=\mathsf{inp}^*\\ 0, \text{ otherwise}\end{cases} \tag{38}$$

An unoptimized circuit that implements the eq function is built as follows:

1. use a gadget matrix that returns the boolean value of $\mathsf{inp}^*_i \stackrel{?}{=} \mathsf{inp}_i$ for some input position $i\in[|\mathsf{inp}|]$. This gadget matrix can be implemented as

$$\text{NOT}\left((\text{NOT}\ \left(\mathsf{inp}^*_i \text{ AND } \mathsf{inp}_i\right)\right) \text{ AND } \left(\text{NOT}\ \left((\text{NOT } \mathsf{inp}^*_i) \text{ AND } (\text{NOT } \mathsf{inp}_i)\right)\right)\right) \tag{39}$$

   Thus the depth of this gadget is 5, and on each of the 5 levels further LWE-related operations are to be performed.

2. use a full-binary tree style of circuit consisting of AND gates that outputs $\bigwedge_{i=1}^{|\mathsf{inp}|}(\mathsf{inp}^*_i \stackrel{?}{=} \mathsf{inp}_i)$. Clearly, this circuit has $\lceil\log_2(|\mathsf{inp}|)\rceil$ levels.

Henceforth, the circuit that computes the evaluation (obtained by applying the construction in step 2 on top of the "gadget" circuit) has depth $\le c\cdot\log_2(|\mathsf{inp}|)$ for some constant c. The matrix multiplication involved in the computation of an AND gate, the values of $\mathbf{G}^{-1}(\mathbf{A}_0)$ and $\mathbf{G}^{-1}(\mathbf{A}_1)$ can be pre-stored, the costly part being a vector $\times$ matrix multiplication. The inner, LWE-related computations within the punctured evaluation algorithm are in NC^1, as for other constructions using LWE tuples, (see for instance [4]). Further details on the complexity of circuits implementing addition/multiplication for elements in $\mathbb{F}_q$ are given in [25, Section 8]). Thus, we can assume that the there exists puncturable PRFs having their punctured evaluation circuit in NC^1 (as expected, also, by [27]).

B GKPVZ13's Encryption Procedure is in NC^1

In this section, we provide an informal argument for the existence of FE schemes having their encryption procedure in NC^1 (an assumption used in [27]). The notation used herein are independent from the ones used in other sections.

B.1 The FE scheme from [19].

Goldwasser *et al.*'s proposal is to regard FE for circuits with a single-bit of output from the perspective of homomorphic operations. Their scheme's *encryption procedure* proceeds as follows: (1) Samples on the fly keys for an FHE scheme – namely $(\mathsf{hpk}, \mathsf{hsk})$ – and encrypts the input m bitwise; let Ψ stand for the FHE ciphertext. (2) Then, the scheme makes use of Yao's garbling protocol GS; this is employed to garble the circuit "$\mathsf{FHE.Dec}(\mathsf{hsk}, \cdot)$" and obtain two labels L_i^0, L_i^1 for each bit in the decomposition of Ψ; (3) Finally, the scheme encrypts Ψ, as well as hpk under a set of ABE public keys (in fact two-outcome ABEs are used). In some sense, Ψ corresponds to an attribute: if $\mathscr{C}_{f_i}(\Psi) = 0$ a label L_i^0 is revealed. Else, the label L_i^1 is returned.

For [19], a *functional key* for a circuit is nothing more than an ABE key issued for the "FHE.Eval" circuit. The trick is that one decrypts an ABE ciphertext with an ABE key; this translates to applying FHE.Eval over an FHE ciphertext. Given the ABE ciphertext encrypts L_i^0, L_i^1, depending on the output value (a bit b), the label L_i^b is returned. After the labels are recovered, they can be used to feed the garbled circuit (included in the ciphertext); the decryptor evaluates and obtains (informally) $\mathsf{FHE.Dec}(f(\Psi))$, thus yielding the expected output in a functional manner. Therefore, it is natural to set the master keys of the FE scheme as only the ABEs' msk and mpk. The total number of ABE keys to be sampled is determined by the length of the FHE ciphertext.

$\underline{\mathsf{FE.Setup}(1^\lambda, n, \ell)}$:
$\mathsf{msk} \leftarrow \emptyset$
$\mathsf{mpk} \leftarrow \emptyset$
for $i \leftarrow 1$ **to** ℓ:
  $(\mathsf{mpk}_i, \mathsf{msk}_i) \leftarrow\!\!\$\ \mathsf{ABE}_2.\mathsf{Setup}(1^\lambda)$
  $\mathsf{mpk} \leftarrow \mathsf{mpk} \cup \mathsf{mpk}_i$
  $\mathsf{msk} \leftarrow \mathsf{msk} \cup \mathsf{msk}_i$
return $(\mathsf{msk}, \mathsf{mpk})$

$\underline{\mathsf{FE.Enc}(\mathsf{mpk}, \mathsf{m})}$:
$(\mathsf{hpk}, \mathsf{hsk}) \leftarrow\!\!\$\ \mathsf{FHE.Setup}(1^\lambda)$
for $i \leftarrow 1$ **to** n:
  $\Psi_i \leftarrow\!\!\$\ \mathsf{FHE.Enc}(\mathsf{hpk}, \mathsf{m}_i)$
$\Psi \leftarrow (\Psi_1, \ldots, \Psi_n)$ // Note that $|\Psi| = \ell$
$(\Gamma, L_1^0, L_1^1, \ldots, L_\ell^0, L_\ell^1) \leftarrow\!\!\$\ \mathsf{GS.Garble}(\mathsf{FHE.Dec}(\mathsf{hsk}, \cdot))$
for $i \leftarrow 1$ **to** ℓ:
  $c_i \leftarrow\!\!\$\ \mathsf{ABE}_2.\mathsf{Enc}(\mathsf{mpk}_i, (\mathsf{hpk}, \Psi), L_i^0, L_i^1)$
$\mathsf{CT} \leftarrow (\Gamma, c_1, \ldots, c_\ell)$
return CT

$\underline{\mathsf{FE.KGen}(\mathsf{msk}, f)}$:
$\mathsf{sk}_f \leftarrow \emptyset$
for $i \leftarrow 1$ **to** ℓ:
  $\mathsf{sk}_i \leftarrow\!\!\$\ \mathsf{ABE}_2.\mathsf{KGen}(\mathsf{msk}_i, \mathsf{FHE.Eval}_f^i)$
  $\mathsf{sk}_f \leftarrow \mathsf{sk}_f \cup \mathsf{sk}_i$
return sk_f

$\underline{\mathsf{FE.Dec}(\mathsf{sk}_f, \mathsf{CT})}$:
$(\Gamma, c_1, \ldots, c_\ell) \leftarrow \mathsf{CT}$
for $i \leftarrow 1$ **to** ℓ:
  $L_i^{d_i} \leftarrow \mathsf{ABE}_2.\mathsf{Dec}(\mathsf{sk}_i, c_i)$
return $\mathsf{GS.Eval}(\Gamma, L_1^{d_1}, \ldots, L_\ell^{d_\ell})$

Fig. 7. In this section, ℓ stands for the FHE's ciphertext's length, while $\mathsf{FHE.Eval}_f^i : \mathscr{K} \times \{0,1\}^{n \cdot \ell} \to \{0,1\}$ stands for a function that applies $\mathscr{C}_f$ on the encrypted input.

B.2 Attribute-Based Encryption

When we consider a *key-policy* setting, a decryption key of an ABE must be generated for one Boolean predicate $P : \{0,1\}^\lambda \to \{0,1\}$. A ciphertext of an ABE in this setting is the encryption of a set of attributes α over $\{0,1\}^\lambda$ and of some plaintext $\mathsf{m} \in \{0,1\}^\gamma$. ABE's correctness specifies that having a decryption key enables to recover the plaintext as long as $P(\alpha) = 1$.

Instantiation of ABE. The seminal work of Gorbunov *et al.* [21] puts forward attribute-based encryption schemes for comprehensive classes of circuits. We review their construction, as it will serve in the circuit complexity analysis for this work. Our description is top-down: we describe the ABE scheme, and then review the TOR framework (their Two-to-One Recoding scheme).

Attribute-Based Encryption from General Circuits. A key-policy ABE is presented in [21]. The main idea consists in evaluating on the fly a given circuit. The bitstring representing the attributes – say α – is known *a priori*, as well as the topology of the circuit – say ϕ – to be evaluated.

For each bit α_i in α, there are two public keys associated – say $(\mathsf{mpk}_i^0, \mathsf{mpk}_i^1)$ – corresponding to 0 and 1. A vector $\mathbf{s} \in \mathbb{F}_q^m$ is sampled uniformly at random, and encoded under the $\mathsf{mpk}_i^{\alpha_i}$ as $\mathsf{mpk}_i^{\alpha_i} \cdot \mathbf{s} + \text{noise}$. Then, the circuit ϕ is evaluated on these encodings. The crux point consists of a **recoding** procedure, which ensures that at the next level, $\mathbf{s}$ is "recoded" under the next public key corresponding to the current gate. By keeping evaluating in such a way, the final output will be an encoding of $\mathbf{s}$ under a circuit-dependent key pk_{out}. The encoding of the form $\mathsf{pk}_{out} \cdot \mathbf{s} + \text{noise}$ is then used to recover the (symmetrically-)encrypted input X. We detail these procedures in what follows:

– $\mathsf{Setup}(1^\lambda)$: consists of ℓ pairs of public keys, where ℓ is the length of the supported attributes α:

$$\begin{pmatrix} \mathsf{mpk}_1^0 & \mathsf{mpk}_2^0 & \dots & \mathsf{mpk}_\ell^0 \\ \mathsf{mpk}_1^1 & \mathsf{mpk}_2^1 & \dots & \mathsf{mpk}_\ell^1 \end{pmatrix}$$

An additional key mpk_{out} is sampled. Concretely, each mpk_i^b corresponds to $\mathbf{A}_i^b \in \mathbb{Z}_q^{n \times m}$. The master secret key consists of $2 \cdot n$ trapdoor matrices, which are described in the TOR subsection (see below).

– $\mathsf{KeyGen}(\mathsf{msk}, \phi)$: considering the circuit representation of $\phi : \{0,1\}^n \to \{0,1\}$. Each wire in the circuit is associated with two public keys, corresponding to a 0 and a 1. For each gate $g_{u,v,w}$, a table consisting of 4 recoding keys are generated: $rk_{g(\alpha,\beta)}^w$ for $g_{\alpha,\beta}^w$ the value of the gate under inputs $\alpha, \beta \in \{0,1\}$.
Based on the value of the gate applied on the inputs received from the attribute (which is known in plain) a recoding key is chosen. This recoding key is then used to recode the value of $\mathbf{s}$ under the new public key.

– $\mathsf{Enc}(\mathsf{mpk}, X, \alpha)$: encrypting X means sampling a random vector $\mathbf{s} \leftarrow_\$ \mathbb{F}_q^m$ and based on the decomposition of α, obtaining the encodings of $\mathbf{s}$ under $\mathsf{mpk}_i^{\alpha_i}$.
Finally, the input X itself is encrypted – via a semantic secure symmetric scheme – under $\mathsf{Encode}(\mathsf{mpk}_{out}, s)$, which acts as a key. Thus, the ciphertext consists of $\left(\alpha, \left\{\mathsf{Encode}(\mathsf{mpk}_i^{\alpha_i} \cdot \mathbf{s} + e_i)\right\}_{i=1}^n, \mathsf{SE.Enc}(\mathsf{Encode}(\mathsf{mpk}_{out}, s), X)\right)$.

– $\mathsf{Dec}(\mathsf{CT}, \mathsf{sk}_\phi)$: the decryption procedure evaluates the circuit given the encodings and according to the attributes, and recovers $\mathsf{Encode}(\mathsf{pk}_{out}, s)$. This is then used to recover X.

Two-To-One Recodings. The beautiful idea in [21] stems in the Two-To-One Recoding mechanism. The crux point is to start with two LWE tuples of the form

$$\mathbf{A}_1 \cdot \mathbf{s} + \mathbf{e}_1 \text{ and } \mathbf{A}_2 \cdot \mathbf{s} + \mathbf{e}_2$$

and "recode" them under a new "target" matrix $\mathbf{A}_{tgt}$. The outcome is indeed a recoding of $\mathbf{s}$: $\mathbf{A}_{tgt} \cdot \mathbf{s} + \mathbf{e}_{tgt}$. In doing so, the recoding mechanism uses two matrices, $\mathbf{R}_1, \mathbf{R}_2$, such that

$$\mathbf{A}_1 \cdot \mathbf{R}_1 + \mathbf{A}_2 \cdot \mathbf{R}_2 = \mathbf{A}_{tgt} .$$

Sampling $\mathbf{R}_1$ is done uniformly at random. $\mathbf{R}_2$ is sampled from an appropriate distribution, depending on a trapdoor matrix $\mathbf{T}$. We do not discuss the details of this scheme's correctness/security, as our interest is related to the efficiency of its *encryption* procedure.

Yao's Garbling Scheme [31]. Garbling schemes have been introduced by Yao [31]. A much appreciated way of garbling circuits is in fact the original proposal by Yao. He considers a family of circuits having k input wires and producing one bit. In this setting, circuit's secret key is regarded as two labels (L_i^0, L_i^1) for each input wire, where $i \in [k]$. The evaluation of the circuit at point x corresponds to an evaluation of $\mathsf{Eval}(\Gamma, (L_1^{x_1}, \dots, L_k^{x_k}))$, where x_i stands for the i^{th} bit of x — thus the encoding $c = (L_1^{x_1}, \dots, L_k^{x_k})$. The garbled circuit Γ can be produced gate by gate, and the labels can be in fact symmetric keys.

B.3 Fully Homomorphic Encryption.

Fully homomorphic encryption (FHE) has been described within the work of Rivest, Adleman and Dertouzos; it was an open problem until the breakthrough work of Gentry [16].

Instantiation of FHE using the GSW levelled FHE. For the sake of clarity we instantiate the FHE component used in [19] (see Figure 7) using the GSW [17] fully homomorphic encryption scheme.

- $\mathsf{GSW.Setup}(1^\lambda, 1^d)$:
  Given the LWE parameters (q, n, χ), set $m := n\log(q)$. Let $N := (n+1)\cdot(\lfloor\log(q)\rfloor + 1)$. Sample
  $$\mathbf{t} \leftarrow \mathbb{Z}_q^n.$$
  Set
  $$\mathsf{hsk} := \mathbf{s} \leftarrow (1, -\mathbf{t}_1, \ldots, -\mathbf{t}_n) \in \mathbb{Z}_q^{n+1}.$$
  Generate $\mathbf{B} \leftarrow_\$ \mathbb{Z}_q^{m\times n}$ and $\mathbf{e_B} \leftarrow_\chi \chi^m$. Set $\mathbf{b} \leftarrow \mathbf{B}\cdot\mathbf{t} + \mathbf{e_B}$.
  Let $\mathbf{A}$ be defined as the $m \times (n+1)$ matrix having $\mathbf{B}$ in its last n columns, preceded by $\mathbf{b}$ as the first one.
  Set $\mathsf{hpk} \leftarrow \mathbf{A}$. Return $(\mathsf{hpk}, \mathsf{hsk})$.
- $\mathsf{GSW.Enc}(\mathsf{hpk}, \mu)$:
  to encrypt a bit μ, first sample $\mathbf{R} \leftarrow_\$ \{0,1\}^{N\times m}$.
  Return as ciphertext: $\mathsf{CT} \leftarrow \mathsf{Flatten}\left(\mu\cdot \mathsf{I}_N + \mathsf{BitDecomp}(\mathbf{A}\cdot\mathbf{R})\right) \in \mathbb{Z}_q^{N\times N}$.
- $\mathsf{GSW.Dec}(\mathsf{CT}, \mathsf{hsk})$:
  Let $\mathbf{v} \leftarrow \mathsf{PowersOfTwo}(\mathbf{s})$.
  Find the index i such that $\mathbf{v}_i = 2^i \in (\frac{q}{4}; \frac{q}{2}]$.
  Compute $\mathbf{x}_i \leftarrow \mathsf{CT}_i \cdot \mathbf{v}$, with CT_i the i^{th} row of CT.
  Return $\mu' \leftarrow \lfloor \frac{\mathbf{x}_i}{\mathbf{v}_i} \rceil$.

We do not discuss the circuit evaluation procedure, because it plays no role in FE's encryption procedure.

B.4 Parallel Complexity of [19]'s encryption procedure when instantiated with GSW13 and GVW13

In this part, we provide an analysis of the parallel complexity of [19]'s encryption procedure when instantiated with GSW13 and GVW13. First, we look at the ciphertext structure in Figure 7. It consists of two main types of elements: i) ABE ciphertexts and ii) a garbled circuit.

The ABE ciphertext. We do not describe the two outcome ABE, but note it can be obtained generically from an ABE in the key-policy setting. The ciphertext structure is described above, and it consists itself of two parts:

1. **Index-Encodings:** According to our notations, α is an index. Based on the index's position α_i, one of the public key (matrices) is selected. Logically, this ciphertext component translates to:
   $$\left(\mathbf{A}_i^0 \cdot \mathbf{s} + \mathbf{e}_i\right) + \alpha_i \cdot \left(\mathbf{A}_i^1 - \mathbf{A}_i^0\right)\cdot \mathbf{s} \tag{40}$$
   Here α_i is an index, but for [19], such indexes are generated through the hpk and the homomorphic ciphertext. Thus we complement Equation (40) with two further subcases.
   - part of indexes will be the homomorphic public key, which consists of either i) the elements of $\mathbf{B}$ or ii) of a vector
     $$\alpha_i \leftarrow (\mathbf{B}\cdot\mathbf{t} + \mathbf{e_B})_\theta \tag{41}$$
     where θ denotes a bit in the binary representation of the above quantity.
     When plugged in with Equation (41), Equation (40) becomes:
     $$\left(\mathbf{A}_i^0 \cdot \mathbf{s} + \mathbf{e}_0\right) + (\mathbf{B}\cdot\mathbf{t} + \mathbf{e_B})_\theta \cdot \left(\mathbf{A}_i^1 - \mathbf{A}_i^0\right)\cdot \mathbf{s} \tag{42}$$
     The circuit to compute that quantity can be realized by several NC^1 circuits, the inner one outputting a bit in position θ, the outer one outputting one bit of ciphertext. As a consequence of [4], we assume that LWE-like tuples can be computed in NC^1. When plugged in with elements of $\mathbf{B}$ from case i), the equation is simpler, and we simply assume the circuit computing it has its depth lower than or equal to the previously mentioned circuit.

– The second subcase is related to the usage of homomorphic ciphertexts as indexes for GVW13. The ciphertext of GSW13 has the following format:

$$\mathsf{Flatten}\left(\mathsf{inp}_\xi \cdot \mathsf{I}_N + \mathsf{BitDecomp}(\mathbf{A}\cdot\mathbf{R})\right) \tag{43}$$

where inp_ξ is a real input for the FE schema and $\mathbf{R}$ is a random matrix.
As for the previous case, a boolean circuit can compute Equation (43) in logarithmic depth. The BitDecomp has essentially constant-depth, as it does rewiring. The matrix multiplication can be computed, element-wise in logarithmic depth (addition can be done in a tournament style, while element multiplication over $\mathbb{F}_q$ can also be performed in logarithmic time). The flattening part can be performed in $\log\log(q+1)+1$ [25].
Once Equation (40) is fed with Equation (43), the size of the circuit will still be logarithmic, as the outer circuit, computing the matrix sum can be highly parallelized.

2. **Label-Encodings:**
The second part of the GVW13 ciphertext is the encoding of the message itself. We analyse the format of these encodings, and also the message to be encoded (a label of a garbled circuit).
The encoding is done in two layers: first, a component within a classical LWE tuple

$$\mathbf{A}_{out}\cdot\mathbf{s}+\mathbf{e} \tag{44}$$

is obtained, which is then used to key a symmetric encryption scheme that will encode the input. We do not analyse the circuit's depth of the SE, but we will assume it is in NC^1, and the Equation (44) can be performed in NC^1, as we assume the existence of one-way functions in NC^1. Their composition is:

$$\mathsf{SE.Enc}((\mathbf{A}_{out}\cdot\mathbf{s}+\mathbf{e}), X) \tag{45}$$

Thus, the composition of these two families of circuits will be in NC^1, as long as obtaining X is in NC^1.
We turn to the problem of populating X. As we use Yao's garbling scheme, X will simply be itself a secret key of a symmetric scheme, used by a garbling table. Thus, generating X can be done by a low depth PRG in NC^1.

The Garbled Circuit. The final part of FE's ciphertext in [19] is the garbled circuit, which uses Yao's garbling. The garbled circuit can be obtained gate by gate. The circuit to be garbled is GSW13's decryption. This decryption procedure consists of an inner product, followed by a division with a predefined value (in fact a power of two), and by a rounding. The total circuit complexity is logarithmic (thus NC^1).

We now inspect the complexity of the circuit producing the garbling of the gates of FHE.Dec. It is clear that every gate garbling process can be parallelized: the structure of FHE's decryption circuit is fixed, enough labels must be sampled (by NC^1 circuits).

For each wire in a gate, there must be one SE key generated. After that, producing one garbling table has the same depth as the encryption circuit together with the SE's key generation procedure. Given that these components are in NC^1, the complexity of the combined circuit is in NC^1.

C FE with Encryption in NC^1 from RLWE

Overview. This section investigates a construction of functional encryption supporting circuits of a *bounded* depth d, a bounded width w (in direct relation to their size), and representing functions believed to be one-way. The data-dependent part of the construction enjoys succinctness, while the data-independent part grows quadratically within the number of supported functions. Luckily, we only need to support a single function.

We begin by introducing the simpler concept of linear functional encryption. Next, we closely look into a specific *decomposable* FE scheme for NC^1 circuits[13], namely the one introduced by Agrawal and Rosen in [2], due to

[13] From now on, when referring to [2] we mean their NC^1 construction.

the succinctness in the size of its output and the simplicity of the ciphertext's structure. This description is followed by a digression on the normal coefficient embedding and the CRT coefficient embedding by pointing out the advantages conferred by the latter embedding.

The [2] Construction for NC^1. In [2], Agrawal and Rosen introduced a novel construction for functional encryption supporting comprehensive classes of circuits, thus building up on the works of [19,1]. As a prime element of novelty, the supported class of functions are now described by arithmetic, rather than Boolean circuits[14]. Second, the size of the ciphertexts in their scheme is succinct, growing with the depth of the circuit, rather than its size. Third, and most importantly for the problem at hand, the ciphertext is decomposable: assuming a plaintext is represented as a vector of dimension w, each of the w elements is encrypted independently. Therefore, replacing bits in the plaintext requires partial changes in the ciphertext.

How to read this section. This section provides, is in fact, an alternative FE having encryption in NC^1 compared to the one described in Appendix B. The FE scheme for NC^1 circuits in [2] depends on functional encryption for inner products in a semi-generic fashion. The ciphertext structure needs to compute *nested* "Regev encodings", having the required nesting level corresponding to the *multiplicative* depth of the circuit. To keep things comprehensible, we describe, first, the interface of a decomposable Lin-FE scheme, and after introduce the scheme for NC^1 circuits.

C.1 Warm-Up: Decomposable Linear Functional Encryption

Functional encryption schemes for linear functionalities (or inner products) encrypt vectors $\mathbf{x}$ and return functional keys for vectors $\mathbf{y}$. The decryption procedure recovers the inner-product $\langle \mathbf{x}, \mathbf{y} \rangle$.

- $\mathsf{Lin\text{-}FE.Setup}(1^\lambda)$: assume the message space contains elements represented by vectors $\mathbf{x} = (x_1, \ldots, x_w)$ of dimension n defined over $\mathscr{R}_p^w$, where the algebraic ring $\mathscr{R}_p$ is defined modulo a large p – a prime in our case. To keep notation compact, we work with row-vectors, rather than the standard column vectors. The master public key is of the form
$$\mathsf{mpk} \leftarrow \left(\mathsf{pk}_1, \ \ldots, \ \mathsf{pk}_w, \mathsf{pk}_{ind}\right),$$
where pk_i stands for the public-key of a PKE scheme. The msk consists of the corresponding secret keys. $\mathsf{Lin\text{-}FE.Setup}(1^\lambda)$ returns the two master keys.

- $\mathsf{Lin\text{-}FE.Enc}(\mathsf{mpk}, \mathbf{x})$: a sufficiently large random tape R is used; then, for each message x_i, a ciphertext is computed as
$$\mathsf{CT}_i \leftarrow_\$ \mathsf{PKE.Enc}(\mathsf{pk}_i, x_i; R).$$
Finally, $\mathsf{CT}_{ind} \leftarrow_\$ \mathsf{Com}(\mathsf{pk}_{ind}, R)$ is a commitment to the random coins used by the scheme. The following ciphertext is returned:
$$\left(\mathsf{CT}_1, \ldots, \mathsf{CT}_w, \mathsf{CT}_{ind}\right).$$

- $\mathsf{Lin\text{-}FE.KGen}(\mathsf{mpk}, \mathsf{msk}, \mathbf{y})$: given a vector $\mathbf{y}$, the key generation algorithm does the following
  1. Compute $\mathsf{pk}_\mathbf{y} \leftarrow_\$ \mathsf{Eval}_{\mathsf{PK}}(\mathsf{pk}_1, \ldots, \mathsf{pk}_w, \mathbf{y})$. The $\mathsf{pk}_\mathbf{y}$ should be regarded as a "functional public-key component". $\mathsf{Eval}_{\mathsf{PK}}$ is a publicly available procedure.
  2. Set $K_\mathbf{y} \leftarrow_\$ \mathsf{GenKey}(\mathsf{mpk}, \mathsf{msk}, \mathsf{pk}_\mathbf{y})$. GenKey is the procedure responsible for generating the functional key, based on $\mathsf{mpk}, \mathsf{msk}$ and $\mathsf{pk}_\mathbf{y}$.

  The functional key $\mathsf{sk}_f := K_\mathbf{y}$ is returned.

- $\mathsf{Lin\text{-}FE.Dec}(\mathsf{sk}_f, \mathsf{CT})$: the decryption first computes a "functional ciphertext" through a public procedure $\mathsf{Eval}_{\mathsf{CT}}$
$$\mathsf{CT}_{\langle \mathbf{y}, \mathbf{x} \rangle} \leftarrow_\$ \mathsf{Eval}_{\mathsf{CT}}\left((\mathsf{CT}_1, \ldots, \mathsf{CT}_w), \mathbf{y}\right).$$
The procedure returns $\mathsf{Decode}(K_\mathbf{y}, \mathsf{CT}_{\langle \mathbf{y}, \mathbf{x} \rangle}, \mathsf{CT}_{ind})$.

[14] Any Boolean circuit can be simulated by an arithmetic circuit over GF(2).

Correctness, in layman terms, should guarantee that the output corresponds to $\langle \mathbf{y}, \mathbf{x} \rangle$ assuming correctly generated ciphertexts and functional keys.

Structural properties of the ciphertext. From a high level point of view, we require that such a scheme attains the following *structural* requirements:

1. **Malleability:** for any x_i, x_j in the message space and for any valid CT_{x_i} it holds that: $\mathsf{CT}_{x_i} + x_j = \mathsf{CT}_{x_i + x_j}$.

2. **Succinctness:** the size of the ciphertext is bounded by a polynomial in security parameter and input length $|\mathsf{CT}_{\mathbf{x}}| \in \mathcal{O}(poly(\lambda), |\mathbf{x}|)$.

3. **Decomposability:** informally, for an FE scheme supporting messages of length w, its ciphertexts and public-key can be decomposed into $w+1$ components such that each component corresponds to a single message in the plaintext vector. That is, the ciphertext and the master public-key can be parsed as:

$$\mathsf{CT}_{\mathbf{x}} \leftarrow \left(\mathsf{CT}_1, \ldots, \mathsf{CT}_w, \mathsf{CT}_{ind}\right), \qquad \mathsf{mpk} \leftarrow \left(\mathsf{pk}_1, \ldots, \mathsf{pk}_w, \mathsf{pk}_{ind}\right).$$

   Additionally, one may write that:

$$\mathsf{CT}_i \leftarrow_{\$} \mathsf{PKE.Enc}(\mathsf{pk}_i, x_i; R).$$

C.2 Regev Encodings

First, we informally recall the simple symmetric encryption scheme presented in [12]. Here, "s" stands for a RLWE secret acting as a secret key, while a and r are the random mask and noise:

$$\begin{aligned} c_1 &\leftarrow a && \in \mathscr{R}_p \\ c_2 &\leftarrow a \cdot s + 2 \cdot r + x && \in \mathscr{R}_p \end{aligned} \tag{46}$$

Recovering the plaintext bit x is done by subtracting $c_1 \cdot s$ from c_2 and reducing modulo 2. Such a simple scheme exhibits powerful homomorphic properties, that are speculated in [2]. From now on, we call "Regev encoding" the mapping between rings $\mathscr{E}^i : \mathscr{R}_{p_{i-1}} \to \mathscr{R}_{p_i}$ such that

$$\mathscr{E}^i(x) = a^i \cdot s + p_{i-1} \cdot e^i + x \ \in \mathscr{R}_{p_i}. \tag{47}$$

C.3 FE for NC^1

We provide an overview of the construction in [2] for NC^1 circuits. We refer the reader to its original description for complete details.

– **Encryption.** The encryption algorithm starts by sampling a RLWE secret s, and "encoding" each input $x_i \in \mathscr{R}_{p_0}$ independently. The result is

$$\left\{\mathscr{E}^1(x_i) | x_i \in \mathscr{R}_{p_0} \wedge i \in [n]\right\},$$

   where $\mathscr{E}^1$ is the map $\mathscr{E}^1 : \mathscr{R}_{p_0} \to \mathscr{R}_{p_1}$ defined in Equation (47). This represents the "Level 1" encoding of x_i. Next, the construction proceeds recursively; the encoding of x_i at "Level 2", takes the *parent* node P (in this case P is $\mathscr{E}^1(x_i)$), and obtains for the left branch:

$$\mathscr{E}^2(P) = \mathscr{E}^2(\mathscr{E}^1(x_i)) = a^2_{1,i} \cdot s + p_1 \cdot e^2_{1,i} + (a^1_{1,i} \cdot s + p_0 \cdot e^1_{1,i} + x_i) \in \mathscr{R}_{p_2} \tag{48}$$

   and for the right branch:

$$\mathscr{E}^2(P \cdot s) = \mathscr{E}^2(\mathscr{E}^1(x_i) \cdot s) = a^2_{2,i} \cdot s + p_1 \cdot e^2_{2,i} + (a^1_{1,i} \cdot s + p_0 \cdot e^1_{1,i} + x_i) \cdot s \in \mathscr{R}_{p_2} \tag{49}$$

   for some $(a^2_{1,i}, a^2_{2,i}) \leftarrow_{\$} \mathscr{R}^2_{p_1}$ and noise terms $(e^2_{1,i}, e^2_{2,i}) \leftarrow_{\chi} \mathscr{R}_{p_2}$. As a general rule, we will write as an *upperscript* of a variable the *level* to which it has been associated. The procedure repeats recursively up to a number of levels d, as depicted in Figure 8.

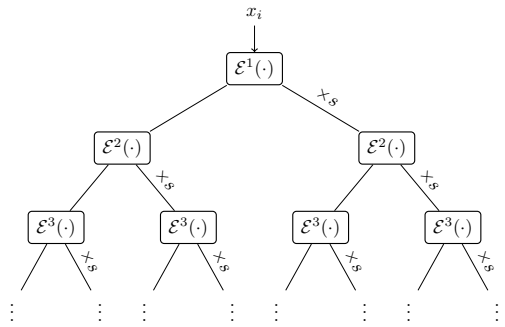


Fig. 8. The tree obtained from encoding x_i in a recursive manner.

We also mention that between any two successive (multiplication) layers, there is an *addition* layer, which replicates the ciphertext in the precedent multiplication layer (and uses its modulus). As it brings no new information, we ignore additive layers from our overview.

The encoding procedure above applies for each $x_1, \ldots, x_n$, obtaining a ciphertext resembling a "forest of trees" (Figure 9). In addition, Level 1 also contains an encoding of s, i.e., $\mathscr{E}^1(s)$, while Level i (for $2 \leq i \leq d$) also contains $\mathscr{E}^i(s^2)$. Up to this point, the technique used by the scheme resembles the ones used in fully homomorphic encryption. The high level idea is to compute the function f obliviously with the help of the encodings.

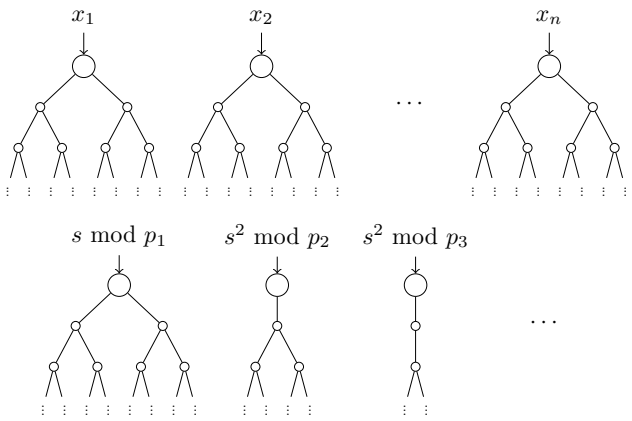


Fig. 9. A high level view of the encodings: each input x_i is recursively encrypted, the resulting ciphertext having a "tree-like" structure. Circular encryptions of s are also provided. The "additive" layers are excluded from the picture.

Notation. We will refer to such ciphertext encoding input bits x_i as "Type 1" ciphertexts (message-dependent), while the ones encoding the secret as being "Type 2" ciphertexts.

– Still, as we are in the FE setting (as opposed to an FHE setting), the ciphertext also contains additional information on s, through the use of a linear functional encryption scheme Lin-FE. Namely, an extra component of the form:

$$\mathbf{d} \leftarrow \mathbf{w} \cdot s + p_{d-1} \cdot \eta \tag{50}$$

is provided as CT_{ind}, where $\eta \leftarrow_{\chi} \mathscr{R}_{p_d}$ denotes a noisy term and $\mathbf{w}$ is part of mpk. We will refer to this component from Equation (50) as a"Type-3" ciphertext.

– The **master secret key** consists of the Lin-FE.msk. The **master public key** consists of the Lin-FE.mpk (the vector $\mathbf{w}$ and the vector $\mathbf{a}^d$) as well as of the set of vectors $\{\mathbf{a}^1, \mathbf{a}^2, \ldots, \mathbf{a}^{d-1}\}$ that will be used by each $\mathscr{E}^i$. Once again, we *stress* that the vector $\mathbf{a}^d$ from Lin-FE.mpk coincides with the public labelling used by the mapping $\mathscr{E}^d$. It can be immediately shown that the size of $\mathbf{a}^{i+1}$ is given by a first order recurrence:

$$L^{i+1} = |\mathbf{a}^{i+1}| = 2 \cdot |\mathbf{a}^i| + 1 \tag{51}$$

with the initial term (the length of $\mathbf{a}^1$) set to the length of the input. The extra term 1 added per each layer is generated by the extra encodings of the key-dependent messages $\{\mathscr{E}^1(s), \mathscr{E}^2(s^2), \ldots, \mathscr{E}^d(s^2)\}$.

– The **functional-key** sk_f is issued via the Lin-FE algorithm as follows: first, based on the circuit representing f and on the public set of $\{\mathbf{a}^1, \mathbf{a}^2, \ldots, \mathbf{a}^d\}$, a public value $\mathsf{PK}_f \leftarrow \mathsf{Eval}_{\mathsf{PK}}(\mathsf{mpk}, f)$ is computed (by performing f-dependent arithmetic combinations of the values in $\{\mathbf{a}^1, \mathbf{a}^2, \ldots, \mathbf{a}^d\}$). Then, a functional key sk_f is issued for PK_f.

– The $\mathsf{Eval}_{\mathsf{PK}}(\mathsf{mpk}, f)$ procedure uses mpk to compute PK_f. In a similar way, $\mathsf{Eval}_{\mathsf{CT}}(\mathsf{mpk}, \mathsf{CT}, f)$ computes the value of the function f obliviously on the ciphertext. We refer the reader to [2] for complete explanations but provide a short overview of these procedures in Appendix C.6.

– **Decryption** is done by computing the circuit for f (known in plain by the decryptor) over the encodings. At level d, the ciphertext will have the following structure:

$$\mathsf{CT}_{f(\mathbf{x})} \leftarrow \mathsf{PK}_f \cdot s + \sum_{i=0}^{d-1} p_i \cdot \eta^{i+1} + f(\mathbf{x}) \,. \tag{52}$$

Next, based on the independent ciphertext $\mathbf{d} \leftarrow \mathbf{w} \cdot s + p_{d-1} \cdot \eta$ and on the functional key, the decryptor recovers

$$\mathsf{PK}_f \cdot s + p_{d-1} \cdot \eta' \,. \tag{53}$$

Finally, $f(\mathbf{x})$ is obtained by subtracting (53) from (52) and repeatedly applying the *mod* operator to eliminate the noise: $(\mathrm{mod}\ p_{d-1}) \ldots (\mathrm{mod}\ p_0)$.

A more compact description is given in Appendix C.5.

C.4 Ciphertext and Public-Key Structure

In the ePrint version of [2, Theorem 5.2], the authors show that the ciphertext exhibits a particular structure of the form:

$$\mathsf{CT}_{f(x)} = P_f(\vec{\mathsf{CT}}^1, \ldots \vec{\mathsf{CT}}^{d-1}) + Lin_f(\vec{\mathsf{CT}}^d) \,. \tag{54}$$

where P_f is a high degree multivariate polynomial in the given encodings $\vec{\mathsf{CT}}^1, \ldots \vec{\mathsf{CT}}^{d-1}$, plus a linear combination of the top level encodings.

C.5 A Compact Description for NC1 circuits

– $(\mathsf{msk}, \mathsf{mpk}) \leftarrow_\$ \mathsf{FE.Setup}(1^\lambda, 1^w, 1^d)$: let d stand for the circuit depth, w stand for the length of the supported inputs and λ for the security parameter.

$$\begin{gathered} \textbf{for } k \leftarrow 1, \ldots, (d-1): \\ \mathbf{a}^k \leftarrow_\$ \mathscr{R}_{p_k}^{L^k} \end{gathered} \tag{55}$$

Here $\{p_k : k \leftarrow 1, \ldots, d\}$ stands for a set of d primes, while L^k denotes the size of an encoding (we assume they are a priori known). Then, a Lin-FE scheme is instantiated:

$$(\mathsf{pk}, \mathsf{msk}) \leftarrow_\$ \mathsf{Lin\text{-}FE.Setup}(1^\lambda)$$

where $\mathsf{pk} \leftarrow (\mathbf{w}, \mathbf{a}^d)$. The following variables are returned:

$$\begin{aligned} \mathsf{mpk} &\leftarrow (\mathbf{a}^1, \ldots, \mathbf{a}^d, \mathbf{w}) \\ \mathsf{msk} &\leftarrow \mathsf{Lin\text{-}FE.msk} \end{aligned} \tag{56}$$

– $\mathsf{FE.KGen}(\mathsf{msk}, \mathsf{m})$: given a function f represented as a circuit of depth d:

$$\mathsf{PK}_f \leftarrow_\$ \mathsf{Eval}_{\mathsf{PK}}(\mathsf{mpk}, f) .$$

Invoke KeyGen to obtain K_f[15] and return it:

$$K_f \leftarrow_\$ \mathsf{Lin\text{-}FE.KGen}(\mathsf{msk}, \mathsf{PK}_f) .$$

– $\mathsf{FE.Enc}(\mathsf{mpk}, \mathsf{m} = [x_1, \ldots, x_w])$: first, for each x_i, compute its encodings for all multiplicative levels $1 \to d$:

$$\mathsf{CT}_i^k \leftarrow_\$ \mathscr{E}^k(x_i),\ \forall k \in [d] . \tag{57}$$

Then, set $\mathbf{d}$ as follows:

$$\mathbf{d} \leftarrow \mathbf{w} \cdot s + \mu .$$

Finally, set the ciphertext corresponding to m as $\mathsf{CT}_{\mathsf{m}} \leftarrow \left(\{\mathsf{CT}^k\}_{k\in[d]}, \mathbf{d}\right)$.

– $\mathsf{FE.Dec}(\mathsf{sk}_f, \mathsf{CT})$: compute

$$\mathsf{CT}_{f(x)} \leftarrow \mathsf{Eval}_{\mathsf{CT}}(\{\mathsf{CT}^k\}_{k\in[d]}, f) .$$

Return $\mathrm{Decode}(K_f, \mathsf{CT}_{f(x)})$.

C.6 Ciphertext and Public-Key Evaluation

The $\mathsf{Eval}_{\mathsf{PK}}(\mathsf{mpk}, f)$ procedure uses mpk to compute PK_f. In a similar way, $\mathsf{Eval}_{\mathsf{CT}}(\mathsf{mpk}, \mathsf{CT}, f)$ computes the value of the function f obliviously on the ciphertext. Both the procedure are defined recursively, that is to compute PK_f^k and $\mathsf{CT}_{f(\mathbf{x})}^k$ at level k, PK_f^{k-1} and $\mathsf{CT}_{f(\mathbf{x})}^{k-1}$ are needed. For a better understanding of the procedures, we will denote the encoding of $f^k(\mathbf{x})$ by c^k, i.e. $c^k = \mathscr{E}^k(f^k(\mathbf{x}))$ and the public key or label of an encoding $\mathscr{E}^k(\cdot)$ by $\mathsf{PK}(\mathscr{E}^k(\cdot))$. Thus, formally, the two procedures are recursively defined as follow:

$\mathsf{Eval}_{\mathsf{PK}}(\cup_{t\in[k]}\mathsf{CT}^t, \ell)$ computes the label for the ℓ^{th} wire in the k level circuit, from the i^{th} and j^{th} wires of $k-1$ level:

1. Addition Level:
   – If $k = 1$ (base case): $\mathsf{PK}(c_\ell^1) \leftarrow \mathsf{PK}_i + \mathsf{PK}_j$.
   – Otherwise, $u_i^{k-1} \leftarrow \mathsf{Eval}_{\mathsf{PK}}^{k-1}(\cup_{t\in[k-1]}\mathsf{CT}^t, i)$ and $u_j^{k-1} \leftarrow \mathsf{Eval}_{\mathsf{PK}}^{k-1}(\cup_{t\in[k-1]}\mathsf{CT}^t, j)$. Compute $\mathsf{PK}(c_\ell^k) \leftarrow u_i^{k-1} + u_j^{k-1}$.
2. Multiplication Level:
   – If $k = 2$ (base case): $\mathsf{PK}(c_\ell^2) \leftarrow u_i^1 u_j^1 \mathsf{PK}(\mathscr{E}^2(s^2)) - u_j^1 \mathsf{PK}(\mathscr{E}^2(c_i^1 s)) - u_i^1 \mathsf{PK}(\mathscr{E}^2(c_j^1 s))$.
   – Otherwise, $u_i^{k-1} \leftarrow \mathsf{Eval}_{\mathsf{PK}}^{k-1}(\cup_{t\in[k-1]}\mathsf{CT}^t, i)$ and $u_j^{k-1} \leftarrow \mathsf{Eval}_{\mathsf{PK}}^{k-1}(\cup_{t\in[k-1]}\mathsf{CT}^t, j)$.
     Compute:

$$\mathsf{PK}(c_\ell^k) \leftarrow u_i^{k-1} u_j^{k-1} \mathsf{PK}(\mathscr{E}^k(s^2)) - u_j^{k-1} \mathsf{PK}(\mathscr{E}^k(c_i^{k-1} s) - u_i^{k-1} \mathsf{PK}(\mathscr{E}^k(c_j^{k-1} s)) .$$

$\mathsf{Eval}_{\mathsf{CT}}(\cup_{t\in[k]}\mathsf{CT}^t, \ell)$ computes the encoding of the ℓ^{th} wire in the k level circuit, from the i^{th} and j^{th} wires of $k-1$ level:

[15] Remember that $\mathbf{w}^\top \cdot \mathbf{k}_f = \mathsf{pk}_f$.

1. Addition Level:
   - If $k = 1$ (base case): $c_\ell^1 \leftarrow \mathscr{E}^1(x_i) + \mathscr{E}^1(x_j)$.
   - Otherwise, let $c_i^{k-1} \leftarrow \mathsf{Eval}_{\mathsf{CT}}^{k-1}(\cup_{t\in[k-1]}\mathsf{CT}^t, i)$ and $c_j^{k-1} \leftarrow \mathsf{Eval}_{\mathsf{CT}}^{k-1}(\cup_{t\in[k-1]}\mathsf{CT}^t, j)$.
     Compute $\mathsf{CT}_\ell^k \leftarrow c_i^{k-1} + c_j^{k-1}$.
2. Multiplication Level:
   - If $k = 2$ (base case): $c_\ell^2 \leftarrow c_i^1 c_j^1 \mathscr{E}^2(s^2) - u_j^1 \mathscr{E}^2(c_i^1 s) - u_i^1 \mathscr{E}^2(c_j^1 s)$.
   - Otherwise, let $c_i^{k-1} \leftarrow \mathsf{Eval}_{\mathsf{CT}}^{k-1}(\cup_{t\in[k-1]}\mathsf{CT}^t, i)$ and $c_j^{k-1} \leftarrow \mathsf{Eval}_{\mathsf{CT}}^{k-1}(\cup_{t\in[k-1]}\mathsf{CT}^t, j)$.

     Compute:
$$\mathsf{CT}_\ell^k \leftarrow c_i^{k-1} c_j^{k-1} + u_i^{k-1} u_j^{k-1} \mathscr{E}^k(s^2) - u_j^{k-1} \mathscr{E}^k(c_i^{k-1} s) - u_i^{k-1} \mathsf{PK}(\mathscr{E}^k(c_j^{k-1} s)\,.$$

We refer the reader to [2] for complete explanations and worked-out base-cases.

C.7 The Normal and CRT Embeddings

By means of classical algebra, one can show that if $\{\mathscr{I}_1, \ldots, \mathscr{I}_N\} \subseteq \mathscr{R}_q$ form a set of co-prime ideals and $\mathscr{I} := \cap_{k=1}^N \mathscr{I}_k$, then $\mathscr{R}/\mathscr{I}$ is isomorphic with the product of $\mathscr{R}/$ideals via the Chinese Remainder Theorem. Stated differently, the ring $\mathscr{R}/\mathscr{I}$ is isomorphic with the direct product:

$$\mathscr{R}/\mathscr{I}_1 \times \ldots \times \mathscr{R}/\mathscr{I}_N\,. \tag{58}$$

Given two elements over $\mathscr{R}_1$ – say $a = a_0 + a_1 \cdot x + \ldots a_{n-1} \cdot x^{N-1}$ and $\underline{b} = \underline{b_0} + b_1 \cdot x + \ldots b_{n-1} \cdot x^{N-1}$, their representation to the CRT embedding becomes: $\overline{a} = (\overline{a_0}, \overline{a_1}, \ldots, \overline{a_{N-1}})$ and $\overline{b} = (\overline{b_0}, \overline{b_1}, \ldots, \overline{b_{N-1}})$ The multiplication of $a \cdot b$ is as simple as

$$\overline{a} \cdot \overline{b} = (\overline{a_0} \cdot \overline{b_0}, \overline{a_1} \cdot \overline{b_1}, \ldots, \overline{a_{N-1}} \cdot \overline{b_{N-1}})$$

Similarly to multiplication, addition is carried out component-wise:

$$\overline{a} + \overline{b} = (\overline{a_0} + \overline{b_0}, \overline{a_1} + \overline{b_1}, \ldots, \overline{a_{N-1}} + \overline{b_{N-1}})$$

To have the complete picture an element $a \in \mathscr{R}_q$ is mapped to the CRT embedding as follows:

$$\begin{aligned} a \rightarrow \Big(& (a_0 + a_1 \cdot \theta_1 + \ldots + a_{N-1} \cdot \theta_1^{N-1})/\mathscr{I}_1, \\ & \ldots \\ & (a_0 + a_1 \cdot \theta_i + \ldots + a_{N-1} \cdot \theta_i^{N-1})/\mathscr{I}_i, \\ & \ldots \\ & (a_0 + a_1 \cdot \theta_N + \ldots + a_{N-1} \cdot \theta_N^{N-1})/\mathscr{I}_N \Big) \end{aligned} \tag{59}$$

where $\theta_i := \theta^{2\cdot i-1}$ and $\theta^{2\cdot N} = 1 \bmod q$. Stated differently, θ is a primitive root of unity.

Noise in the CRT embedding. As a caveat, the noise is not "small" with respect to this CRT embedding. On the other hand, for the non-invertible elements (given that they share a common factor h of degree N-1), the CRT representation states that N-1 out of N coordinates are 0. That is, $\alpha_i = (0, 0, \ldots, 0, \alpha_{i,N-1})$ and thus the product $\alpha_i \cdot s$ w.r.t. the CRT embedding reveals $\alpha_i \cdot s = (0 \cdot s_1, \ldots, 0 \cdot s_{N-2}, \alpha_{i,N-1} \cdot s_{N-1}) = (0, \ldots, 0, \alpha_{i,N-1} \cdot s_{N-1})$.

Constant elements in the CRT embedding. We note that the 0 element is mapped to the all 0 coefficient embedding. Furthermore an element of the form

$$a = a_0 + 0 \cdot x + \ldots 0 \cdot x^{N-1} \tag{60}$$

has the corresponding embedding as:

$$\overline{a} = \left(a_0/\mathscr{I}_1,\ \ldots,\ a_0/\mathscr{I}_i,\ \ldots\ a_0/\mathscr{I}_N\right) \tag{61}$$

For the problem at hand, mapping a bit (representing the message) means working with the all zero or all one vector of CRT coefficients. This is useful as we will need to map input bits to elements in the CRT embedding.

As a general rule, any element that is irreducible with respect to an ideal, will be mapped to itself when using the coefficients of the CRT embedding.

C.8 Our Bonus Result: AR17's Encryption is IND-ADP Admissible

We detail on the track that is presented in Section 1. To prove the indistinguishability of the iO scheme, we tackle the constraints on BPs. We know the BP has polynomial size[16] and it supports keyed bivariate functions on nimvs and inps. We also assume that the function is not constant zero (the ciphertext is clearly non-constant) and there are two different sets of nimvs under which the BPs are equivalent.

We continue and further analyse the ciphertext's structure, which leads to some interesting observations (Appendix C.13) on basing security on LWE rather than RLWE.

To recap, we have one BP for every bit of the resulting ciphertext in Equation (57). We have three main classes of ciphertexts:

- **Type 1**: data-dependent ciphertexts, which are elements depicted in Figure 8.
- **Type 2**: nimv-dependent ciphertexts, encrypting the randomness used to encrypt the tree.
- **Type 3**: the **w**-dependent ciphertext (Equation (50)), which is used during the key derivation procedure.

We proceed to analyse the BP implementation of each type of ciphertext. Section 1 provides insights for the simplest case of the (succinct) data-dependent ciphertext component. Type 2 ciphertexts are treated almost identically, while Type 3 are simpler to handle. We start with Type 3 ciphertexts, which are simpler and closer to the toy example presented in Section 1.

C.9 ADP-admissible BPs for Type 3 Ciphertexts

First, we rewrite the Type 3 ciphertext component using the CRT embedding. The original ciphertext component (using the standard RLWE form) is:

$$\mathbf{d} \leftarrow \Big(\sum_{j=0}^{N-1} w_{(j)}^{d} \cdot X^j\Big) \cdot \Big(\sum_{j=0}^{N-1} s_{(j)}^{d} \cdot X^j\Big) + p_{d-1} \cdot \Big(\sum_{j=0}^{N-1} \eta_{(j)}^{d} \cdot X^j\Big) \tag{62}$$

As regards notation, we emphasize that the superscript binds the variable to a level, and does not represent a power. For the sake of clarity in presentation, we omit the superscript d in this subsection.

Thus, in the CRT embedding, the ciphertext becomes:

$$\begin{aligned}\Big(&\overline{w_{(0)}} \cdot \overline{s_{(0)}} + p_{d-1} \cdot \overline{\eta_{(0)}}, \ldots, \\ &\overline{w_{(j)}} \cdot \overline{s_{(j)}} + p_{d-1} \cdot \overline{\eta_{(j)}}, \ldots, \\ &\overline{w_{(N-1)}} \cdot \overline{s_{(N-1)}} + p_{d-1} \cdot \overline{\eta_{(N-1)}}\Big)\end{aligned} \tag{63}$$

In Equation (63), we use the fact that constant c is mapped to a constant element having c on all coordinates.

The benefits of the CRT representation are immediately observable: the multiplications and additions are performed component-wise, modulo q. Thus, it becomes easier to reason about branching programs producing ciphertexts with respect to the CRT embedding.

[16] Our family of circuits is in NC^1.

The BP representation of Type 3 ciphertexts using the CRT Embedding. As explained in the introduction, each BP outputs a single bit. Thus, let BP retrieve any output bit of any coordinate in the CRT representation of a Type 3 ciphertext. Suppose it outputs bit ℓ of

$$\overline{w_{(j)}} \cdot \overline{s_{(j)}} + p_{d-1} \cdot \overline{\eta_{(j)}} \,. \tag{64}$$

The values of each $\overline{a_{(j)}}$ and p_{d-1} are public and we embed them in the BP. We also know that:

$$\overline{\eta_{(j)}} = \Big(\sum_{k=0}^{N-1} \eta_{(k)} \cdot \theta^{k\cdot(2j-1)} \Big) \mod \mathscr{I}_j \,. \tag{65}$$

The noise distribution plays a crucial role. Instead of working with Gaussian noise, herein we sample η from the Binomial distribution, namely having each element $\eta_j = \sum_{i=0}^{2\sigma^2-1}(b_i - b'_i)$, where σ^2 is the Binomial variance and $b_i, b'_i \in \{0,1\}^2$ are uniformly sampled.

Given the observation above on sampling the noise, we proceed with the usage of the Binomial distribution and using the above notation Equation (65) can be written explicitly as:

$$\overline{\eta_{(j)}} = \Big(\sum_{k=0}^{N-1} \big(\sum_{u=0}^{2\sigma^2-1} (b_{u,k} - b'_{u,k}) \big) \cdot \theta^{k\cdot(2j-1)} \Big) \mod \mathscr{I}_j \,. \tag{66}$$

To further simplify notation, we rewrite Equation (64) as:

$$g(\mathsf{inp}) + \sum_{k=0}^{N-1} h_k(\mathsf{inp}) \tag{67}$$

where $\overline{w_{(j)}} \cdot s = g(\mathsf{inp})$ and $p_{d-1} \cdot \overline{\eta_j} = \sum_{k=0}^{N-1} h_k(\mathsf{inp})$.

Optimizing the "size" of BPs. This part informally presents an alternative way to build BPs computing Equation (64) using as gadgets inner branching programs – inBPs. The BP outputting bit ℓ of Equation (64) can be represented as an "addition with carry" BP which: first chains the addition of lowest bitwise outputs of Equation (67), that represents the carry, then based on carry continues to add the second bit (another batch of BP) up until it outputs the desired bit ℓ. To visualize the BP, imagine a long chain: the output of the first result bit will split in twain, depending on the carry bit, and the process continues for all output bits ($\lceil \log_2(q) \rceil$). The terms to be summed are themselves coming from "inner BPs", which are detailed below and ignored for the moment. Observe that we can compose an IND-ADP-admissible "outer BP" by connecting IND-ADP-admissible inner-BPs.

Remark 9 (A Locality Property). Let $\mathfrak{C}_{d,|\mathsf{nimv}|+(n+1)}$ stand for a circuit class fulfilling conditions $1 \rightarrow 6$ in described in Theorem 1 and $\mathscr{C}$ one circuit in this class. Let inBP denote an IND-ADP-admissible circuit sampled from $\mathfrak{C}$ that computes an inner gadget element within Equation (64). Then there exists an augmented branching program BP′ able to compute each bit of a Type 3 ciphertext which such that it represents and IND-ADP-admissible circuit.

The proof strategy is depicted in Figure 10. We inject flare nodes between any two inner BP. A flare node v will always depend on the first (dummy) input bit and connect each possible output state from an inner BP to the "terminal 1" node. More specifically, we do this by modifying the matrix $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$ and injecting lines corresponding to dummy nodes.

The relation $\mathbf{A} \leftarrow \mathbf{I}_m + \overline{\mathbf{G}}^{-1}_{\mathsf{nimv}||\vec{0}} \cdot \mathbf{S}$ is preserved because whenever we consider a line j from $\overline{\mathbf{G}}^{-1}_{\mathsf{nimv}||\vec{0}}$ which is to be multiplied with the column c in $\mathbf{S}$ and the result is 1 (the "bad" event), our invariant will follow the arcs in the upward/left direction and will try to reach a contradiction of the form: line j in $\overline{\mathbf{G}}^{-1}_{\mathsf{nimv}||\vec{0}}$ multiplied with some other column $c \neq j$ from $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$ results in 1 (unless $j = c$).

In the case of bad events, we will "go up and left" up until we reach the flare node which will prevent us from going upwards (as it breaks the upper flows). In such cases, line j in $\overline{\mathbf{G}}^{-1}_{\mathsf{nimv}||\vec{0}}$ multiplied with a column indexed by $c \neq j$ will give 1, and there will be no more arcs to follow upwards, because the flare node interrupted the path. This will imply that line j multiplied with the column of $\mathbf{S}$ have to be zero, which is convenient for our setting.

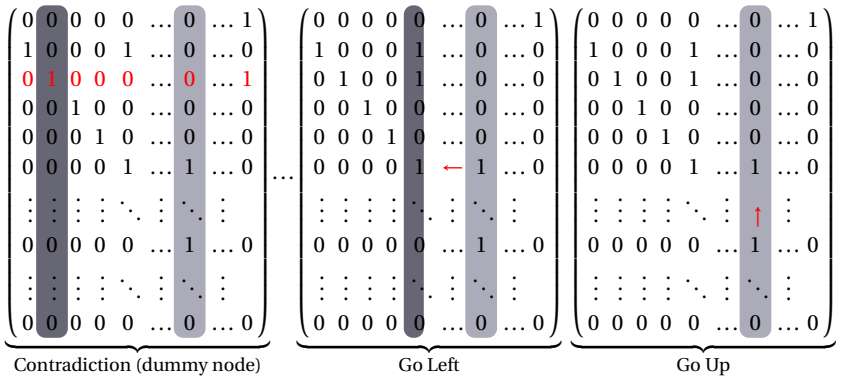


Fig. 10. Type 3's base matrix ($\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$): Introducing a dummy node and reaching contradiction. Read from right to left.

To re-enable the flow in the BP (from the start node to the terminal ones), we will introduce in the matrix corresponding to the first input bit on line j, two values set to 1 that will "switch" the input.

The previous result assumes that the inner BPs are well ordered in the sense that nodes depending on inputs occur after nodes depending on nimvs on any path within the base matrix. We show here how such BPs can be crafted.

Inner BPs. To compute the bits in functions $g(\mathsf{inp})$ and $h_k(\mathsf{inp})$ from Equation (67), we assume the existence of another set of inner BPs, say

$$\mathscr{S} := \{\mathsf{BP}_g[0], \mathsf{BP}_{h_1}[0], \ldots, \mathsf{BP}_{h_k}[0], \ldots, \mathsf{BP}_g[\ell], \mathsf{BP}_{h_1}[\ell], \ldots, \mathsf{BP}_{h_k}[\ell]\} .$$

We want to show their IND-ADP-admissibility. We show below how they can be transformed in such (non-minimal) BPs. From a high level, the idea is very similar to what have been shown in the proof above. We group nodes into batches; each batch should have nodes depending on nimvs in front of nodes depending on inps on inner paths.

Remark 10 (Inner BP*).* Let $\mathfrak{C}_{d,|\mathsf{nimv}|+(n+1)}$ denote a circuit class computing one element in the set $\mathscr{S}$ above. Then, there exists an IND-ADP-admissible BP that computes a circuit $\mathscr{C}$ from $\mathfrak{C}_{d,|\mathsf{nimv}|+(n+1)}$.

Our argument shares the injection of flare nodes technique with the argument used to justify of Remark 9. We use the following trick: given that BP′ constitutes a *sequential* representation of a circuit. Thus we order the nodes in the inner BP′ and identify clusters of nodes such that nodes depending on inps occur after the nodes depending on nimvs.

Once this condition is altered, i.e. one (or more) inp node(s) sits "above" one or more nimv node(s) we introduce dummy and flare nodes. Suppose that inp-dependent nodes $\{u_1, \ldots,\}$ occur before nimv nodes $\{t_1, \ldots\}$. We will introduce a set of "flare" nodes $\{v_1, \ldots\}$ to act as ancestors of every $\{t_1, \ldots\}$. Then, we will redirect each $\{v_1, \ldots\}$ to the terminal 1 node and thus make $\{t_1, \ldots\}$ unreachable from the start node.

As stated before, the crux point for proving IND-ADP security is the following relation:

$$\mathbf{A} \leftarrow \mathbf{I}_m + \overline{\mathbf{G}}^{-1}_{\mathsf{nimv}||\vec{0}} \cdot \mathbf{S}$$

from Section 6.

Now, assume that a line corresponding to u_1 from $\overline{\mathbf{G}}^{-1}_{\mathsf{nimv}||\vec{0}}$ is multiplied with the column t_1 in $\mathbf{S}$, and the result is 1 (the "bad" event). As shown in the proof of Theorem 1 it must be the case that the line-t_1 × column-j will give 0, for all $j \neq u_1$. The problematic case occurs if $j = u_1$.

If we consider BP′, since $j > u_1$, and since we follow the arcs in BP′, we can reach the case $j = u_1$, given that u_1 may be connected to t_1. However, in BP, this is no longer the case. The newly injected flare node ensures that the arcs in the upward/left direction and will never reach column u_1 from $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$ and thus the bad event does not occur.

As for the previous case, to re-enable the flow in the BP (from the start node to the terminal ones), we have to inject in the difference matrix corresponding to the first input bit on line ν_1, two values set to 1 that will "switch" the input.

Essentially, the previous two remarks show how BPs can be augmented into some in which condition (6) holds with locally, and thus the BP is IND-ADP admissible.

C.10 ADP-admissible BPs for Type 1 Ciphertexts

The representation of Type 1 ciphertexts using the CRT Embedding. The simple case of ciphertexts in R_{p_1} was treated in Section 1. We proceed with ciphertexts for level $i > 1$, which we know it is either $a^i \cdot s^i + p_{i-1} \cdot e^i + \mathsf{CT}^{i-1}$ or $a^i \cdot s^i + p_{i-1} \cdot e^i + s^{i-1} \cdot \mathsf{CT}^{i-1}$, where CT^{i-1} denotes the lower level ciphertext. If we expand the expression above, we obtain:

$$\mathsf{CT}^i = a^i \cdot s^i + p_{i-1} \cdot e^i + \sum_{k=1}^{i-1} \left(a^k \cdot s^k \cdot P^k + p_{k-1} \cdot e^k \cdot P^k \right) + \mathsf{m}_i \cdot P^1 \tag{68}$$

having

$$P^k = \prod_{k \le j < i} s_j^{b_j}, \tag{69}$$

where b_i can be 1 or 0 depending on the chosen path in the ciphertext "tree" described in Figure 8.

We rewrite Equation (68) using the CRT embedding and obtain the following representation for the j^{th} component within the CRT embedding:

$$\mathsf{CRT}(\mathsf{CT}^i)_{(j)} \leftarrow \overline{a^i_{(j)}} \cdot \overline{s^i_{(j)}} + p_{i-1} \cdot \overline{e^i_{(j)}} + \sum_{k=1}^{i-1} \left(\overline{a^k}_{(j)} \cdot \overline{s^k}_{(j)} \cdot \overline{P^k}_{(j)} + \overline{p_{k-1}}_{(j)} \cdot \overline{e^k}_{(j)} \cdot \overline{P^k}_{(j)} \right) + \mathsf{m}_i \cdot \overline{P^1}_{(j)}, \tag{70}$$

where we note that

$$\overline{P^k}_{\nu} = \prod_{k \le j < i} \overline{s_j^{b_j}}_{\nu}, \tag{71}$$

As for Type 3 ciphertexts, we have to take care of noise sampling. Again, we prefer sampling Binomial noise:

$$\overline{e^i_{(j)}} = \left(\sum_{r=0}^{N-1} e^i_{(r)} \cdot \theta^{r \cdot (2j-1)} \right) \bmod \mathscr{I}_j, \tag{72}$$

A similar equation can be written to sample lower level noise terms.

$$\overline{e^k_{(j)}} = \left(\sum_{r=0}^{N-1} e^k_{(r)} \cdot \theta^{r \cdot (2j-1)} \right) \bmod \mathscr{I}_j, \tag{73}$$

After replacing the noise distribution, we obtain $e_j = \sum_{r=0}^{2\sigma^2-1} (b_r - b'_r)$, where σ^2 is the Binomial variance and $(b_r, b'_i) \in \{0,1\}^2$ are uniformly sampled, and write the noise explicitly as:

$$\overline{e^i_{(j)}} = \left(\sum_{r=0}^{N-1} \left(\sum_{u=0}^{2\sigma^2-1} (b_{u,r} - b'_{u,r}) \right) \cdot \theta^{r \cdot (2j-1)} \right) \bmod \mathscr{I}_j; \tag{74}$$

and a similar formula can be derived for e^k.

Although Equation (70) comes with a higher computational complexity, its corresponding CRT embedding can still be simplified and rewritten as:

$$m(\mathsf{inp}) + \sum_{k=0}^{N-1} g_k(\mathsf{inp}) + \sum_{k=0}^{N-1} h_k(\mathsf{inp}) \tag{75}$$

where $\mathsf{m}_i \cdot \overline{P^1}_{(j)} = m(\mathsf{inp})$, $\overline{a^k_{(j)}} \cdot s^k_{(j)} \cdot \overline{P^k}_{(j)} = g_k(\mathsf{inp})$ and $p_{k-1} \cdot \overline{P^k}_{(j)} = h_k(\mathsf{inp})$.

The "shape" of BPs for Type 1 ciphertexts. As for the simpler case (Type 3), the BP outputting bit ℓ of Equation (75) can be represented as an "adder with carry". The significant difference is in the way we obtain the summands, a complexity that is *fortunately* hidden in the nested branching programs. But the high level idea stays the same: chains the addition of the branching programs outputting the rightmost bits (form the carry). As for type 3, based on carry, choose a path in the tree and continue with adding the carry to the second bits (another batch of BP) or not. This process is repeated until one obtains the desired bit of the j^{th} CRT representation of any ciphertext.

To prove the BP that outputs one bit of a Type 1 ciphertext (its CRT representation) is IND-ADP admissible, one can proceed with the higher level approach. Essentially, we restate a previous remark which assumes admissible inner BPs.

Remark 11 (Locality for Type 1 Ciphertexts). Let $\mathfrak{C}_{d,|\mathsf{nimv}|+(n+1)}$ stand for a circuit class fulfilling conditions $1 \rightarrow 6$ in described in Theorem 1 and $\mathscr{C}$ one circuit in this class. Let inBP denote an IND-ADP-admissible circuit sampled from $\mathfrak{C}$ that computes an inner gadget element within Equation (64). Then there exists a branching program BP$'$ able to compute each bit of a Type 1 ciphertext which such that it represents and IND-ADP-admissible circuit.

The argument is literally identical to the one used for Type 3 ciphertexts, except for a difference stemming from using different BPs. This remark assumes that the inner BPs are well-ordered in the sense that nodes depending on inputs occur after nodes depending on nimvs on any path. The setting here is identical.

Inner BPs. The shape of inner BPs is radically changed from Type 3. We first discuss the newly added complexity and see how this translates into the form of BPs. The most convoluted terms are the P terms. They represent products of CRT coefficients. Still, computing products can be tackled through branching program in a linear way.

Suppose one needs to multiply two integers represented in binary as: $a = (a_i, \ldots, a_0)$ and $b = (b_i, \ldots, b_0)$. The multiplication can be performed in the classical way, by first getting $(a_i \cdot b_0, \ldots, a_0 \cdot b_0)$ $(a_i \cdot b_1, \ldots, a_0 \cdot b_1)$ and $(a_i \cdot b_i, \ldots, a_0 \cdot b_i)$ and the summing them up through addition with carry.

This algorithm can be extended to recursively handle multiplication of n elements. However, each component can be, in fact, an inner BP. Thus, it can be shown trivially that such nested BPs can be "chained" to recover the values of each g_k, h_k or m.

In what follows, we restate the result described for Type 3 ciphertexts, which show how to introduce flare nodes and ensure that on every path containing nimv-dependent nodes, all inp-nodes occur after the nimv nodes.

Remark 12. Let $\mathfrak{C}_{d,|\mathsf{nimv}|+(n+1)}$ denote a circuit class computing one bit of a Type 1 ciphertext. Then, there exists an IND-ADP-admissible BP that computes a circuit $\mathscr{C}$ from $\mathfrak{C}_{d,|\mathsf{nimv}|+(n+1)}$.

We have seen above how the function m, g_k, h_k can be rewritten by using inner binary decision diagrams that implement the base functionality, namely producing uniform outputs (done through PRFs). Once the branching program has been transformed to use such building blocks, we can apply the same technique as in Remark 12 that involves adding flare nodes and interrupting paths that visit nimv-dependent nodes after inp-dependent nodes.

As for the case of Type 3 ciphertexts, we have to re-enable the correct paths in the BP (from the start node to the terminal ones), we have to inject in the matrix corresponding to the first input bit on line v_1, two values set to 1 that will "switch" the input.

Essentially, the previous two proofs show how BPs can be transformed into some in which condition (6) holds with locality, and thus the BP is ADP admissible.

C.11 ADP-admissible BPs for Type 2 Ciphertexts

The BP representation of Type 2 ciphertexts through CRT Embedding. Handling Type 2 ciphertexts can be done similarly to Type 1 ciphertexts. From a high level point of view, the most significant difference consists in replacing the message/input data m_i with the value of the secret. In our analysis, we will handle exclusively the case for the encryption of s, and not $s \cdot s$, at any level (see also the discussion in Appendix C.13 on getting rid of circular security at the expense of having the supported functions in NC^0).

A Type 2 ciphertext can be written as:

$$\mathsf{CT}^i = a^i \cdot s^i + p_{i-1} \cdot e^i + \sum_{k=1}^{i-1} \left(a^k \cdot s^k \cdot P^k + p_{k-1} \cdot e^k \cdot P^k \right) + s^1 \cdot P^1 \tag{76}$$

where P^k is defined as for the previous case (Type 1) as:

$$P^k = \prod_{k \le j < i} s_j^{b_j}, \tag{77}$$

With respect to the CRT embedding, Equation (76)'s j^{th} component becomes:

$$\mathsf{CRT}(\mathsf{CT}^i)_{(j)} \leftarrow \overline{a^i_{(j)}} \cdot \overline{s^i_{(j)}} + p_{i-1} \cdot \overline{e^i_{(j)}} + \sum_{k=1}^{i-1} \left(\overline{a^k}_{(j)} \cdot \overline{s^k}_{(j)} \cdot \overline{P^k}_{(j)} + \overline{p_{k-1}}_{(j)} \cdot \overline{e^k}_{(j)} \cdot \overline{P^k}_{(j)} \right) + s^1_i \cdot \overline{P^1}_{(j)}, \tag{78}$$

where we note that

$$\overline{P^k}_v = \prod_{k \le j < i} \overline{s_j^{b_j}}_v. \tag{79}$$

As for Type 1 ciphertexts, the noise sampled from the Binomial distribution can be written directly as:

$$\overline{e^i_{(j)}} = \left(\sum_{r=0}^{N-1} \left(\sum_{u=0}^{2\sigma^2-1} (b_{u,r} - b'_{u,r}) \right) \cdot \theta^{r \cdot (2j-1)} \right) \bmod \mathscr{I}_j. \tag{80}$$

and a similar formula can be derived for each e^k.

As for the previous case, the CRT embedding can still be simplified and rewritten as:

$$s(\mathsf{inp}) + \sum_{k=0}^{N-1} g_k(\mathsf{inp}) + \sum_{k=0}^{N-1} h_k(\mathsf{inp}) \tag{81}$$

where $s^1 \cdot \overline{P^1}_{(j)} = s(\mathsf{inp})$, $\overline{a^k_{(j)}} \cdot s^k_{(j)} \cdot \overline{P^k}_{(j)} = g_k(\mathsf{inp})$ and $p_{k-1} \cdot \overline{P^k}_{(j)} = h_k(\mathsf{inp})$.

The "shape" of BPs for Type 2 ciphertexts. The BP outputting bit ℓ^{th} of Equation (81) can be represented as an "adder with carry". The only difference to Type 1 ciphertexts consists in the representation of function $s(\cdot)$, which requires an extra multiplication compared to the $m(\cdot)$ function used in Equation (75). Still, the extra multiplication can be done in polynomial time in size of the modulus q, using the same approach.

To prove that a BP outputting one bit of a Type 2 ciphertext in the CRT representation is IND-ADP admissible, we rely on the same, higher level approach, where we break the problem into multiple pieces which are assumed to fulfil Condition 6. Then we state the following remark on their composition.

Remark 13. Let $\mathfrak{C}_{d,|\mathsf{nimv}|+(n+1)}$ stand for a circuit class fulfilling conditions $1 \to 6$ in described in Theorem 1 and $\mathscr{C}$ one circuit in this class. Let inBP denote an IND-ADP-admissible circuit sampled from $\mathfrak{C}$ that computes an inner gadget element. Then there exists an augmented branching program BP′ able to compute each bit of a Type 2 ciphertext which such that it represents and IND-ADP-admissible circuit.

The argument is identical to the one used for Type 3 ciphertexts, except for a different using different BPs.

Inner BPs for Type 2 Ciphertexts. The shape of inner BPs is similar to ones in the previous case. As stated previously, the only part that changes is an extra multiplication with s^1, or for the upper levels, two multiplications with s^{i-1} multiplications per level i. Such multiplications can be handled in a similar way.

Remark 14. Let $\mathfrak{C}_{d,|\mathsf{nimv}|+(n+1)}$ denote a circuit class computing one element in Equation (81) above. Then, there exists an IND-ADP-admissible BP that computes a circuit $\mathscr{C}$ from $\mathfrak{C}_{d,|\mathsf{nimv}|+(n+1)}$.

The argument is virtually identical to Type 1 and 2 ciphertexts.

The puncturable PRF functionality. In all parts of our proofs, concerning Type 1, 2 and 3 ciphertexts we will constantly invoke puncturable PRFs for multiple times. To remove the constraint of using different punctured keys for each pPRF once it gets invoked and to handle the production of different outputs given the same inputs, we can prefix the pPRF inputs for each particular function $g_k(\cdot), h_k(\cdot), s(\cdot), m(\cdot)$.

C.12 Considerations on the 5th Condition

Herein we consider the 5^{th} condition as well. It states that the first row of our matrix has the special form $(0,0,\dots 0,1)$, which means it always sets the base matrix's determinant to 1. Equivalently stated, the function always evaluates to 1 when evaluated on input $(0||*\dots*)$.

This is advantageous as we can easily handle the inverse matrix of the base case (which always exists). Still, there is one pedantic case that needs to be handled: the first column (corresponding to the first input bit) multiplied with some column equates 1. Then, we need to follow the path in the BP upwards. As stated, fortunately, the flare nodes interrupt it.

C.13 Remarks on AR17

There are several questions that are related to our work. The first one concerns efficiency. The size of a binary decision diagram is upper bounded, in the worst case by 2^{2d}, where d is the depth of the circuit. In our case, such circuits are growing linearly with the number of multiplication layers, which is restrictive in practice. The natural question: can be build lower BPs, or replace them with some other primitive? The second question regards the succinct functional encryption to be used. Instead of working with AR17, can we work with a "friendlier" encryption procedure from another scheme, one that is easier to reason about?

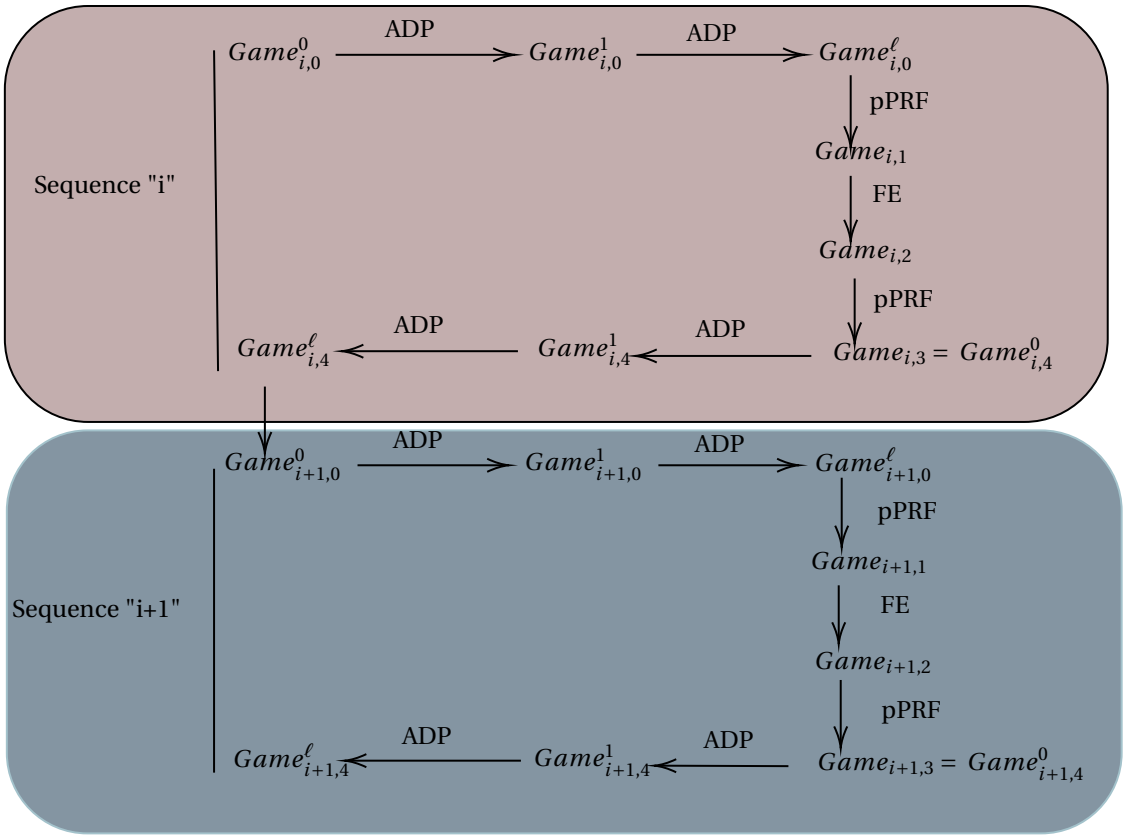
C.14 Instantiation from Standard LWE

Because we work with AR17, we can replace RLWE with standard LWE. Given that the FE ciphertext needs to support *one* function in NC^1, say f, we can obtain the randomized encoding of f, and then encrypt the randomness terms and the message used to evaluate the randomized encoding. This means that an AR17 ciphertext needs to support v^2 functions, where v is the size of the BP for f. This further implies an increase in ciphertext's size. We will end up with an AR17 having only 3 levels, but with a large data independent component.

If we analyse the size of v, we can see that $v \leq 2^{2d}$, and the BP will admit a representation using at most 2^{4d} nodes. The Assuming that we need to support 2^{2d} ciphertexts and given that AR17 ciphertext grows additively with the square of the number of functions[2, p. 1], the ciphertext AR17 will grow additively with 2^{8d}. As long as $d \in O(\log_2(n))$ (i.e. the supported class of functions is in NC^1), the ciphertext will be polynomial and will depend on the depth of the circuit, not on its size, so it still fulfils succinctness requirements. We leave this for future work.

D The Full iO Proof

We provide explicit code-based games for the hybrid transitions.



D.1 $\mathsf{Game}^{j-1}_{i,0} \rightarrow \mathsf{Game}^{j}_{i,0}$:

Case $i = 1$. The first type of hybrid is presented in what follows and corresponds to the very first set of hybrids, corresponding to $i = 1$. Pictorially, the IND-ADP game is presented on the right, which makes use of a PPT algorithm, the reduction $\mathscr{B}$ to impersonate an adversary. Itself, the reduction makes use of a PPT distinguisher $\mathscr{A}$, able to distinguish between the distribution of two hybrids.

$\mathsf{IND\text{-}ADP}_{\mathscr{B}}(1^\lambda, \ell, n)$:	$\mathscr{B}_{\mathscr{A}}(1^\lambda, 1^{g'}, 1^{d'}, 1, j)$:	$\mathscr{A}(1^\lambda, \mathscr{C}_0, \mathscr{C}_1)$:
100. $b_{\mathsf{IND\text{-}ADP}} \leftarrow\$ \{0,1\}$		
		101. $\mathsf{m}_0 \leftarrow \mathsf{Base2}(\mathscr{C}_0)$
		102. $\mathsf{m}_1 \leftarrow \mathsf{Base2}(\mathscr{C}_1)$
		103. send $(\mathsf{m}_0, \mathsf{m}_1)$
	104. receive $(\mathsf{m}_0, \mathsf{m}_1)$	
	105. $(\mathsf{msk}, \mathsf{mpk}) \leftarrow\$ \mathsf{FE.Setup}(1^{g'}, 1^{d'})$	
	106. $\rho \leftarrow 2^n - i$.	
	107. $\mathsf{k}_{\mathsf{pPRF}} \leftarrow\$ \mathsf{pPRF.Setup}(1^\lambda)$	
	108. $c \leftarrow\$ \mathsf{FE.Enc}(\mathsf{mpk}, \mathsf{m}_0 \| \rho; \mathsf{pPRF}(\mathsf{k}, \rho))$.	
	109. $w \leftarrow\$ \{0,1\}^n$	
	110. $\mathsf{k}_0^* \leftarrow \mathsf{pPRF.Puncture}(\mathsf{k}, 0\|\|w)$.	
	111. $\mathsf{k}_1^* \leftarrow \mathsf{pPRF.Puncture}(\mathsf{k}, 1\|\|\rho)$.	
	112. $\mathsf{sk}_{\mathsf{Uc}} \leftarrow\$ \mathsf{FE.KGen}(\mathsf{msk}, \mathsf{Uc})$	
	113. $\mathscr{C}_0 := \mathscr{C}^j_{\mathsf{mpk}, \mathsf{k}_0^*, \mathsf{m}_0, \perp, \rho, c}(b\|\|\mathsf{inp})$	
	114. if $b = 0$: return 0	
	115. if $\mathsf{inp} < \rho$: return $\mathsf{FE}_j.\mathsf{Enc}\left(\mathsf{mpk}, \mathsf{m}_0\|\|\mathsf{inp}; \mathsf{pPRF}(\mathsf{k}_0^*, 1\|\|\mathsf{inp})\right)$	
	116. if $\mathsf{inp} = \rho$: return c	
	117. if $\mathsf{inp} > \rho$: return $\mathsf{FE}_j.\mathsf{Enc}\left(\mathsf{mpk}, \perp\|\|\mathsf{inp}; \mathsf{pPRF}(\mathsf{k}_0^*, 1\|\|\mathsf{inp})\right)$	
	118. $\mathscr{C}_1 := \mathscr{C}^j_{\mathsf{mpk}, \mathsf{k}_1^*, \mathsf{m}_0, \perp, \rho, c}(b\|\|\mathsf{inp})$	
	119. if $b = 0$: return 0	
	120. if $\mathsf{inp} < \rho$: return $\mathsf{FE}_j.\mathsf{Enc}\left(\mathsf{mpk}, \mathsf{m}_0\|\|\mathsf{inp}; \mathsf{pPRF}(\boxed{\mathsf{k}_1^*}, 1\|\|\mathsf{inp})\right)$	
	121. if $\mathsf{inp} = \rho$: return c	
	122. if $\mathsf{inp} > \rho$: return $\mathsf{FE}_j.\mathsf{Enc}\left(\mathsf{mpk}, \perp\|\|\mathsf{inp}; \mathsf{pPRF}(\boxed{\mathsf{k}_1^*}, 1\|\|\mathsf{inp})\right)$	
	123. send $(\mathscr{C}_0, \mathscr{C}_1)$	
124. receive $(\mathscr{C}_0, \mathscr{C}_1)$		
125. $\overline{\mathscr{C}} \leftarrow\$ \mathsf{ADP.Setup}(1^\lambda, \mathscr{C}_b)$		
126. send $\overline{\mathscr{C}}$		
	127. receive $\overline{\mathscr{C}}$	
	128. sample all $\left\{\mathsf{ADP}^{j'}\right\}_{j' \in [\ell], j' \neq j}$ as in prev game	
	129. send $\left(\mathsf{sk}_{\mathsf{Uc}}, \left\{\mathsf{ADP}^j\right\}_{j \in [\ell]}\right)$ as $\mathsf{iO}_{\mathsf{m}_b}$.	
		130. receive $\overline{\mathscr{C}}$
		131. return $b_{\mathscr{A}}$
	132. receive $b_{\mathscr{A}}$	
	133. send $b_{\mathscr{B}} := b_{\mathscr{A}}$	
134. receive $b_{\mathscr{B}}$		
135. return $b_{\mathsf{IND\text{-}ADP}} \stackrel{?}{=} b_B$		

Fig. 11. The transition $\mathsf{Game}^{j-1}_{1,0} \rightarrow \mathsf{Game}^{j}_{1,0}$. Interactions are done in the consecutive order of the numbered lines. Reduction is placed in the middle column, having its internal working described. Interactions between the parties are done through send, receive.

The adversary $\mathscr{A}$ will always query with two functionally-equivalent circuit, and will use their binary representation: $(\mathsf{m}_0, \mathsf{m}_1)$. The reduction $\mathscr{B}$ described in Figure 11 must ensure the circuits $\mathscr{C}_0$ and $\mathscr{C}_1$ on lines 113 and 118 are functionally equivalent. This can be seen as the evaluation under k_0^* and k_1^* is identical given that on lines 116 and 121, both circuits return a precomputed, hardcoded value.

Case $i > 1$. This subcase corresponds to $i > 1$, such that a message m_1 is encrypted whenever $i > \rho$.

$\underline{\mathsf{IND\text{-}ADP}_{\mathscr{B}}(1^\lambda, \ell, n)}$:
100. $b_{\mathsf{IND\text{-}ADP}} \leftarrow\$ \{0,1\}$

$\underline{\mathscr{B}_{\mathscr{A}}(1^\lambda, 1^{g'}, 1^{d'}, i, j)}$:

$\underline{\mathscr{A}(1^\lambda, \mathscr{C}_0, \mathscr{C}_1)}$:

101. $\mathsf{m}_0 \leftarrow \mathrm{Base2}(\mathscr{C}_0)$
102. $\mathsf{m}_1 \leftarrow \mathrm{Base2}(\mathscr{C}_1)$
103. send $(\mathsf{m}_0, \mathsf{m}_1)$
104. receive $(\mathsf{m}_0, \mathsf{m}_1)$
105. $(\mathsf{msk}, \mathsf{mpk}) \leftarrow\$ \mathsf{FE.Setup}(1^{g'}, 1^{d'})$
106. $\rho \leftarrow 2^n - i$.
107. $\mathsf{k}_{\mathsf{pPRF}} \leftarrow\$ \mathsf{pPRF.Setup}(1^\lambda)$
108. $c \leftarrow\$ \mathsf{FE.Enc}(\mathsf{mpk}, \mathsf{m}_0 || \rho; \mathsf{pPRF}(\mathsf{k}, \rho))$.
109. $w \leftarrow\$ \{0,1\}^n$
110. $\mathsf{k}_0^* \leftarrow \mathsf{pPRF.Puncture}(\mathsf{k}, 0||w)$.
111. $\mathsf{k}_1^* \leftarrow \mathsf{pPRF.Puncture}(\mathsf{k}, 1||\rho)$.
112. $\mathsf{sk}_{\mathsf{Uc}} \leftarrow\$ \mathsf{FE.KGen}(\mathsf{msk}, \mathsf{Uc})$
113. $\mathscr{C}_0 := \underline{\mathscr{C}^j_{\mathsf{mpk}, \mathsf{k}_0^*, \mathsf{m}_0, \mathsf{m}_1, \rho, c}(b||\mathsf{inp})}$
114. if $b = 0$: return 0
115. if $\mathsf{inp} < \rho$: return $\mathsf{FE}_j.\mathsf{Enc}\left(\mathsf{mpk}, \mathsf{m}_0||\mathsf{inp}; \mathsf{pPRF}(\mathsf{k}_0^*, 1||\mathsf{inp})\right)$
116. if $\mathsf{inp} = \rho$: return c
117. if $\mathsf{inp} > \rho$: return $\mathsf{FE}_j.\mathsf{Enc}\left(\mathsf{mpk}, \mathsf{m}_1||\mathsf{inp}; \mathsf{pPRF}(\mathsf{k}_0^*, 1||\mathsf{inp})\right)$
118. $\mathscr{C}_1 := \underline{\mathscr{C}^j_{\mathsf{mpk}, \mathsf{k}_1^*, \mathsf{m}_0, \mathsf{m}_1, \rho, c}(b||\mathsf{inp})}$
119. if $b = 0$: return 0
120. if $\mathsf{inp} < \rho$: return $\mathsf{FE}_j.\mathsf{Enc}\left(\mathsf{mpk}, \mathsf{m}_0||\mathsf{inp}; \mathsf{pPRF}(\boxed{\mathsf{k}_1^*}, 1||\mathsf{inp})\right)$
121. if $\mathsf{inp} = \rho$: return c
122. if $\mathsf{inp} > \rho$: return $\mathsf{FE}_j.\mathsf{Enc}\left(\mathsf{mpk}, \mathsf{m}_1||\mathsf{inp}; \mathsf{pPRF}(\boxed{\mathsf{k}_1^*}, 1||\mathsf{inp})\right)$
123. send $(\mathscr{C}_0, \mathscr{C}_1)$
124. receive $(\mathscr{C}_0, \mathscr{C}_1)$
125. $\overline{\mathscr{C}} \leftarrow\$ \mathsf{ADP.Setup}(1^\lambda, \mathscr{C}_b)$
126. send $\overline{\mathscr{C}}$
127. receive $\overline{\mathscr{C}}$
128. sample all $\left\{\mathsf{ADP}^{j'}\right\}_{j' \in [\ell], j' \neq j}$ as in prev game
129. send $\left(\mathsf{sk}_{\mathsf{Uc}}, \left\{\mathsf{ADP}^j\right\}_{j \in [\ell]}\right)$ as $\mathsf{iO}_{\mathsf{m}_b}$.
130. receive $\overline{\mathscr{C}}$
131. return $b_{\mathscr{A}}$
132. receive $b_{\mathscr{A}}$
133. send $b_{\mathscr{B}} := b_{\mathscr{A}}$
134. receive $b_{\mathscr{B}}$
135. return $b_{\mathsf{IND\text{-}ADP}} \stackrel{?}{=} b_B$

Fig. 12. The transition $\mathsf{Game}^{j-1}_{i,0} \rightarrow \mathsf{Game}^{j}_{i,0}$ (subcase $i > 1$). Interactions are done in consecutive order.

The game for this subcase is similar to the subcase where $i = 1$. The only change occurs in the description of circuits, where we can see that on lines 117 and 122 in Figure 12, that the messages m_1 are encrypted. It is easy to see the two circuits are functionally equivalent.

D.2 $\mathsf{Game}^{\ell}_{i,0} \to \mathsf{Game}_{i,1}$

Case $i = 1$. For the case that $i = 1$, corresponding to the very first batch of hybrids, the $\perp$ symbol is encrypted corresponding to $\mathsf{inp} > \rho$.

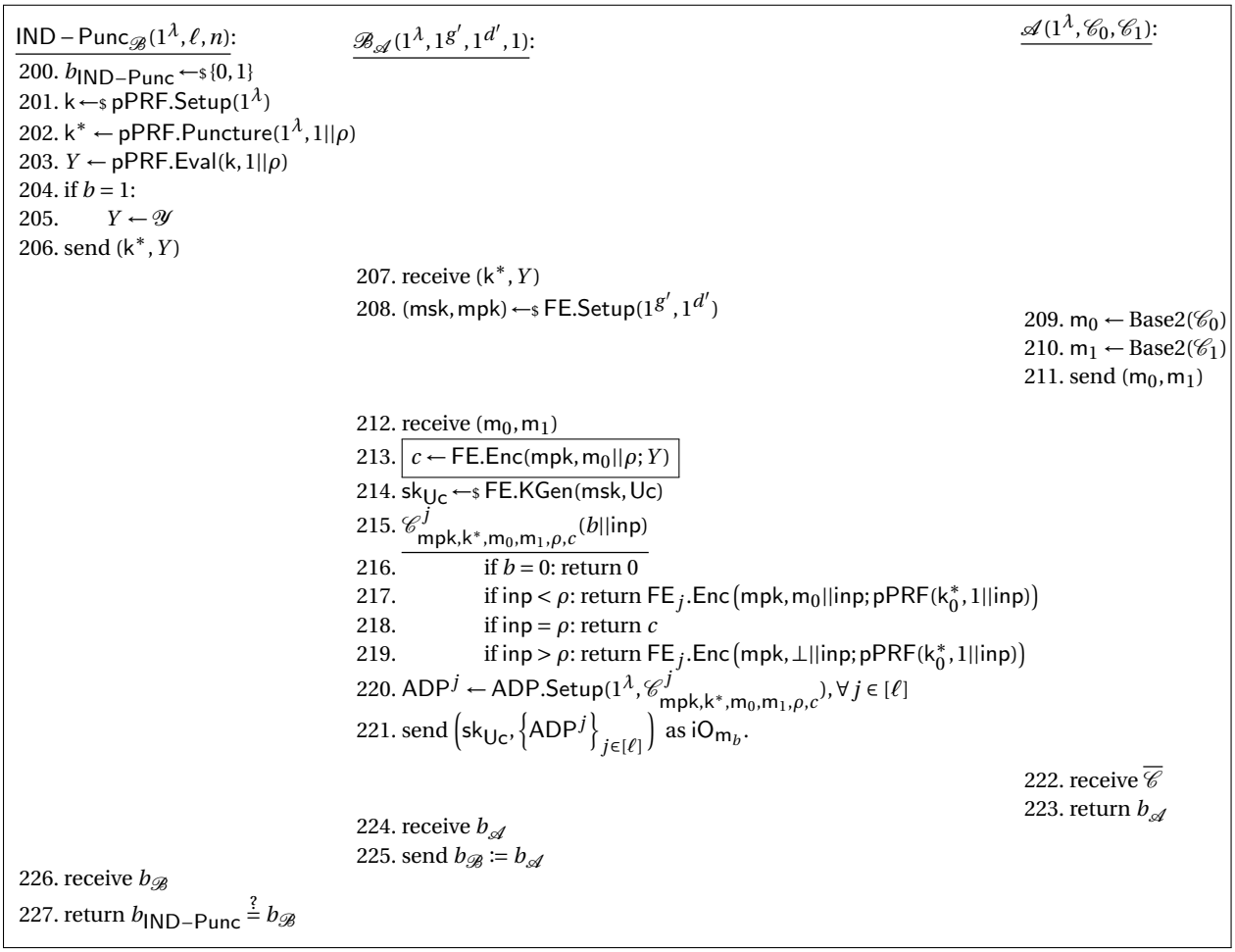


Fig. 13. The transition $\mathsf{Game}^{\ell}_{1,0} \to \mathsf{Game}_{1,1}$. Interactions are done in consecutive order.

This is the particular subcase corresponding to $i = 1$ in the sequence of hybrids, as opposed to the general case that follows. The difference is that in this game, line 219 encrypts $\perp$ as opposed to m_1. Assuming the adversary distinguishes the two settings with advantage $\epsilon_{\mathscr{A}}$, $\mathscr{B}$ wins the IND − Punc game with the same advantage.

Case $i > 1$. This is the general subcase similar to the previous subcase, up to the noticeable difference of using m_1, instead of $\perp$.

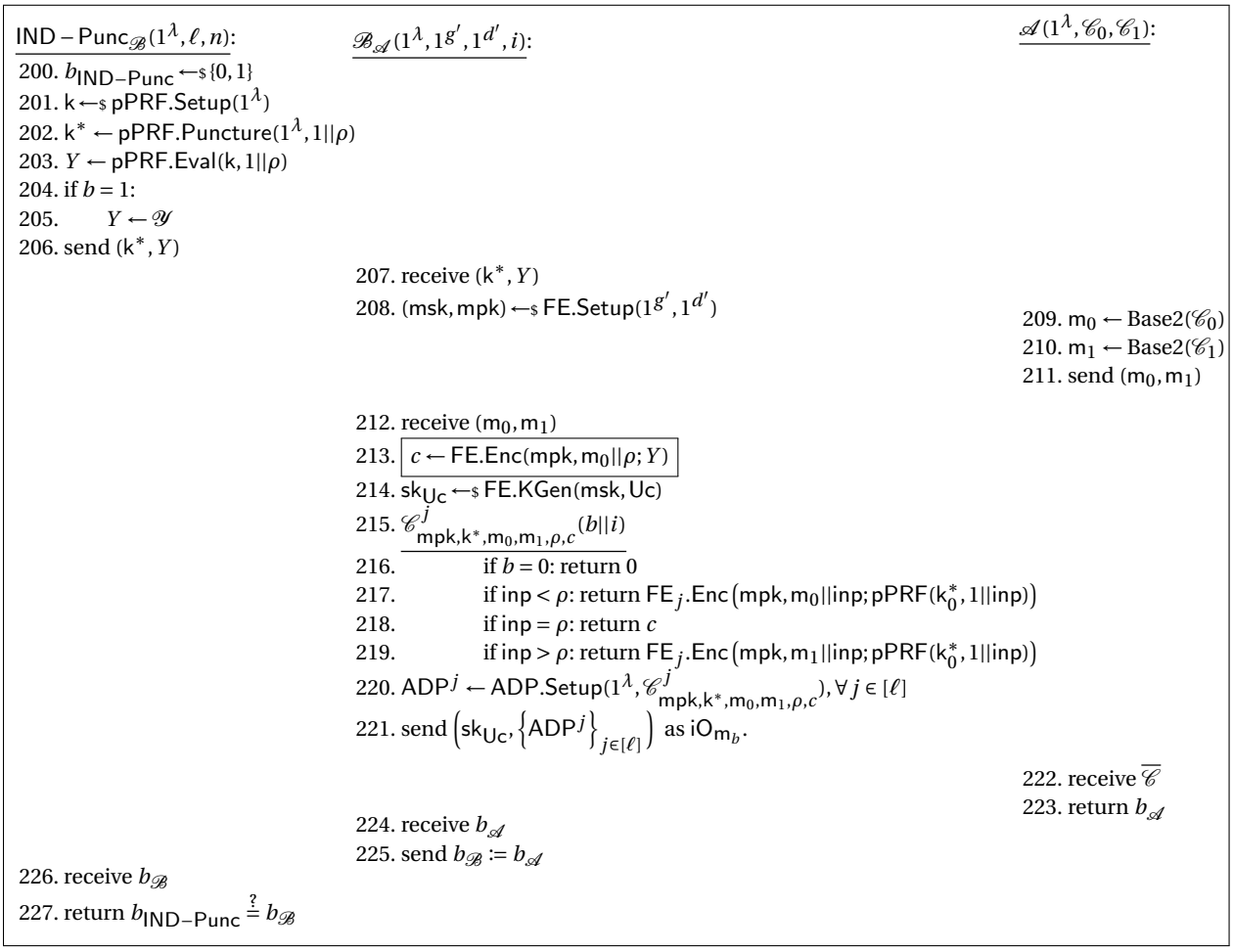


Fig. 14. The transition $\mathsf{Game}^{\ell}_{i,0} \rightarrow \mathsf{Game}_{i,1}$. Interactions are done in consecutive order.

The argument is the same as for the case $i = 1$.

D.3 $\mathsf{Game}_{i,1} \to \mathsf{Game}_{i,2}$

Case $i = 1$. For the very first set of hybrids corresponding to $i = 1$, $\perp$ gets encrypted for $i = 1$.

$\underline{\mathsf{s\text{-}IND\text{-}FE\text{-}CPA}_{\mathscr{B}}(1^\lambda)}$:	$\underline{\mathscr{B}_{\mathscr{A}}(1^\lambda, 1^{g'}, 1^{d'}, i)}$:	$\underline{\mathscr{A}(1^\lambda, \mathscr{C}_0, \mathscr{C}_1)}$:
300. $b_{\mathsf{FE}} \leftarrow\$ \{0,1\}$		
		301. $\mathsf{m}_0 \leftarrow \mathsf{Base2}(\mathscr{C}_0)$
		302. $\mathsf{m}_1 \leftarrow \mathsf{Base2}(\mathscr{C}_1)$
		303. send $(\mathsf{m}_0, \mathsf{m}_1)$
	304. receive $(\mathsf{m}_0, \mathsf{m}_1)$	
	305. send $(\mathsf{Uc}, \mathsf{m}_0 \Vert \rho, \mathsf{m}_1 \Vert \rho)$	
306. receive $(\mathsf{Uc}, \mathsf{m}_0 \Vert \rho, \mathsf{m}_1 \Vert \rho)$		
307. $(\mathsf{msk}, \mathsf{mpk}) \leftarrow\$ \mathsf{FE.Setup}(1^\lambda, 1^{g'}, 1^{d'})$		
308. $c \leftarrow\$ \mathsf{FE.Enc}(\mathsf{mpk}, \mathsf{m}_b \Vert \rho)$		
309. $\mathsf{sk}_f \leftarrow\$ \mathsf{FE.KGen}(\mathsf{msk}, \mathsf{Uc})$		
310. send $(\mathsf{mpk}, \mathsf{sk}_f, c)$		
	311. receive $(\mathsf{mpk}, \mathsf{sk}_f, c)$	
	312. $\mathsf{k} \leftarrow\$ \mathsf{pPRF.Setup}(1^\lambda)$	
	313. $\mathsf{k}^* \leftarrow \mathsf{pPRF.Puncture}(\mathsf{k}, 1 \Vert \rho)$	
	314. $\underline{\mathscr{C}^j_{\mathsf{mpk},\mathsf{k}^*,\mathsf{m}_0,\mathsf{m}_1,\rho,c}(b \Vert \mathsf{inp})}$	
	315. if $b = 0$: return 0	
	316. if $\mathsf{inp} < \rho$: return $\mathsf{FE}_j.\mathsf{Enc}\left(\mathsf{mpk}, \mathsf{m}_0 \Vert \mathsf{inp}; \mathsf{pPRF}(\mathsf{k}^*, 1 \Vert \mathsf{inp})\right)$	
	317. if $\mathsf{inp} = \rho$: return $\boxed{c}$	
	318. if $\mathsf{inp} > \rho$: return $\mathsf{FE}_j.\mathsf{Enc}\left(\mathsf{mpk}, \mathsf{m}_1 \Vert \mathsf{inp}; \mathsf{pPRF}(\mathsf{k}^*, 1 \Vert \mathsf{inp})\right)$	
	319. $\mathsf{ADP}^j \leftarrow \mathsf{ADP.Setup}(1^\lambda, \mathscr{C}^j_{\mathsf{mpk},\mathsf{k}^*,\mathsf{m}_0,\mathsf{m}_1,\rho,c}), \forall j \in [\ell]$	
	320. send $\left(\mathsf{sk}_{\mathsf{Uc}}, \left\{\mathsf{ADP}^j\right\}_{j\in[\ell]}\right)$ as $\mathsf{iO}_{\mathsf{m}_b}$.	
		321. receive $\overline{\mathscr{C}}$
		322. return $b_{\mathscr{A}}$
	323. receive $b_{\mathscr{A}}$	
	324. send $b_{\mathscr{B}} := b_{\mathscr{A}}$	
325. receive $b_{\mathscr{B}}$		
326. return $b_{\mathsf{IND\text{-}FE\text{-}CPA}} \stackrel{?}{=} b_{\mathscr{B}}$		

Fig. 15. The reduction is in the middle. We used a simplified version of the FE selective experiment, where the adversary (impersonated by the reduction $\mathscr{B}$) commits to one function and one pair of messages a priori.

The reduction to FE security is natural. $\mathscr{B}$ asks for a functional key corresponding to the universal circuit, as well as for one ciphertext corresponding to either m_0 or m_1. The two messages are received on line 304 (Figure 15) from the adversary $\mathscr{A}$. Then the challenge ciphertext is embedded as the challenge (differing) ciphertext in line 317, and $\mathscr{A}$ is asked to distinguish. Clearly $\mathscr{B}$ wins the FE game with the same advantage as $\mathscr{A}$ distinguishes the settings.

Case $i > 1$. For the remaining sets of hybrids corresponding to $i > 1$, m gets encrypted for $i > 1$.

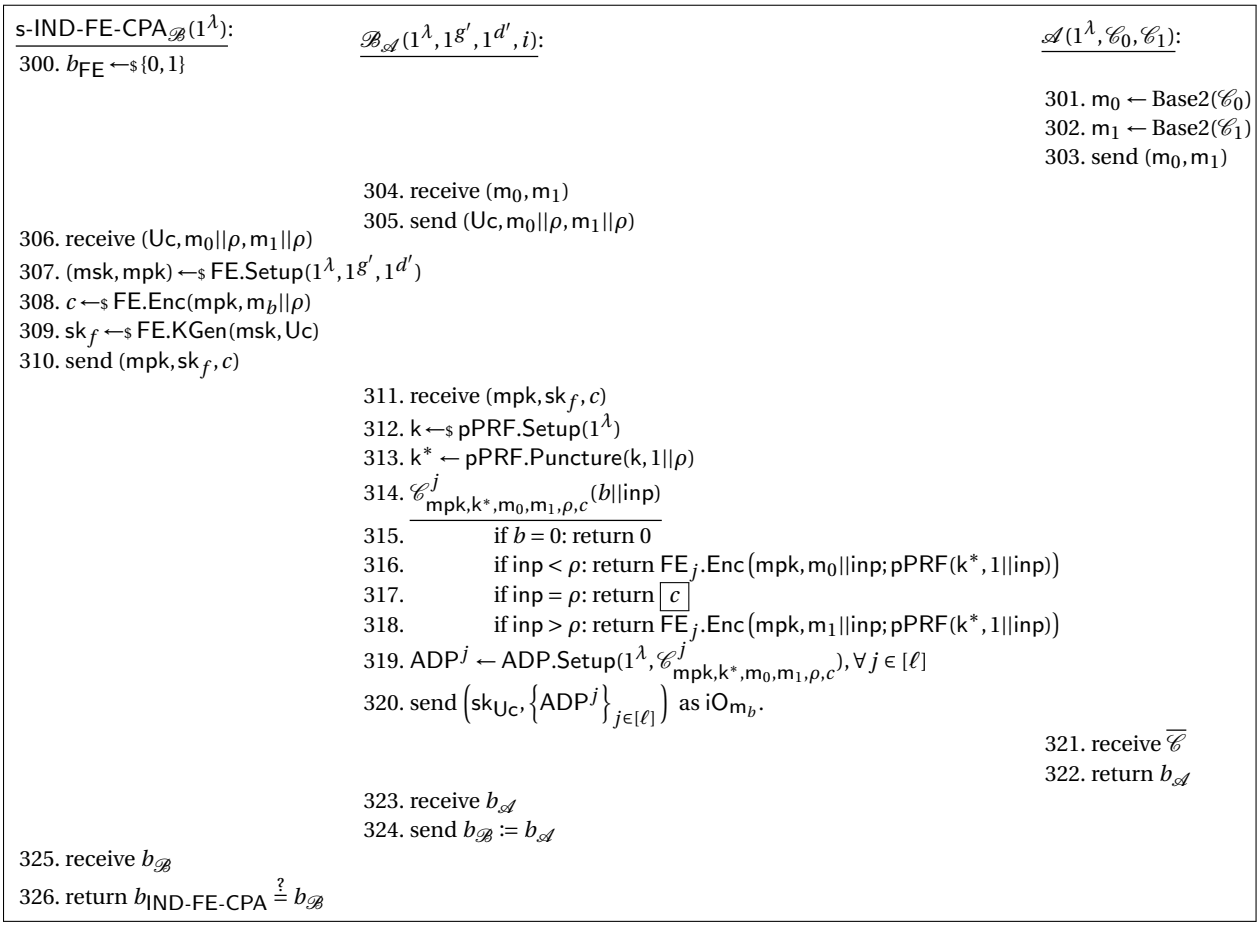


Fig. 16. The reduction to the FE selective security for the general case $i > 1$.

The reduction is similar. for the general case, there is the message m_1 that is encrypted on line 318 (Figure 16), as opposed to the special symbol $\perp$.

D.4 $\mathsf{Game}_{i,2} \to \mathsf{Game}_{i,3}$

Case $i = 1$. For the case that $i = 1$, corresponding to the very first batch of hybrids, the $\perp$ symbol is encrypted corresponding to $\mathsf{inp} > \rho$.

$\underline{\mathsf{IND-Punc}_{\mathscr{B}}(1^\lambda, \ell, n)}$:
400. $b_{\mathsf{IND-Punc}} \leftarrow\!\!\$ \{0,1\}$
401. $\mathsf{k} \leftarrow\!\!\$ \mathsf{pPRF.Setup}(1^\lambda)$
402. $\mathsf{k}^* \leftarrow \mathsf{pPRF.Puncture}(1^\lambda, 1||\rho)$
403. $Y \leftarrow \mathscr{Y}$
404. if $b = 1$:
405. $\quad Y \leftarrow \mathsf{pPRF.Eval}(\mathsf{k}, 1||\rho)$
406. send (k^*, Y)

$\underline{\mathscr{B}_{\mathscr{A}}(1^\lambda, 1^{g'}, 1^{d'}, 1)}$:
407. receive (k^*, Y)
408. $(\mathsf{msk}, \mathsf{mpk}) \leftarrow\!\!\$ \mathsf{FE.Setup}(1^{g'}, 1^{d'})$
412. receive $(\mathsf{m}_0, \mathsf{m}_1)$
413. $\boxed{c \leftarrow \mathsf{FE.Enc}(\mathsf{mpk}, \mathsf{m}_0||\rho; Y)}$
414. $\mathsf{sk}_{\mathsf{Uc}} \leftarrow\!\!\$ \mathsf{FE.KGen}(\mathsf{msk}, \mathsf{Uc})$
415. $\underline{\mathscr{C}^j_{\mathsf{mpk},\mathsf{k}^*,\mathsf{m}_0,\mathsf{m}_1,\rho,c}(b||\mathsf{inp})}$
416. $\quad$ if $b = 0$: return 0
417. $\quad$ if $\mathsf{inp} < \rho$: return $\mathsf{FE}_j.\mathsf{Enc}\left(\mathsf{mpk}, \mathsf{m}_0||\mathsf{inp}; \mathsf{pPRF}(\mathsf{k}_0^*, 1||\mathsf{inp})\right)$
418. $\quad$ if $\mathsf{inp} = \rho$: return c
419. $\quad$ if $\mathsf{inp} > \rho$: return $\mathsf{FE}_j.\mathsf{Enc}\left(\mathsf{mpk}, \perp||\mathsf{inp}; \mathsf{pPRF}(\mathsf{k}_0^*, 1||\mathsf{inp})\right)$
420. $\mathsf{ADP}^j \leftarrow \mathsf{ADP.Setup}(1^\lambda, \mathscr{C}^j_{\mathsf{mpk},\mathsf{k}^*,\mathsf{m}_0,\mathsf{m}_1,\rho,c}), \forall j \in [\ell]$
421. send $\left(\mathsf{sk}_{\mathsf{Uc}}, \left\{\mathsf{ADP}^j\right\}_{j\in[\ell]}\right)$ as $\mathsf{iO}_{\mathsf{m}_b}$.
424. receive $b_{\mathscr{A}}$
425. send $b_{\mathscr{B}} := b_{\mathscr{A}}$
426. receive $b_{\mathscr{B}}$
427. return $b_{\mathsf{IND-Punc}} \stackrel{?}{=} b_{\mathscr{B}}$

$\underline{\mathscr{A}(1^\lambda, \mathscr{C}_0, \mathscr{C}_1)}$:
409. $\mathsf{m}_0 \leftarrow \mathrm{Base2}(\mathscr{C}_0)$
410. $\mathsf{m}_1 \leftarrow \mathrm{Base2}(\mathscr{C}_1)$
411. send $(\mathsf{m}_0, \mathsf{m}_1)$
422. receive $\overline{\mathscr{C}}$
423. return $b_{\mathscr{A}}$

Fig. 17. The transition $\mathsf{Game}_{1,2} \to \mathsf{Game}_{1,3}$. Interactions are done in consecutive order.

This game is the inverse of $\mathsf{Game}_{1,1}$. Notice that on lines 403 and 405 in Figure 17, a random value is sampled for $b = 0$. This is the inverse of lines 203 and 205 from Figure 14.

Case $i > 1$. This is the general subcase similar to the previous subcase, up to the noticeable difference of using m_1, instead of $\perp$.

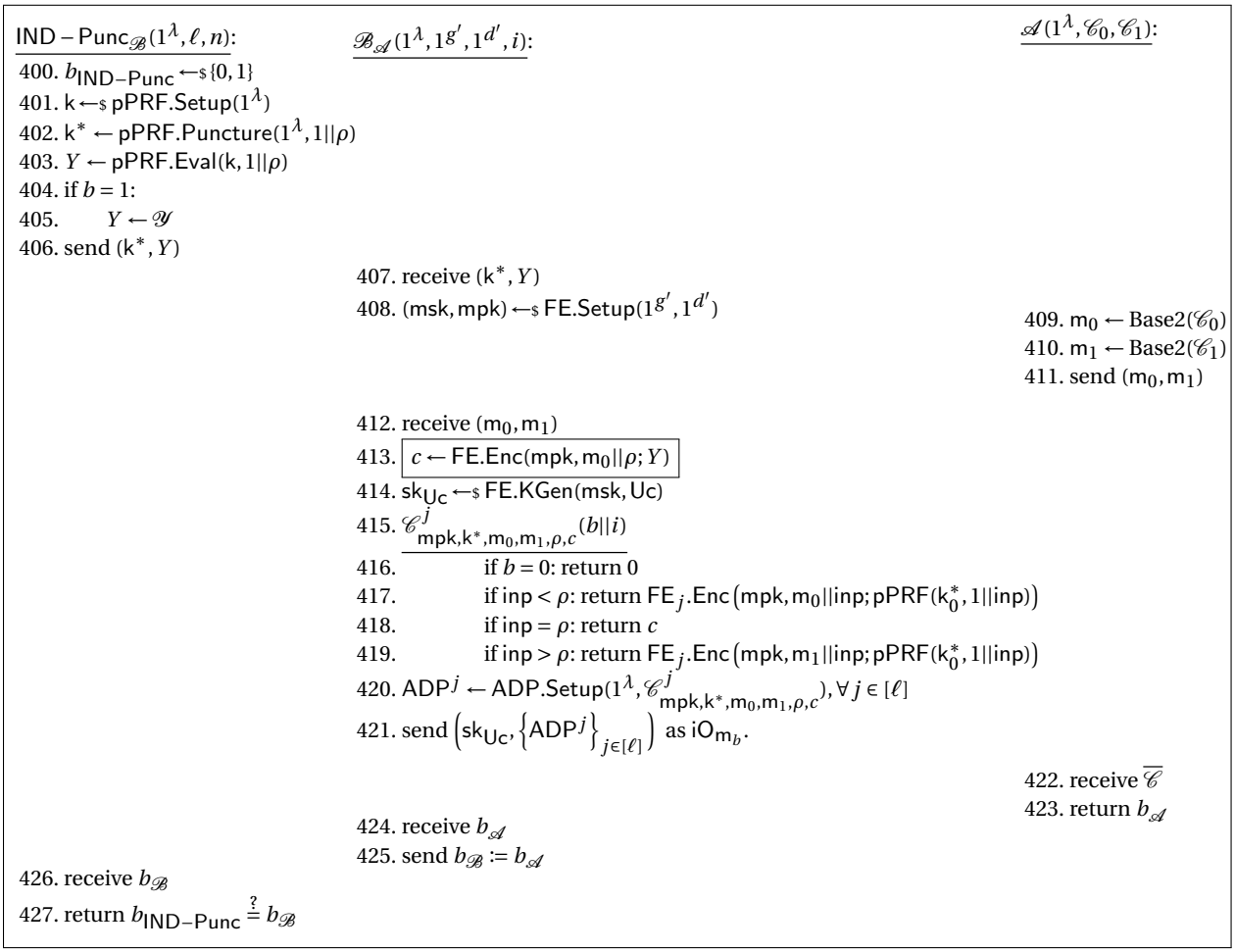


Fig. 18. The transition $\mathsf{Game}_{i,2} \rightarrow \mathsf{Game}_{i,3}$. Interactions are done in consecutive order.

The argument is the same as for the case $i = 1$, except that for the general case m_1 is encrypted within the circuit, on lines 419 (Figure 18).

D.5 End of Sequence i Game: $\mathsf{Game}_{i,4}^{j-1} \to \mathsf{Game}_{i,4}^{j}$

Remember that $\mathsf{Game}_{i,3} = \mathsf{Game}_{i,4}^{0}$. We will iterate from $j = 1$ to $\ell - 1$.

Case i=1. For the first batch, we will have to consider the particular case of encrypting the $\perp$ symbol on line 518 in Figure 19.

$\underline{\mathsf{IND\text{-}ADP}_{\mathscr{B}}(1^\lambda, \ell, n)}$:
500. $b_{\mathsf{IND\text{-}ADP}} \leftarrow\$ \{0,1\}$

$\underline{\mathscr{B}_{\mathscr{A}}(1^\lambda, 1^{g'}, 1^{d'}, i, j)}$:
504. receive $(\mathsf{m}_0, \mathsf{m}_1)$
505. $(\mathsf{msk}, \mathsf{mpk}) \leftarrow\$ \mathsf{FE.Setup}(1^{g'}, 1^{d'})$
506. $\rho_0 \leftarrow \boxed{2^n - 1}$.
507. $\rho_1 \leftarrow \boxed{2^n - 2}$.
508. $\mathsf{k}_{\mathsf{pPRF}} \leftarrow\$ \mathsf{pPRF.Setup}(1^\lambda)$
509. $c_0 \leftarrow\$ \mathsf{FE.Enc}(\mathsf{mpk}, \boxed{\mathsf{m}_1} || \rho_0; \mathsf{pPRF}(\mathsf{k}, \rho_0))$.
510. $c_1 \leftarrow\$ \mathsf{FE.Enc}(\mathsf{mpk}, \boxed{\mathsf{m}_0} || \rho_1; \mathsf{pPRF}(\mathsf{k}, \rho_1))$.
511. $w \leftarrow\$ \{0,1\}^n$
512. $\mathsf{k}^* \leftarrow \mathsf{pPRF.Puncture}(\mathsf{k}, 0||w)$.
513. $\mathsf{sk}_{\mathsf{Uc}} \leftarrow\$ \mathsf{FE.KGen}(\mathsf{msk}, \mathsf{Uc})$
514. $\mathscr{C}_0 := \underline{\mathscr{C}^j_{\mathsf{mpk},\mathsf{k}^*,\mathsf{m}_0,\mathsf{m}_1,\rho_0,c_0}(b||\mathsf{inp})}$
515. if $b = 0$: return 0
516. if $\mathsf{inp} < \rho_0$: return $\mathsf{FE}_j.\mathsf{Enc}\left(\mathsf{mpk}, \mathsf{m}_0||\mathsf{inp}; \mathsf{pPRF}(\mathsf{k}_0^*, 1||\mathsf{inp})\right)$
517. if $\mathsf{inp} = \rho_0$: return c_0
518. if $\mathsf{inp} > \rho_0$: return $\mathsf{FE}_j.\mathsf{Enc}\left(\mathsf{mpk}, \perp||\mathsf{inp}; \mathsf{pPRF}(\mathsf{k}_0^*, 1||\mathsf{inp})\right)$
519. $\mathscr{C}_1 := \underline{\mathscr{C}^j_{\mathsf{mpk},\mathsf{k}^*,\mathsf{m}_0,\mathsf{m}_1,\rho_1,c_1}(b||\mathsf{inp})}$
520. if $b = 0$: return 0
521. if $\mathsf{inp} < \rho_1$: return $\mathsf{FE}_j.\mathsf{Enc}\left(\mathsf{mpk}, \mathsf{m}_0||\mathsf{inp}; \mathsf{pPRF}(\mathsf{k}_1^*, 1||\mathsf{inp})\right)$
522. if $\mathsf{inp} = \rho_1$: return c_1
523. if $\mathsf{inp} > \rho_1$: return $\mathsf{FE}_j.\mathsf{Enc}\left(\mathsf{mpk}, \mathsf{m}_1||\mathsf{inp}; \mathsf{pPRF}(\mathsf{k}_1^*, 1||\mathsf{inp})\right)$
524. send $(\mathscr{C}_0, \mathscr{C}_1)$

525. receive $(\mathscr{C}_0, \mathscr{C}_1)$
526. $\overline{\mathscr{C}} \leftarrow\$ \mathsf{ADP.Setup}(1^\lambda, \mathscr{C}_b)$
527. send $\overline{\mathscr{C}}$

528. receive $\overline{\mathscr{C}}$
529. sample all $\left\{\mathsf{ADP}^{j'}\right\}_{j' \in [\ell], j' \neq j}$ as in prev game
530. send $\left(\mathsf{sk}_{\mathsf{Uc}}, \left\{\mathsf{ADP}^j\right\}_{j \in [\ell]}\right)$ as $\mathsf{iO}_{\mathsf{m}_b}$.

$\underline{\mathscr{A}(1^\lambda, \mathscr{C}_0, \mathscr{C}_1)}$:
501. $\mathsf{m}_0 \leftarrow \mathsf{Base2}(\mathscr{C}_0)$
502. $\mathsf{m}_1 \leftarrow \mathsf{Base2}(\mathscr{C}_1)$
503. send $(\mathsf{m}_0, \mathsf{m}_1)$

531. receive $\overline{\mathscr{C}}$
532. return $b_{\mathscr{A}}$

533. receive $b_{\mathscr{A}}$
534. send $b_{\mathscr{B}} := b_{\mathscr{A}}$

535. receive $b_{\mathscr{B}}$
536. return $b_{\mathsf{IND\text{-}ADP}} \stackrel{?}{=} b_{\mathscr{B}}$

Fig. 19. The "end of sequence" game decrements the value of ρ.

It is easy to see the two circuits built by the reduction $\mathscr{B}$ are equivalent. When queried in input point $2^n - 2$, the first circuit returns a ciphertext corresponding to m_0, while the second, will return as well, the challenge ciphertext corresponding to m_0. Both ciphertext are obtained under the same randomness term $\mathsf{pPRF}(\mathsf{k}^*, 1||2^n - 2)$. When queried in point $2^n - 1$, both circuits return an FE encryption of m_1 obtained w.r.t the same term $\mathsf{pPRF}(\mathsf{k}^*, 1||2^n - 1)$.

Case i>1. For the general case, the only difference is that m_1 appears in the circuit, as opposed to the special symbol $\perp$.

$\underline{\mathsf{IND\text{-}ADP}_{\mathscr{B}}(1^\lambda,\ell,n)}$:

500. $b_{\mathsf{IND\text{-}ADP}} \leftarrow\$ \{0,1\}$

$\underline{\mathscr{B}_{\mathscr{A}}(1^\lambda,1^{g'},1^{d'},i,j)}$:

504. receive $(\mathsf{m}_0,\mathsf{m}_1)$
505. $(\mathsf{msk},\mathsf{mpk}) \leftarrow\$ \mathsf{FE.Setup}(1^{g'},1^{d'})$
506. $\rho_0 \leftarrow \boxed{2^n - i}$.
507. $\rho_1 \leftarrow \boxed{2^n - (i+1)}$.
508. $\mathsf{k}_{\mathsf{pPRF}} \leftarrow\$ \mathsf{pPRF.Setup}(1^\lambda)$
509. $c_0 \leftarrow\$ \mathsf{FE.Enc}(\mathsf{mpk}, \boxed{\mathsf{m}_1} || \rho_0; \mathsf{pPRF}(\mathsf{k},\rho_0))$.
510. $c_1 \leftarrow\$ \mathsf{FE.Enc}(\mathsf{mpk}, \boxed{\mathsf{m}_0} || \rho_1; \mathsf{pPRF}(\mathsf{k},\rho_1))$.
511. $w \leftarrow\$ \{0,1\}^n$
512. $\mathsf{k}^* \leftarrow \mathsf{pPRF.Puncture}(\mathsf{k}, 0||w)$.
513. $\mathsf{sk}_{\mathsf{Uc}} \leftarrow\$ \mathsf{FE.KGen}(\mathsf{msk},\mathsf{Uc})$
514. $\mathscr{C}_0 := \underline{\mathscr{C}^j_{\mathsf{mpk},\mathsf{k}^*,\mathsf{m}_0,\mathsf{m}_1,\rho_0,c_0}(b||\mathsf{inp})}$
515. if $b=0$: return 0
516. if $\mathsf{inp} < \rho_0$: return $\mathsf{FE}_j.\mathsf{Enc}\left(\mathsf{mpk}, \mathsf{m}_0||\mathsf{inp}; \mathsf{pPRF}(\mathsf{k}_0^*, 1||\mathsf{inp})\right)$
517. if $\mathsf{inp} = \rho_0$: return c
518. if $\mathsf{inp} > \rho_0$: return $\mathsf{FE}_j.\mathsf{Enc}\left(\mathsf{mpk}, \mathsf{m}_1||\mathsf{inp}; \mathsf{pPRF}(\mathsf{k}_0^*, 1||\mathsf{inp})\right)$
519. $\mathscr{C}_1 := \underline{\mathscr{C}^j_{\mathsf{mpk},\mathsf{k}^*,\mathsf{m}_0,\mathsf{m}_1,\rho_1,c_1}(b||\mathsf{inp})}$
520. if $b=0$: return 0
521. if $\mathsf{inp} < \rho_1$: return $\mathsf{FE}_j.\mathsf{Enc}\left(\mathsf{mpk}, \mathsf{m}_0||\mathsf{inp}; \mathsf{pPRF}(\mathsf{k}_1^*, 1||\mathsf{inp})\right)$
522. if $\mathsf{inp} = \rho_1$: return c
523. if $\mathsf{inp} > \rho_1$: return $\mathsf{FE}_j.\mathsf{Enc}\left(\mathsf{mpk}, \mathsf{m}_1||\mathsf{inp}; \mathsf{pPRF}(\mathsf{k}_1^*, 1||\mathsf{inp})\right)$
524. send $(\mathscr{C}_0,\mathscr{C}_1)$

525. receive $(\mathscr{C}_0,\mathscr{C}_1)$
526. $\overline{\mathscr{C}} \leftarrow\$ \mathsf{ADP.Setup}(1^\lambda,\mathscr{C}_b)$
527. send $\overline{\mathscr{C}}$

528. receive $\overline{\mathscr{C}}$
529. sample all $\left\{\mathsf{ADP}^{j'}\right\}_{j'\in[\ell],j'\neq j}$ as in prev game
530. send $\left(\mathsf{sk}_{\mathsf{Uc}}, \left\{\mathsf{ADP}^j\right\}_{j\in[\ell]}\right)$ as $\mathsf{iO}_{\mathsf{m}_b}$.

$\underline{\mathscr{A}(1^\lambda,\mathscr{C}_0,\mathscr{C}_1)}$:

501. $\mathsf{m}_0 \leftarrow \mathsf{Base2}(\mathscr{C}_0)$
502. $\mathsf{m}_1 \leftarrow \mathsf{Base2}(\mathscr{C}_1)$
503. send $(\mathsf{m}_0,\mathsf{m}_1)$

531. receive $\overline{\mathscr{C}}$
532. return $b_{\mathscr{A}}$

533. receive $b_{\mathscr{A}}$
534. send $b_{\mathscr{B}} := b_{\mathscr{A}}$

535. receive $b_{\mathscr{B}}$
536. return $b_{\mathsf{IND\text{-}ADP}} \stackrel{?}{=} b_{\mathscr{B}}$

Fig. 20. The "end of sequence" for the general case ($i > 1$) decrements the value of ρ.

The game is almost identical as the one in Figure 19, except for the encodings of m_1 in line 523 in Figure 20.

D.6 Final Game

$\underline{\mathsf{IND\text{-}ADP}_{\mathscr{B}}(1^\lambda,\ell,n)}$:
600. $b_{\mathsf{IND\text{-}ADP}} \leftarrow\$ \{0,1\}$

$\underline{\mathscr{B}_{\mathscr{A}}(1^\lambda,1^{g'},1^{d'},i,j)}$:
604. receive $(\mathsf{m}_0,\mathsf{m}_1)$
605. $(\mathsf{msk},\mathsf{mpk}) \leftarrow\$ \mathsf{FE.Setup}(1^{g'},1^{d'})$
606. $\rho_0 \leftarrow \boxed{1}$.
607. $\rho_1 \leftarrow \boxed{2^n-1}$.
608. $\mathsf{k}_{\mathsf{pPRF}} \leftarrow\$ \mathsf{pPRF.Setup}(1^\lambda)$
609. $c \leftarrow\$ \mathsf{FE.Enc}(\mathsf{mpk}, \boxed{\mathsf{m}_1} || \rho_0; \mathsf{pPRF}(\mathsf{k},\rho_0))$.
610. $w \leftarrow\$ \{0,1\}^n$
611. $\mathsf{k}_0^* \leftarrow \mathsf{pPRF.Puncture}(\mathsf{k}, 0||w)$.
612. $\mathsf{sk}_{\mathsf{Uc}} \leftarrow\$ \mathsf{FE.KGen}(\mathsf{msk},\mathsf{Uc})$
613. $\mathscr{C}_0 := \underline{\mathscr{C}^j_{\mathsf{mpk},\mathsf{k}_0^*,\mathsf{m}_0,\mathsf{m}_1,\rho_0,c_0}(b||i)}$
614. if $b=0$: return 0
615. if $i<\rho_0$: return $\mathsf{FE}_j.\mathsf{Enc}\left(\mathsf{mpk},\mathsf{m}_0||i;\mathsf{pPRF}(\mathsf{k}_0^*,1||i)\right)$
616. if $i=\rho_0$: return c
617. if $i>\rho_0$: return $\mathsf{FE}_j.\mathsf{Enc}\left(\mathsf{mpk},\mathsf{m}_1||i;\mathsf{pPRF}(\mathsf{k}_0^*,1||i)\right)$
618. $\mathscr{C}_1 := \underline{\mathscr{C}^j_{\mathsf{mpk},\mathsf{k}_1^*,\mathsf{m}_0,\mathsf{m}_1,\rho_1,c_1}(b||i)}$
619. if $b=0$: return 0
620. if $i<\rho_1$: return $\mathsf{FE}_j.\mathsf{Enc}\left(\mathsf{mpk},\mathsf{m}_0||i;\mathsf{pPRF}(\mathsf{k}_1^*,1||i)\right)$
621. if $i=\rho_1$: return c
622. if $i>\rho_1$: return $\mathsf{FE}_j.\mathsf{Enc}\left(\mathsf{mpk},\mathsf{m}_1||i;\mathsf{pPRF}(\mathsf{k}_1^*,1||i)\right)$
623. send $(\mathscr{C}_0,\mathscr{C}_1)$

624. receive $(\mathscr{C}_0,\mathscr{C}_1)$
625. $\overline{\mathscr{C}} \leftarrow\$ \mathsf{ADP.Setup}(1^\lambda,\mathscr{C}_b)$
626. send $\overline{\mathscr{C}}$

627. receive $\overline{\mathscr{C}}$
628. sample all $\left\{\mathsf{ADP}^{j'}\right\}_{j'\in[\ell],j'\neq j}$ as in prev game
629. send $\left(\mathsf{sk}_{\mathsf{Uc}},\left\{\mathsf{ADP}^j\right\}_{j\in[\ell]}\right)$ as $\mathsf{iO}_{\mathsf{m}_b}$.

632. receive $b_{\mathscr{A}}$
633. send $b_{\mathscr{B}} := b_{\mathscr{A}}$

634. receive $b_{\mathscr{B}}$
635. return $b_{\mathsf{IND\text{-}ADP}} \stackrel{?}{=} b_{\mathscr{B}}$

$\underline{\mathscr{A}(1^\lambda,\mathscr{C}_0,\mathscr{C}_1)}$:
601. $\mathsf{m}_0 \leftarrow \mathrm{Base2}(\mathscr{C}_0)$
602. $\mathsf{m}_1 \leftarrow \mathrm{Base2}(\mathscr{C}_1)$
603. send $(\mathsf{m}_0,\mathsf{m}_1)$

630. receive $\overline{\mathscr{C}}$
631. return $b_{\mathscr{A}}$

Fig. 21. The final game switches the limiting variable ρ from 0 back to 2^n-1.

The final game is used to be identical to the experiment that has m_1 encrypted. In doing so, the values of ρ are switched, and the same is done with the values of m_0 and m_1.