Do Affine Determinant Programs Allow to Get iO From Standard Assumptions?

Abstract. Affine determinant programs (ADP), introduced in [8], were proposed as an alternative path to achieve indistinguishability obfuscation. Recently, the inceptive candidate has been shown insecure for several counterexamples within the work of [33], that also proposed a heuristic patch.

In this paper, we consider compact functional encryption (cFE, Pass *et al.* [27]) as the cornerstone used to build indistinguishability obfuscation (iO). We prove the indistinguishability of a different instantiation of single-key cFE, using indistinguishably-secure ADPs. The core idea is to instantiate the exponentially-efficient indistinguishability obfuscator (XiO) used in [27] with a particular ADP for succinct functional encryption (sFE).

Most importantly, we extend existing results from literature: by considering a circuit's binary decision diagram (BDD) representation, we obtain a set of functional and topological conditions on BDD that make the BDD admit indistinguishably-secure ADP implementations. We provide a partially different proof of these conditions.

Finally, we focus on finding a missing piece of this puzzle: ADP-admissible sFE. We show that the encryption procedure of a RLWE-based, levelled, single-key succinct functional encryption scheme supporting circuits in complexity class NC^1 that has a relatively simple algebraic structure, is ADP-admissible. These facts provide an affirmative answer for the problem of obtaining cFE (and then iO) from affine determinant programs.

Keywords: compact FE, BDD, affine determinant programs.

1 Introduction

Cryptographic obfuscation aims at making programs unintelligible while preserving their functionality. Indistinguishability obfuscation (iO) [5] is a particular cryptographic obfuscation technique guaranteeing that it is hard to tell apart which of the functionally equivalent circuits have been used to generate an obfuscated version of one of them (which is made available). Immediate applications of iO include preventing software piracy: for instance, an operating system provider may release a unique and obfuscated version of the original executable to each of its clients. Software piracy may be prevented by tracking the version that was replicated as well as the client that replicated it.

Further pre-eminent cryptographic applications of indistinguishability obfuscation include: non-interactive multi-partite key exchange (NIKE) [12], functional encryption (FE) [11,17] or multilinear maps [16,15].

Recently, iO has been shown to be realizable assuming well-established cryptographic assumptions, thanks to a series of intricate works by Jain, Lin and Sahai, culminating with [25]. The proposed instantiation of iO uses as a building block a very particular variation of functional encryption, denoted *compact* FE. Functional encryption itself is an encryption mechanism that allows high-precision decryption: given a ciphertext corresponding to some message m and a functional key issued for some function f, the decryptor should recover $f(\mathsf{m})$, and ideally nothing else about m. Various flavours of FE have been shown to imply iO, including *compact* functional encryption [27].

1.1 Overview of ADPs and Their Variant from [8]

On a totally different path from mainstream approaches mentioned above, Bartusek *et al.*[8] proposed a simpler candidate indistinguishability obfuscator operating on affine determinant program.

Affine Determinant Programs. The high level idea behind introducing affine determinant programs is as follows: one can consider algebraically simpler schemes for circuits in NC^1, which suffice for generalizing iO to all polynomial-sized circuits.

An affine determinant program capable of evaluating inputs of length n bits consists of a set of $n+1$ matrices: a "base" matrix $\mathbf{T}_0$, and n square matrices of dimension m: $\{\mathbf{T}_i : i \in \{1,\dots,n\}\}$ with elements over some finite field $\mathbb{F}_p$. Let this affine determinant program, corresponding to some functionality $f \in \mathsf{NC}^1$, be denoted by ADP. The evaluation of ADP given an n-bit input string m is as simple as:

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

where m_i is the i^{th} bit of m.

How to build simple ADPs in a nutshell. Building ADPs is not a straightforward step. Oversimplified, it is imperative to start with functions in NC^1, as they admit binary decision diagrams of polynomial size[6]. One example of such a diagram for a simple XOR function is in Figure 1.

Once the binary decision diagram (BDD) of the desired f has been obtained, we note that every non-trivial[1] vertex depends on some input bit – say m_i – and has two possible outgoing edges (see for instance the example depicted in Figure 1).

Next, an input – say $\mathsf{m} \coloneqq \vec{0}$ – is considered, and we obtain a path in the directed acyclic graph (DAG), going from the start node to a terminal node value (0 or 1).

Then, the adjacency matrix of this path within the DAG of the BDD is obtained over $\mathbb{F}_p$. In further steps, in rough terms, a second diagonal is added, the first line and last columns are removed. This constitutes the skeleton of the base matrix $\mathbf{T}_0$, which is to be further multiplied with randomization matrices.

[1] Start and Terminal nodes do not matter here.

To build the remaining $\mathbf{T}_i$, we consider the binary decision diagram and consider the path corresponding to input $00\ldots010\ldots0$, where 1 occurs in the i^{th} position. This gives us another adjacency matrix, which we post-process as explained above. Then, we set the core of $\mathbf{T}_i$ as the difference $\mathbf{T}_{0\ldots0} - \mathbf{T}_{0\ldots010\ldots0}$, which will be further randomized.

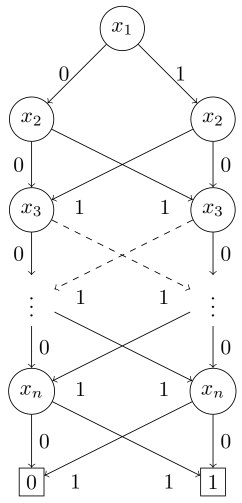

Fig. 1. An example of BDD corresponding to $x_1 \oplus \ldots \oplus x_n$. Observe that each node x_i has two possible outgoing edges, a path through a graph being chosen based on the values settled for each x_i.

The Randomization Techniques and the Obfuscator Introduced in [8]. The flavour of affine determinant programs described in [8] are meant to provide an efficient way to obfuscate *all* polynomial sized circuits. To this end, the authors point out several issues[2] with the core obfuscator that was described in the paragraphs above and provide heuristic countermeasures.

We recap briefly the changes the work of [8] enforces over the set $\{\mathbf{T}_0, \mathbf{T}_1, \ldots, \mathbf{T}_n\}$ which are fully motivated in their paper. These include: (1) random local substitutions by adding dummy vertices; (2) matrices consisting of small and even noise values are added to $\mathbf{T}_i$; (3) block diagonal matrices are used to increase the dimension and inject more randomness into the matrices on which the determinant will be computed; finally, (4) randomizing matrices are used.

The final transformation is a composition of the the previous functions.

Technically, the transformation above gives rise to an ADP, as the evaluation formula in Equation (1) still holds. The claim made is that this ADP can act directly as an obfuscator for all polynomial sized circuits, without further modifications.

The attack by Yao, Chen and Yu. Interestingly, even with the countermeasures, the proposed obfuscator sketched above does not guarantee indistinguishability for several classes of functions admitting relatively simple circuit representations. In [33], the authors put forth a series of beautiful attack ideas, by considering relatively simple functions. They show the original scheme's random local substitution mechanism is deficient for such functionalities and suggest an informal patch.

1.2 Our First Result and the Techniques Employed

Our work targets indistinguishability-security for affine determinant programs (IND-ADP) for a "custom-made" class of circuits, which are needed to achieve compact functional encryption. The results herein can be summarized as: (1) showing that IND-ADP is sufficient to achieve selectively-secure compact functional encryption; (2) proving conditions on the behaviour/circuit topology of

[2] We would like to emphasize that the straightforward construction presented above ma be suitable to *iO*-obfuscate specific classes of functions, but they will certainly fail in obfuscating all programs representable through polynomial sized circuits.

programs in order to admit IND-ADP-secure implementations. The main question we leave open at this stage is related to providing concrete implementations of circuits fulfilling the aforementioned conditions.

***iO* from (Compact) Functional Encryption.** Pass *et al.* [27] have shown that *single-key compact* functional encryption can be obtained assuming the existence of (1) *succinct* functional encryption for NC^1 (denoted sFE throughout this work), (2) *puncturable* pseudorandom functions – denoted pPRF – and (3) an *exponentially-efficient* indistinguishability obfuscator.

It is well known that selectively-secure compact FE can be used to instantiate iO [4,14]. [27] introduces a method to achieve compact FE using exponentially efficient iO, where the latter construction is sightly more efficient than a lookup table (the trivial obfuscator). The XiO is used to obfuscate a circuit that produces succinct functional encryption ciphertexts (see [27, Section 4.1]). Notably, a puncturable PRF is used to generate the random coins needed by the succinct sFE.Enc procedure:

$$\mathscr{C}_{\mathsf{mpk},\mathsf{k},\mathsf{m}}(i) \coloneqq \mathsf{sFE.Enc}\left(\mathsf{mpk}, (\mathsf{m}, i); \mathsf{pPRF}(\mathsf{k}, i)\right) \ . \tag{2}$$

The compact ciphertext generated by the compact cFE is simply an XiO-obfuscated version of $\mathscr{C}_{\mathsf{mpk},\mathsf{k},\mathsf{m}}(i)$, while a functional-key is generated by querying $\mathsf{sFE.KGen}(\mathsf{msk}_{\mathsf{sFE}}, \mathscr{C}_f(\cdot,\cdot))$, such that

$$\mathscr{C}_f(\mathsf{m}, i) \coloneqq f_i(\mathsf{m}). \tag{3}$$

Simply, the master keys (public and secret) of cFE are the same to the ones of the underlying sFE.

The primitive that is missing in the picture is the XiO obfuscator. Several recent works show instantiation of XiO from various assumptions which are by now considered well-established.

Instantiating XiO from ADPs. Our main idea consists in using affine determinant programs strictly for the problem of instantiating the XiO used to derive cFE, while leaving the remaining parts of the scheme intact. To this end, we will consider a circuit having its (oversimplified) inner structure described in Figure 2. The idea is that we will set the limit ρ to the maximum input, and the circuit is expected to always produce sFE encryptions of m. We explain below how the ADP topology forces us to consider all branches occurring in the proof in the real scheme (as well as in proofs).

Our approach is to simply use ADPs to "obfuscate" circuits like the one above. For technical reasons, we work with ADPs having elements over $\mathbb{F}_2$. This implies the usage of ν circuits, ν being the length of the succinct functional encryption (sFE) ciphertext used by each circuit; thus, we employ one circuit to produce a single output bit.

The second noticeable difference (to be motivated soon) is linked to the usage of puncturable pseudorandom functions[3]. Instead of using the normal evaluation

[3] A puncturable PRF is a PRF with two modes of operation: a normal one, using a key k and an input m, producing (pseudo-)random values y; and a punctured mode,

$$\underline{\underline{\mathscr{C}_{\mathsf{mpk},\mathsf{k},\mathsf{m},\rho}\left(i\right):}}$$

$$\mathsf{CT}_i \leftarrow \begin{cases} \mathsf{sFE.Enc}\left(\mathsf{mpk}, (\mathsf{m}, i); \mathsf{pPRF}(\mathsf{k}, 1||i)\right) , & \text{if } i < \rho \\ c, \text{where } c \leftarrow \mathsf{sFE}_{\ell}.\mathsf{Enc}\left(\mathsf{mpk}, (\mathsf{m}, i); \mathsf{pPRF}(\mathsf{k}, 1||i)\right) , & \text{if } i = \rho \\ \mathsf{sFE.Enc}\left(\mathsf{mpk}, (\$, i); \mathsf{pPRF}(\mathsf{k}, 1||i)\right) , & \text{otherwise} \end{cases}$$

return CT_i

Fig. 2. The circuit outputs (bits of) a succinct functional encryption (sFE) ciphertext. pPRF stands for a puncturable pseudorandom function. To decrypt, one will apply a functional key corresponding to some function f over (m, i) and recover the i^{th} bit of $f(\mathsf{m})$.

function of the pPRF, we use the punctured evaluation algorithm. The main constraint is the different topology of the normal evaluation vs punctured evaluation: for most puncturable PRFs, these circuits differ[4]. The trick we are using is to have a slightly longer input domain, and puncture a key in some random input starting with $0\ldots$, while always evaluating the pPRF in inputs starting with $1||\ldots$. In this way, we are guaranteed that we can reuse exactly the same binary decision diagram to deal with both punctured keys. Stated differently, we want to preserve the topology of the underlying BDD behind our ADP, and only vary some "sensitive" variables (we will refer to them as "nimv").

By recycling the previous argument, our strategy is to preserve the topology of the underlying BDD, and only change a designated set of variables, the nimv. This is the reason, while in our security proof, we will change the k, the ρ and m, but leave the BDD unaltered.

Security considerations for ADPs: topology, behaviour and "nimvs". Proving the security of our cFE instantiation from IND-ADP secure affine determinant programs is relatively straightforward, although we need two layers of nested hybrid games.

The intuition behind our proof technique is as follows: we want to successively replace each of the ν circuits described in Figure 2 with ones using a punctured key for the challenge input – say j. Then, we will use the indistinguishability of the pPRF in the punctured point as well as a genuine randomness to obtain the ciphertext c. Finally, we will use the indistinguishability of the succinct functional encryption scheme to switch between the challenge messages m_0 to m_1. Extra game hops are used to transit from settings where ρ will be updated.

Immediately, it can be observed that values such as k or ρ represent *non-input mutable variables*, or nimv: they are highly sensitive, and leaking them means breaking the indistinguishability of ADPs. On the other hand these nimv values are not-input dependent. In some sense, one shall imagine the circuit in Figure 2 as taking 2-inputs: nimv and m.

where a punctured key k^*, punctured in some point m^*, can be used to compute all PRF values, like in the normal mode, except for the PRF value corresponding to the puncture input m^*.

[4] Think for instance at the length of the punctured key in GGM's punctured evaluation compared to the length of the key for GGM's normal evaluation.

A "proto"-ADP can be built with $|\mathsf{nimv}|+n+1$ matrices; during the setup, the nimv values are settled as the $\mathsf{k}, \rho, \mathsf{m}$ are chosen. Based on their value, the ADP is obtained from the "proto"-ADP by summing up the nimv matrices $\{\mathbf{T}_i\}_{i\in[|\mathsf{nimv}|]}$ to the "proto"-$\mathbf{T}_0$. This will represent the new $\mathbf{T}_0$ to be used for inputs of length n. From this perspective, we can say that ADP will "depend" on the nimv values. We will sometimes refer to ADPs as having "embedded" nimv values.

The beauty behind ADPs. Obfuscation relates to hiding the structure of programs. Essentially, we want to achieve the same for ADPs: preserve their structure, but hide the nimv values. The main problem in proving ADP indistinguishability is the lack of average-case assumptions: perhaps excepting multivariate system of equations, there is nothing in the cryptographic-assumption landscape to be easily related to ADPs.

In this work, we proceed differently. We consider perfect security[5]:

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

Namely, for any R_0 used to ADP-encode a circuit with nimv_0 sensitive variables, there is unique randomness term R_1 use to encode the same circuit w.r.t. nimv_1. From previous works, we know that this perfect security notion provides us with a set of functional conditions on $\mathscr{C}_{\mathsf{nimv}}$[6] and its BDD topology that may allows us to achieve perfect security notion for *one* specific class [7]. These conditions are to be introduced informally below, all but the 6th being unchanged. But before, we introduce notations with $\mathfrak{C}_{a,b}$ denoting a class of circuits of depth a and input length b; we restate the result in the first result:

Theorem 1 (Informal). *Let sFE stand for a succinct functional encryption scheme supporting circuits of depth d' and input n and having its encryption procedure represented as circuits of depth d. Let $\mathfrak{C}_{d,|nimv|+n}$ stand for a class of circuits of depth d and input length n. Let ADP be an IND-ADP-secure affine determinant program for the construction defined in Equation (2), satisfying conditions (1) → (6) below. Then, there is a selectively indistinguishably-secure compact functional encryption scheme for functions in class $\mathfrak{C}_{d',n}$ supporting one functional key.*

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

Condition 2 *Every $\mathscr{C} \in \mathfrak{C}_{d,|nimv|+n}$ implements a two input function $f : \{0,1\}^{|nimv|+n} \to \{0,1\}$.*

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

$$\Pr\left[f(|nimv|, \mathsf{inp}) = 1 | (|nimv|, \mathsf{inp}) \leftarrow \mathcal{R}(1^\lambda, f)\right] > \frac{1}{\mathsf{poly}(|nimv|+n)} \ . \quad (4)$$

[5] Note that this will **not** bring a perfectly secure compact FE, as the later proof relies on the security of succinct FE and puncturable PRFs.

[6] For instance, it should be non-vanishing, a result matching an earlier finding of [21].

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

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

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

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

Condition 6 *On any local path, any node-index of a vertex depending on input is greater than any node-index of a vertex depending on* $|\mathsf{nimv}|$*.*

We detail on such conditions in Sections 1.3 and 4 and provide a partially new, alternative, proof in Appendix A.2. The main question concerns if functions fulfilling such conditions exists. We address it below, and answer it in affirmative.

1.3 ADP-Based Compact FE through RLWE-Based Succinct FE

Our second result is that selectively indistinguishably-secure cFE can be built from affine determinant programs that produce succinct functional encryption ciphertexts. All in all, the first result imposes constraints on the order of nodes within the binary decision diagrams used to instantiate the ADPs. The constraints are used to show the indistinguishability of two functionally equivalent ADPs, which then can be used during the proof of cFE. The *most demanding constraint* requires to reorder nodes in a BDD such that, on any internal path, nodes depending on sensitive variables (such as keys or differing messages) occur before nodes depending on inputs.

It is relatively easy to remark that the first four conditions above are easy to accomplish by any function that is (1) in NC^1, (2) bivariate, (3) non-constant and (4) functionally equivalent to one built under different nimvs. Condition (5) is needed for the proof, while the real difficulty stands in achieving (6). The second part of our work focuses exclusively on how to tackle the latter two. The problem left open in the first result is to identify succinct functional encryption schemes as well as puncturable PRFs, such that the size of corresponding ADPs is polynomial, and they satisfy the constraints above.

Our central result for the second part regards the applicability of a specific *decomposable* functional encryption scheme introduced by Agrawal and Rosen in [2], that provides a poly-sized binary decision diagram, and then an ADP that satisfies constraints defined in Theorem 1.

Theorem 2 (Informal). *Let* sFE *stand for the functional encryption scheme defined in [2] supporting a single functional key derivation query. Then there exists an 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 n bits[7]. If the encryption algorithm obtains a ciphertext $\mathsf{CT} = \{\mathsf{CT}_1, \ldots, \mathsf{CT}_n\}$ 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*.

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) [13] and encrypts each bit of the plaintext, independently, as follows: at first multiplication level – the ciphertext consists of

$$\mathsf{CT}_i^1 \leftarrow a \cdot s + p_0 \cdot e + \mathsf{m}_i \quad \in \mathcal{R}_{p_1} \tag{5}$$

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

$$a' \cdot s + p_1 \cdot e' + \mathsf{CT}_i^1 \ \in \ \mathcal{R}_{p_2} \quad \text{and} \quad a'' \cdot s + p_1 \cdot e'' + (\mathsf{CT}_i^1 \cdot s) \ \in \mathcal{R}_{p_2} \ . \tag{6}$$

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 addition 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:

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

and use 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 Equation (2), 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 (we also briefly compare it to an alternative sFE scheme in Appendix I). Still, there are *two* major constraints that prevents us to limpidly think about binary decision diagrams that output AR17 ciphertexts.

Firstly, 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.

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

Secondly, we use puncturable pseudorandom functions for NC^1 in a black box way, and we may need to delve into pPRFs instantiations as well. There exists relatively simple pPRFs in NC^1 due to two construction by Boneh, Lewi, Montgomery and Raghunathan [10, page 2-3, Equations (1.1) and (1.2)].

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

$$\mathcal{R}/p_1\mathcal{R} \cong \big(\mathcal{R}/\mathcal{I}_1 \times \ldots \times \mathcal{R}/\mathcal{I}_N\big) \tag{7}$$

where both the addition and multiplication of two elements is done coordinate-wise and $p_1\mathcal{R} \coloneqq \cap_{i=1}^{N}\mathcal{I}_i$ and each $\mathcal{R}/\mathcal{I}_i$ 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 explicitly form written as:

$$\Big(\sum_{j=0}^{N-1} a_{(j)} \cdot X^j\Big) \cdot \Big(\sum_{j=0}^{N-1} s_{(j)} \cdot X^j\Big) + p_0 \cdot \Big(\sum_{j=0}^{N-1} e_{(j)} \cdot X^j\Big) + \mathsf{m}_i \tag{8}$$

becomes with respect to this CRT embedding:

$$\begin{aligned}\Big(&\overline{a_{(0)}} \cdot \overline{s_{(0)}} + p_0 \cdot \overline{e_{(0)}} + \mathsf{m}_i, \ldots, \\ &\qquad \overline{a_{(j)}} \cdot \overline{s_{(j)}} + p_0 \cdot \overline{e_{(j)}} + \mathsf{m}_i, \ldots, \\ &\qquad\qquad \overline{a_{(N-1)}} \cdot \overline{s_{(N-1)}} + p_0 \cdot \overline{e_{(N-1)}} + \mathsf{m}_i\Big)\end{aligned} \tag{9}$$

In Equation (9), we use the fact that any constant c that does not need to be reduced modulo any ideal $\mathcal{I}_i$ in Equation (7) 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 binary decision diagrams producing ciphertexts with respect to the CRT embedding.

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

$$\overline{a_{(j)}} \cdot \overline{s_{(j)}} + p_0 \cdot \overline{e_{(j)}} + \mathsf{m}_i \tag{10}$$

The values of each $\overline{a_{(j)}}$ and p_0 are public and they can be "hardcoded" within the binary decision diagram. 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:

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

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 (10) can be rewritten as:

$$g(i) + \Big(\sum_{k=0}^{N-1} p_0 \cdot h_k(i) \Big) + \mathsf{m}_i \tag{11}$$

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

Step 2. We want to ensure that the BDD representation has, on any path containing nimv-dependent nodes, the nodes depending on nimvs "on top of" the nodes depending on input. Our key observation is related to a *locality* property.

We observe the BDD corresponding to Equation (11) can be visualized as a binary decision diagram simulating addition with "carry" (and in the end the mod operand), where the additions correspond to the terms involving the function g.

To handle local node ordering (Condition 6), we introduce nodes depending on the (dummy) input bit 1 between any two addition layers in the BDD: considering the base matrix $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$, we will introduce between every two addition layers a "dummy" node depending on input 1 that points to the terminal node 1. This acts as a "switch[8]" in the BDD representation, as it interrupts the flow from the start node to the terminal node, between additions performed in Equation (11).

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 $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$.

Step 3. Everything that has been done up to this stage was in relation with ADP representation for level 1 ciphertexts in AR17. Now, we show how to extend to higher level ciphertexts.

Fortunately, the idea is relatively straightforward, just that the higher the level, the more complex Equation (11) becomes. This is because it needs to cope with the multiplication of functions of type g. We present this extension, in full, in the main part of this work. Still, in Appendix J we also point out that 3 levels may suffice for the problem at hand.

Roadmap. The rest of the work is divided as follows: in Section 2, we introduce the standard notations to be adopted throughout the paper, followed by the definitions of the primitives that we use as building blocks. Section 2.4 reviews the construction of randomized encodings from binary decision diagrams. Appendix B reviews the status-quo of attacks on ADPs and provides a novel insight into the security of plain ADPs. In Section 3 we prove the cFE based on ADPs, while in Section 4 we review the conditions for getting perfect security.

[8] Like an electrical switch.

In Appendix A we provide a detail look into ADPs and prove they achieve indistinguishability. In Appendix D we provide an overview of the decomposable FE scheme in [2]. Finally, Appendix F shows its BDD representation fulfils the constraints put forth in Theorem 1. Prior reviews are provided in Appendix K.

2 Background

Algorithmic and Mathematical Definitions. In our work, an algorithm is equivalent to a Turing machine. The security parameter denoted by $\lambda \in \mathbb{N}^*$ is provided to all algorithms in its unary representation, denoted by 1^λ. We assume that an algorithm is randomized, otherwise we clearly state that it is deterministic. "Probabilistic polynomial-time" in the security parameter is abbreviated by PPT. Given a randomized algorithm $\mathcal{A}$ we denote the action of running $\mathcal{A}$ on input(s) $(1^\lambda, x_1, \dots)$ with uniform random coins r and assigning the output(s) to $(y_1, \dots)$ by $(y_1, \dots) \leftarrow_\$ \mathcal{A}(1^\lambda, x_1, \dots; r)$. When $\mathcal{A}$ is given oracle access to some procedure $\mathcal{O}$, we write $\mathcal{A}^{\mathcal{O}}$. We denote the cardinality of some finite set S by $|S|$, and the action of sampling an uniformly at random element x from X by $x \leftarrow_\$ X$. We let bold variables such as $\mathbf{w}$ represent column vectors. Similarly, bold capitals usually stand for matrices (e.g. $\mathbf{A}$). A subscript $\mathbf{A}_{i,j}$ indicates an entry in the matrix. We abuse notation and write $\alpha^{(u)}$ to denote that variable α is associated to some entity u. For any variable $k \in \mathbb{N}^*$, we define $[k] \coloneqq \{1, \dots, k\}$. A real-valued function $\textsc{Negl}(\lambda)$ is negligible if $\textsc{Negl}(\lambda) \in \mathcal{O}(\lambda^{-\omega(1)})$. We denote the set of all negligible functions by $\textsc{Negl}$. Throughout the paper $\perp$ stands for a special error symbol. We use $||$ to denote concatenation. We consider circuits as the prime model of computation for representing (abstract) functions. Unless stated otherwise, we use n to denote the input length of the circuit, s for its size and d for its depth.

2.1 Functional Encryption Scheme - Public-Key Setting

Definition 1 (Functional Encryption - Public Key Setting[11]). *Let $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in N}$ be an ensemble, where $\mathcal{F}_\lambda$ is a finite collections of functions $f : \mathcal{M}_\lambda \to Y_\lambda$. A functional encryption scheme* FE *in the public-key setting consists of a tuple of* PPT *algorithms* (Setup, KGen, Enc, Dec) *such that:*

- $(\mathsf{msk}, \mathsf{mpk}) \leftarrow_\$ \mathsf{FE.Setup}(1^\lambda)$: *takes as input the unary representation of the security parameter λ and outputs a pair of master secret/public keys.*
- $\mathsf{sk}_f \leftarrow_\$ \mathsf{FE.KGen}(\mathsf{msk}, f)$*: given the master secret key and a function f, the (randomized) key-generation procedure outputs a corresponding* sk_f*.*
- $\mathsf{CT} \leftarrow_\$ \mathsf{FE.Enc}(\mathsf{mpk}, \mathsf{m})$*: the randomized encryption procedure encrypts the plaintext* m *with respect to* mpk.
- $\mathsf{FE.Dec}(\mathsf{CT}, \mathsf{sk}_f)$*: decrypts the ciphertext* CT *using the functional key* sk_f *in order to learn a valid message $f(\mathsf{m})$ or a special symbol $\perp$, in case the decryption procedure fails.*

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

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

A public-key functional encryption scheme FE *is semantically secure if there exists a stateful* PPT *simulator* $\mathcal{S}$ *such that 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, where the $\mathsf{IND\text{-}FE\text{-}CPA}$ *experiment is described in Figure 3 (left).*

Furthermore, we introduce efficiency constraints for functional encryption. We call an FE *scheme succinct if the size of the ciphertext grows as a polynomial in the depth, and not the size of the supported class of circuits:* $|\mathsf{CT}| \leq \mathsf{poly}(\lambda, d)$*, where* d *is the maximal depth of the circuit implementing some* $f \in \mathcal{F}_\lambda$*.*

We call an FE *scheme weakly sublinear compact if there exists* $\epsilon > 0$ *such that for every* $\lambda \in \mathbb{N}^*$*, every* $\mathsf{m} \in \mathcal{M}_\lambda$ *and every* $\mathsf{mpk} \leftarrow_\$ \mathsf{FE.Setup}(\lambda)$ *we have that*

$$|\mathsf{CT}| \leq \mathsf{poly}(\lambda, |\mathsf{m}|) \cdot s^{1-\epsilon}$$
$$\mathsf{Time}(\mathsf{FE.Enc}(\mathsf{mpk}, \mathsf{m})) \leq \mathsf{poly}(\lambda, |\mathsf{m}|, s)$$

where s *is the maximal size of a circuit implementing some function in* $\mathcal{F}_\lambda$*.*

2.2 Indistinguishability Obfuscation

We use the formal indistinguishability definition for an obfuscator of a class of circuits [26].

Definition 2 (Indistinguishability Obfuscation (*iO*) for a circuit class). *A* PPT *algorithm* iO *is an indistinguishability obfuscator for a class of circuits* $\{\mathscr{C}_\lambda\}_{\lambda \in \mathbb{N}^*}$ *if the following conditions are satisfied:*

- ***Correctness****:* $\Pr\left[\forall x \in \mathcal{D}, \mathscr{C}(x) = \overline{\mathbf{C}}(x) | \overline{\mathbf{C}} \leftarrow_\$ \mathrm{iO}(\mathscr{C})\right] = 1$.
- ***Indistinguishability****:*

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

is negligible, where $\mathcal{D}$ *is the input domain of the circuits* C*.*

Exponentially Efficient* iO*: *is an* iO *obfuscator with the additional constraint that its size is less than* $\mathsf{poly}(\lambda, |\mathscr{C}|) \cdot 2^{n \cdot (1-\epsilon)}$*.*

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

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

```
$\mathsf{PRF}^{\mathcal{A}}_{\mathsf{PRF}}(\lambda)$:
$b \leftarrow\!\!\$ \{0,1\}$; $\mathsf{L} \leftarrow \emptyset$
$K \leftarrow\!\!\$ \mathsf{Setup}(1^\lambda)$
$b' \leftarrow\!\!\$ \mathcal{A}^{\mathsf{Prf}(\cdot)}(1^\lambda)$
**return** $(b' \overset{?}{=} b)$

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

Fig. 3. Games defining the security of pseudorandom functions (right), as well as FE security (left).

2.3 Affine Determinant Programs

The introduction gives an overview over affine determinant programs, and how to evalaute them. Here, we provide the formal definition, by referring to the work of [8]. We postpone instantiations from randomized encodings of binary decision diagrams to Section 2.4 and its variation to Appendix B.

Definition 3 (Affine Determinant Programs). *An affine determinant program over $\mathbb{F}_p$, parameterized by input length n and dimension m, consists of two algorithms:*

$\mathsf{Prog} \leftarrow\!\!\$ \mathsf{ADP.Setup}(1^\lambda, \mathscr{C})$*: the* Setup *is a randomized algorithm such that given a circuit description $\mathscr{C}$ of some function $f : \{0,1\}^n \to \{0,1\}$, output a set of $n+1$ matrices:*

$$\mathsf{Prog} := (\mathbf{T}_0,\ \mathbf{T}_1,\ \ldots,\ \mathbf{T}_n) \in \mathbb{F}_p^{m \times m}$$

$b \leftarrow \mathsf{ADP.Eval}(\mathsf{Prog}, \mathsf{inp})$*: is a deterministic procedure, that given the program* Prog *and some input* m*, return a binary value b, defined as:*

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

Correctness*: For all $\mathsf{m} \in \{0,1\}^n$, it holds that*

$$\Pr\left[\mathscr{C}(\mathsf{m}) = \mathsf{Prog}(\mathsf{m}) \,\middle|\, \mathsf{Prog} \leftarrow\!\!\$ \mathsf{ADP.Setup}(1^\lambda, \mathscr{C})\right] = 1 .$$

Security*: We say that a* ADP *scheme is* IND-ADP *secure with respect to a class of circuits $\mathcal{C}_\lambda$, if $\forall(\mathscr{C}_1, \mathscr{C}_2) \in \mathcal{C}_\lambda \times \mathcal{C}_\lambda$ such that $\forall \mathsf{m} \in \{0,1\}^\lambda, \mathscr{C}_1(\mathsf{m}) = \mathscr{C}_2(\mathsf{m})$ it holds that:*

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

is negligible. If the advantage of the adversary is exactly 0*, we state the scheme achieves perfect security.*

The security definition above makes clear the link between the security of an iO obfuscator [5] and indistinguishability for ADPs [9].

2.4 Direct ADPs from Randomized Encodings

Randomized Encodings via Binary Decision Diagrams. In this part, we remind the instantiation of randomized encodings(see [24] for a definition) from BDDs. A binary decision diagram can be thought as method to sequentially evaluate a function. Each input bit is used to elect a specific branch of a circuit computing the function is followed until a terminal node – 0 or 1 – is reached (we assume that we only work with single bit output functions). We stress that any function $f : \{0,1\}^n \to \{0,1\}$ in NC^1 admits a polynomial size binary decision diagram representation. In consequence, one can imagine any function $f' : \{0,1\}^n \to \{0,1\}^{n'}$ as a concatenation of n' binary decision diagrams, each outputting a single bit. Barrington shows in [6] that the shorter the depth of the circuit representation of f, the shorter will be the length of the corresponding binary decision diagram. [33] provides a clear overview on the transformation between the transition diagram of a Turing machine and a binary decision diagram.

Here we take $\mathbf{G}_{\mathsf{m}}$ to act as the adjacency matrix for the binary decision diagram of some $f : \{0,1\}^{|\mathsf{m}|} \to \{0,1\}$. For technical reasons, the entries in the main diagonal will be set to -1s; it is easy to observe that every row have at most one extra 1 apart from the -1 appearing on the main diagonal. $\overline{\mathbf{G}}_{\mathsf{m}}$ will stand for the matrix obtained by removing the last row of and the first column $\mathbf{G}_{\mathsf{m}}$. Ishai [23] proves that $f(\mathsf{m}) = \mathbf{det}(\overline{\mathbf{G}}_{\mathsf{m}})$. Moreover, matrices $\mathbf{R}_l$ and $\mathbf{R}_r$ of a designated form exist, such that the following holds:

$$\mathbf{R}_l \cdot \overline{\mathbf{G}}_{\mathsf{m}} \cdot \mathbf{R}_r = \left(\begin{array}{c|c} \vec{\mathbf{0}} & f(\mathsf{m}) \\ \hline -\mathbf{I} & \vec{\mathbf{0}} \end{array}\right) = \begin{pmatrix} 0 & 0 & \dots & 0 & f(\mathsf{m}) \\ -1 & 0 & \dots & 0 & 0 \\ 0 & -1 & \dots & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \dots & -1 & 0 \end{pmatrix} = \qquad (12)$$
$$= \overline{\mathbf{G}}_{f(\mathsf{m})} \in \mathbb{F}_p^{m \times m}$$

The main reason to make use of such a representation of $f(\mathsf{m})$, as a product of two matrices $\mathbf{R}_l$ and $\mathbf{R}_r$ is rooted in the simulation security of the randomized encoding. More specifically, the value $f(\mathsf{m})$ is given to the simulator; the simulator, on the other hand, knowing the value of $f(\mathsf{m})$ can simulate a product of (1) either full-ranked matrices or (2) of rank $m-1$ matrices. Thus, such a representation provides a natural randomized encoding. During the decoding phase,

[9] Trivially, a secure $\mathsf{IND}\text{-}\mathsf{ADP}$ affine determinant program gives rise to an indistinguishability obfuscator for the specific class of circuits.

one has to compute the determinant of $\mathbf{R}_l \cdot \overline{\mathbf{G}}_{\mathsf{m}} \cdot \mathbf{R}_r$, which allows to recover the value of $f(\mathsf{m})$. This is possible given that $\mathbf{R}_l, \mathbf{R}_r$ have determinant 1.

The matrices $\mathbf{R}_r, \mathbf{R}_l \in \mathbb{F}_p^{m \times m}$ have the following shape:

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

To generalize the previous observation, one can use different distributions for $\mathbf{R}_l, \mathbf{R}_r$. For clarity, consider $\mathbf{L}$ and $\mathbf{R}$ as two matrices sampled uniformly at random from the set of invertible matrices over $\mathbb{F}_p^{m \times m}$. One can see that both $\mathbf{L}$ and $\mathbf{R}$ are full-rank matrices, with non-negligible probability. Thus, we can write them as:

$$\mathbf{R} \leftarrow \mathbf{R}_r \cdot \mathbf{R}' \qquad \text{and} \qquad \mathbf{L} \leftarrow \mathbf{L}' \cdot \mathbf{R}_l \ .$$

Observe that:

$$\begin{aligned} \mathbf{L} \cdot \overline{\mathbf{G}}_{\mathsf{m}} \cdot \mathbf{R} &= (\mathbf{L}' \cdot \mathbf{R}_l) \cdot \overline{\mathbf{G}}_{\mathsf{m}} \cdot (\mathbf{R}_r \cdot \mathbf{R}) = \mathbf{L}' \cdot (\mathbf{R}_l \cdot \overline{\mathbf{G}}_{\mathsf{m}} \cdot \mathbf{R}_r) \cdot \mathbf{R} \\ &= \mathbf{L}' \cdot \overline{\mathbf{G}}_{f(\mathsf{m})} \cdot \mathbf{R}' \end{aligned} \tag{13}$$

Given that both $\mathbf{L}'$ and $\mathbf{R}'$ are full-rank matrices, $\det\left(\mathbf{L} \cdot \overline{\mathbf{G}}_{\mathsf{m}} \cdot \mathbf{R}\right) = \det\left(\overline{\mathbf{G}}_{\mathsf{m}}\right) = f(\mathsf{m})$. Moreover, it is easy to observe that every entry of the resulting $m \times m$ matrix

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

is in fact a sum of monomials of degree three. As described by [23], while "splitting" every entry within $\mathbf{T}_{i,j}$ in monomials, no monomial depends on more than one input bit of m. Furthermore, every monomial includes one component from each of $\mathbf{L}$ and $\mathbf{R}$. Stated differently, every monomial contains at most one entry from $\overline{\mathbf{G}}_{\mathsf{m}}$, which is m-dependent.

3 Mind the Gap: Compact FE through XiO

After seeing the attacks on the affine determinant program obfuscator, we ask the question if a set of functions can be securely ADP-obfuscated and used to get an *iO* obfuscator. We focus, in this part, on the usage of such ADPs, under the assumption they achieving *indistinguishability*, in building *compact functional encryption* (which can be used immediately to get *iO*).

The main idea we borrow from [27] is to re-use the structure of a succinct functional encryption scheme to build compact FE. The trick is to have the cFE ciphertext able to generate "on the fly" a large number of sFE ciphertexts. This is achieved by setting the compact FE ciphertext to be a family of obfuscated circuits $\overline{\mathbf{C}}$ that produces succinct FE ciphertexts:

$$\mathsf{cFE.CT} \coloneqq \left\{ \overline{\mathbf{C}}^{(\ell)}_{\mathsf{mpk},\mathsf{k},\mathsf{m},\tau}(i) \right\}_{\ell \in [\nu]} , \tag{14}$$

where mpk (and msk) are the public (secret) keys of the succinct functional encryption scheme sFE, ν is the length of the sFE ciphertext, τ is the output length of the supported class of functions f and $\mathbf{C}^{(\ell)}$ is a circuit outputting the ℓ^{th} bit of sFE as:

$$\mathbf{C}^{(\ell)}_{\mathsf{mpk},K,\mathsf{m}}(i) \coloneqq \mathsf{sFE}_\ell.\mathsf{Enc}(\mathsf{mpk}, (\mathsf{m}, i); \mathsf{PRF}(K, i)) \ . \tag{15}$$

To achieve the full functionality, during decryption, one will first run the obfuscated circuits in some point i, thus producing a succinct FE ciphertext. Then it will make use of a compact FE functional key, which is nothing else than an sFE functional key issued for the "generic" circuit computing the i^{th} bit of $f(\mathsf{m})$, with both (i, m) being taken as input by this generic circuit. In this way, one can learn all the input bits of $f(\mathsf{m})$.

Our cFE scheme is almost identical, up to the following noticeable changes: the XiO is instantiated from a set of ADPs, but the puncturable pseudorandom function pPRF that is used will evaluate always on inputs starting with 1, under a punctured key in some point $0\ldots$. The latter constraint stems from our requirements for achieving IND-ADP-security related to the topology of the circuit to be ADP-encoded. A second difference, also rooted in our ADP indistinguishability requirements (detailed in Section 4 and Appendix A), requires to use if-conditions in our real ADP, as well as in the proof.

Definition 4 (Compact Functional Encryption via IND-secure ADP). *Consider a family of functions represented by NC^1 circuits $\{\mathscr{C}_{n,\tau}\}_{(n,\tau)\in\mathbb{N}^*\times\mathbb{N}^*}$ having input length n and output length τ. Let sFE denote a succinct functional encryption scheme supporting the family $\{\mathscr{C}_{n,\tau}\}_{(n,\tau)\in\mathbb{N}^*\times\mathbb{N}^*}$ such that the ciphertext's length is $\nu \in \mathbb{N}^*$. Let $\mathsf{pPRF} : \{0,1\}^{k+n+1} \to \{0,1\}^\mu$ denote a secure puncturable pseudorandom function (Definition 10). Let $\{\mathsf{ADP}^{(\ell)}\}_{\ell\in[\nu]} : \{0,1\}^n \to \{0,1\}$ denote a set of affine determinant programs over $\mathbb{F}_2$ reaching indistinguishability (Definition 3). Define a compact functional encryption scheme cFE for the aforementioned family of NC^1 circuits as follows:*

$\underline{(\mathsf{cFE.mpk}, \mathsf{cFE.msk}) \leftarrow_\$ \mathsf{cFE.Setup}(1^\lambda)}$*:*

(1) sample a sFE pair of keys: $(\mathsf{sFE.mpk}, \mathsf{sFE.msk}) \leftarrow_\$ \mathsf{sFE.Setup}(1^\lambda)$.

(2) set: $(\mathsf{cFE.mpk}, \mathsf{cFE.msk}) \leftarrow (\mathsf{sFE.mpk}, \mathsf{sFE.msk})$.

(3) return $(\mathsf{cFE.mpk}, \mathsf{cFE.msk})$.

$\underline{\mathsf{cFE.CT} \leftarrow_\$ \mathsf{cFE.Enc}(\mathsf{cFE.mpk}, \mathsf{m})}$*:*

(1) build the circuit depicted in Figure 4.

(2) use the ADP to obtain each $\overline{\mathbf{C}}^{(\ell)}$:

$$\overline{\mathbf{C}}^{(\ell)}_{\mathsf{sFE.mpk},\mathsf{m},\mathsf{pPRF.k}_i} \leftarrow_\$ \mathsf{ADP.Setup}(1^\lambda, \mathscr{C}^{(\ell)}_{\mathsf{sFE.mpk},\mathsf{m},\mathsf{pPRF.k}_i,\tau}(\cdot)) \ .$$

(3) return $\left\{\overline{\mathbf{C}}^{(\ell)}_{\mathsf{sFE.mpk},\mathsf{m},\mathsf{pPRF.k}_i}\right\}_{\ell\in[\nu]}$.

$\underline{\mathsf{cFE.sk}_f \leftarrow_\$ \mathsf{cFE.KeyGen}(\mathsf{cFE.msk}, \mathscr{C}_f)}$*:*

(1) consider the circuit $\mathscr{C}'$ taking (m, i) and outputting the i^{th} bit of $\mathscr{C}_f$:

$$\mathscr{C}'(i, \mathsf{m}) \leftarrow \mathscr{C}^i(\mathsf{m}) \ .$$

$$\underline{\underline{\mathscr{C}^{(\ell)}_{\mathsf{sFE.mpk,m,pPRF.k},\rho}\Big(i\Big):}}$$

$$\mathsf{CT}_i^{(\ell)} \leftarrow \begin{cases} \mathsf{sFE}_\ell.\mathsf{Enc}\left(\mathsf{mpk}, (\mathsf{m}, i); \mathsf{pPRF}(\mathsf{k}, 1||i)\right)\ , & \text{if } i < \rho \\ c_\ell,\ \text{ where } c \leftarrow \mathsf{sFE}_\ell.\mathsf{Enc}\left(\mathsf{mpk}, (\mathsf{m}, i); \mathsf{pPRF}(\mathsf{k}, 1||i)\right), & \text{if } i = \rho \\ \mathsf{sFE}_\ell.\mathsf{Enc}\left(\mathsf{mpk}, (\$, i); \mathsf{pPRF}(\mathsf{k}, 1||i)\right)\ , & \text{otherwise} \end{cases}$$

$$\textbf{return } \mathsf{CT}_i^{(\ell)}$$

Fig. 4. Note that the pPRF can only be evaluated on half of its input space, as 1 is concatenated to every input. This is a **noticeable difference** to [27]. The limit ρ will be set to be the binary length of the output τ, meaning the circuit will output only the ℓ^{th} bit of the sFE encryption of m. However, during the proof, we will hybridize on ρ.

(2) issue a functional key for $\mathscr{C}'$ under sFE.msk *and return it:*

$$\mathsf{sk}_f \leftarrow_\$ \mathsf{sFE}(\mathsf{sFE.msk}, \mathscr{C}')\ .$$

<u>cFE.Dec(sk$_f$, CT)</u>*:*

(1) Run the circuit $\overline{\mathbf{C}}$ issued during encryption end recover the ν bits of sFE. *phase:*

$$\mathsf{CT}_i \leftarrow \overline{\mathbf{C}}^{(1)}(i)||\ldots||\overline{\mathbf{C}}^{(\nu)}(i)$$

(2) Run the sFE *decryption on top of* CT_i *and recover the i^{th} bit of $f(\mathsf{m})$:*

$$y_i \leftarrow \mathsf{cFE.Dec}(\mathsf{sk}_f, \mathsf{CT}_i)\ .$$

(3) Repeat the previous steps for any $i \in [\tau]$ and reconstruct $y_1||\ldots||y_\tau$.

3.1 Augmenting NC^1 BDDs for Keyed Functions

The forthcoming part (Section 3.2) makes use of ADPs to instantiate XiO. However, we will not use exactly the basic ADP described in Section 2.4, but rather an "augmented" version, described in this part. We stress that this "augmented ADP" is going to be used only by Appendix A, while in Section 3.2, ADPs can be treated as black-box objects, as we only make use of their indistinguishability property (Definition 3).

Nimvs and Augmenting BDDs. We propose a way to *augment* a binary decision diagram using a set of intermediate nodes without changing the program's functionality; our purpose in doing this is to isolate the "sensitive" variables, which we call *non-input mutable variables* or nimv. In the context of Definition 4, one should think at nimv as pPRFs keys, m, ρ or c, or any other variables on which ADPs may differ while preserving the functionality. Clearly, hiding them is a desideratum if indistinguishability-security is envisioned.

We use the terminology introduced in Section 2.4. Let us consider the binary decision diagram BDD representation of some keyed functionality $f(\mathsf{nimv}, \mathsf{m})$, instantiated as a circuit by $\mathscr{C}(\mathsf{nimv}||\mathsf{m})$; BDD's DAG representation consists of two complementary sets of nodes: one containing the nodes depending on the nimv, and the other ones depending on the input (the message m). Assume that any other node (if any) that is independent by input is included in the first set. Furthermore, we instantiate the nimv to some values (embedded in the circuit). This fact induces a binary value to all nodes depending on nimv in BDD.

An augmented binary decision diagram is simply a BDD, with an extra set of nodes that is to be added to the original graph of BDD. We explain what are these extra nodes. Let v denote a node depending on nimv, and let u denote any other node such that there is an arc $v \to u$ in the digraph representation of BDD. We augment the digraph by introducing the auxiliary node α between v and u. Observe that after this step, v is no longer directly connected to u, as the path becomes $v \to \alpha \to u$. Figure 5 presents the intuition behind this simple augmentation step.

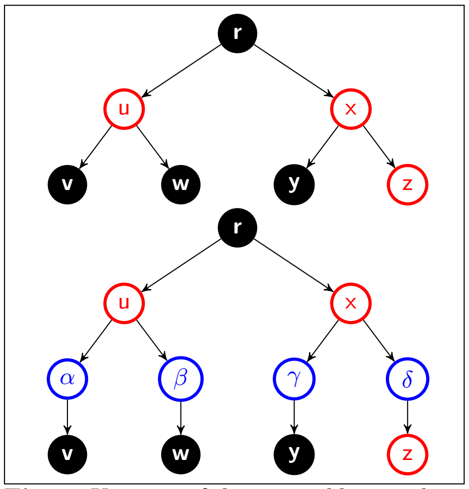

Fig. 5. Up: part of the original binary decision diagram. Down: the augmented binary decision diagram corresponding to a $\mathscr{C}$. The extra auxiliary nodes depicted in blue. Red nodes correspond to vertices settled by the bits of the nimv.

Definition 5 (Augmented binary decision diagrams for circuits in NC^1). *Let BDD be the binary decision diagram corresponding to some circuit $\mathscr{C}_{nimv} \in \mathsf{NC}^1$ that embeds nimv. Let $\mathcal{V}$ denote the set of vertices settled by nimv. For each vertex $v \in \mathcal{V}$ let u be a vertex such that there exists an arc from v to u. An augmented binary decision diagram ABDD is defined by extending the BDD graph and introducing an intermediate vertex α on the path between any node v depending on nimv and any child vertex u.*

As expected, an augmented binary decision diagram preserves correctness. We will work exclusively over $\mathbb{F}_2$, as this is the field to be used in Appendix A. It is easy to observe that over $\mathbb{F}_2$, computing the determinant is equivalent to getting the permanent [31] of a matrix. Remind that over this algebraic structure, the determinant is simply the sum of $m!$ permutations. If a path from the start node to the terminal node 1 exists, then this path and the 1s occurring on the second diagonal will force one of the sums occurring in the development of the permanent to be 1.

Luckily, the size of the augmented binary decision diagram ABDD is below $3 \times |\mathsf{BDD}|$. Given that the starting BDD has $|\mathsf{BDD}|$ nodes, each key-dependent node will add two other nodes, resulting in a very lax upper-bound of $3 \times |\mathsf{BDD}|$.

A major advantage given by ABDDs is a method to decouple the rows (or columns) in $\mathbf{R}$ (or $\mathbf{L}$) that depend on nimv, from the remaining nodes. More explicitly, when the adjacency matrix $\mathbf{G}$ of ABDD will be multiplied with $\mathbf{R}$, the lines in $\mathbf{R}$ that are triggered by the nodes depending on nimv are separated from the lines in $\mathbf{R}$ that are triggered by the input m.

In a similar way, the columns in $\mathbf{L}$ can be split in three disjoint sets, depending on either the nimv, the message or the auxiliary variables (note the *asymmetry* to the randomizing matrix $\mathbf{R}$, where we only split its rows in two disjoint sets).

In a second step, we introduce *flare* nodes, to handle node ordering in paths. The problem we tackle: on each path, all low-index nodes must depend on nimvs (or auxiliaries) and high-index nodes must depend on input nodes.

To do so, consider a node ordering, and three non-empty lists of nodes $\mathtt{L}_1$, $\mathtt{L}_2$, $\mathtt{L}_3$. $\mathtt{L}_1$ contains only nodes depending on nimvs or auxiliary nodes, $\mathtt{L}_2$ contains only nodes depending on inputs, and $\mathtt{L}_3$ similar to $\mathtt{L}_1$. By ordering, all nodes in $\mathtt{L}_1$ have a lower index than the ones in $\mathtt{L}_2$, and this transitive relations is preserved between $\mathtt{L}_2$ and $\mathtt{L}_3$. What we want is to "decouple" nodes in $\mathtt{L}_2$ from the ones in $\mathtt{L}_3$. To do so, we introduce flares, after each "terminal" node from $\mathtt{L}_2$. A "terminal" vertex $v \in \mathtt{L}_2$ is one that points to some $v' \notin \mathtt{L}_2$. We introduce $\mathit{fl}^1, \mathit{fl}^0$ depending on input bit 1 such that $v \rightarrow \mathit{fl}^0 \rightarrow 1$ and $v \rightarrow \mathit{fl}^1 \rightarrow v'$. That is, fl acts as a switch: whenever the first input is 1, fl^1 re-enables the normal flow; whenever the first input node is 0, fl^0 points to the terminal node 1.

Definition 6 (Flare Nodes). *Let* BDD *be the binary decision diagram corresponding to some circuit* $\mathscr{C}_{\mathsf{nimv}} \in \mathsf{NC}^1$ *that embeds* nimv. *Let* $\prec$ *denote a node ordering relation. Let* $\mathtt{L}_1, \mathtt{L}_3$ *denote lists of increasingly ordered vertices settled by* nimv *and auxiliary nodes. Let* $\mathtt{L}_2$ *be an ordered list of nodes that are settled by inputs. Let* $\mathtt{L}_1 \prec \mathtt{L}_2 \prec \mathtt{L}_3$ *such that the relation is applied vertex-wise. For any node* v *in* $\mathtt{L}_2$ *such that* $v \rightarrow v'$ *and* $v' \notin \mathtt{L}_2$ *introduce a node* fl^0 *such that* $v \rightarrow \mathit{fl}^0 \rightarrow 1$. *Introduce* $\mathit{fl}^1 \rightarrow v'$. *Let the node* fl^b *be triggered by the value* b *of the first input bit.*

Finally, we need the first input bit to denote a dummy node. That is, we increase the arity of the ADP by 1. Evaluating in the first input set to 0 will always result in 1, while switching the first input bit to 1 means normal evaluation.

As for the case of auxiliary nodes, it can be easily observed that injecting flares will always preserve the polynomial size of the BDD.

3.2 Security of our Compact FE Scheme from Definition 4

Our compact FE scheme follows from the one in [27]. The main significant difference consists in implementing the succinct FE encryption procedure through affine determinant programs for augmented BDDs, instead of using the full power of an exponentially-efficient indistinguishability obfuscator (XiO) for $\mathsf{P}^{\mathsf{log}}/\mathsf{poly}$[10].

cFE Security from IND-Secure ADPs. The proof is structured similarly to the one in [27], up to the variation in the black-box usage of an (augmented) ADP. A second notable difference concerns the usage of a punctured PRF key in the real construction: mind the fact that our circuit evaluates the pPRF exclusively in inputs having the first bit set to 1, while the punctured key is punctured under a point having the first bit set to 0. Hence, the pPRF evaluation is always possible.

[10] An astute reader may notice that, in fact, an iO for NC^1 would suffice here, as we assume the existence of pPRFs in NC^1.

The reason behind embedding a punctured key in the real construction is that we can only prove indistinguishability for ADPs under *identically* structured binary decision diagrams. Put differently, it is usually the case that the *normal* and *punctured* evaluation procedures differ for existing pPRFs (e.g. [9]) in NC^1, while we want a unique $\mathsf{pPRF.Eval}$ procedure. We also stress that we consider only the selective security notion (Definition 1).

Theorem 3. *Assuming* pPRF *is a puncturable pseudorandom function,* ADP *is an* $\mathsf{IND\text{-}ADP}$*-secure augmented affine determinant program for* NC^1 *circuits and* sFE *is an* $\mathsf{s\text{-}IND\text{-}FE\text{-}CPA}$*-secure succinct functional encryption scheme for* $\{\mathscr{C}_{n,\tau}\}_{(n,\tau)\in\mathbb{N}^*\times\mathbb{N}^*}$*, we have that* cFE *as defined in Definition 4 is an* $\mathsf{s\text{-}IND\text{-}FE\text{-}CPA}$*-secure functional encryption scheme for* $\{\mathscr{C}_{n,\tau}\}_{(n,\tau)\in\mathbb{N}^*\times\mathbb{N}^*}$ *with weakly sublinear compactness.*

Proof (Theorem 3). The idea is to iterate over every possible input $i \in [\tau]$ and gradually change the output of the circuit that is ADP-encoded, from releasing $\mathsf{sFE.Enc}(\mathsf{mpk}, \mathsf{m}_0)$ to $\mathsf{sFE.Enc}(\mathsf{mpk}, \mathsf{m}_1)$. The proof follows through a hybrid argument. The transitions between two hybrids involves the usage of multiple sub-hybrid games, the transitions between these being based on the indistinguishability of ADPs, sFE and pPRF. We present the hybrids and when looking on the transitions, we detail the sub-hybrids and prove the game hops.

Throughout the proof, we assume that in every game, the $\mathsf{s\text{-}IND\text{-}FE\text{-}CPA}$ adversary against cFE has access to an oracle that will produce a single functional key, for the queried function f. Our simulator will be responsible for building such an oracle.

$\underline{\mathsf{Game}_0}$: this is the game corresponding to the setting where m_0 is encrypted as:

$$\mathsf{CT}_i \leftarrow \mathsf{sFE.Enc}\left(\mathsf{mpk}, (\mathsf{m}_0, i); \mathsf{pPRF}(K, 1||i)\right) .$$

This is because the ADPs obfuscate circuits producing the sFE ciphertext corresponding to the same message – say m_0 – over the entire set of inputs $i \in [\tau]$. This game corresponds to the setting $b = 0$ in Definition 1.

$\underline{\mathsf{Game}_{j, j\in\{0,\ldots,\tau\}}}$: in this game, we change the circuit to be obfuscated via our ADPs (we will have sub-hybrids $\forall \ell \in [\nu]$) in the following way:

$$\overline{\mathbf{C}}^{(\ell)}_{\mathsf{pPRF}.K,\mathsf{m}_0,\mathsf{m}_1,j}(i) \coloneqq \begin{cases} \mathsf{sFE}_\ell.\mathsf{Enc}\big(\mathsf{mpk}, (\mathsf{m}_0, i); \mathsf{pPRF}(\mathsf{k}, 1||i)\big) , & \text{if } i < \tau - j \\ c, \text{ where } c \leftarrow \mathsf{sFE}_\ell.\mathsf{Enc}\big(\mathsf{mpk}, (\mathsf{m}_0, i); \mathsf{pPRF}(K, 1||i)\big), & \text{if } i = \tau - j \\ \mathsf{sFE}_\ell.\mathsf{Enc}\big(\mathsf{mpk}, (\mathsf{m}_1, i); \mathsf{pPRF}(\mathsf{k}, 1||i)\big) , & \text{otherwise} \end{cases}$$

$\underline{\mathsf{Game}_\tau}$: $\mathsf{CT}_i \leftarrow \mathsf{sFE.Enc}\left(\mathsf{mpk}, (\mathsf{m}_1, i); \mathsf{pPRF}(K, 1||i)\right)$, for all $i \in [\tau]$. In this game, the ADPs encode circuits that issue the succinct ciphertext corresponding to m_1.

$\underline{\mathsf{Game}_{\mathsf{final}}}$: we switch from

$$\overline{\mathbf{C}}^{(\ell)}_{\mathsf{pPRF}.K,\mathsf{m}_0,\mathsf{m}_1,0}(i) \coloneqq \begin{cases} \mathsf{sFE}_\ell.\mathsf{Enc}\big(\mathsf{mpk}, (\mathsf{m}_0, i); \mathsf{pPRF}(\mathsf{k}, 1||i)\big) , & \text{if } i < 0 \\ c, \text{ where } c \leftarrow \mathsf{sFE}_\ell.\mathsf{Enc}\big(\mathsf{mpk}, (\mathsf{m}_0, i); \mathsf{pPRF}(K, 1||i)\big), & \text{if } i = 0 \\ \mathsf{sFE}_\ell.\mathsf{Enc}\big(\mathsf{mpk}, (\mathsf{m}_1, i); \mathsf{pPRF}(\mathsf{k}, 1||i)\big) , & \text{otherwise} \end{cases}$$

to

$$\overline{\mathbf{C}}^{(\ell)}_{\mathsf{pPRF}.K,\mathsf{m}_0,\mathsf{m}_1,\tau}(i) \coloneqq$$
$$\begin{cases} \mathsf{sFE}_\ell.\mathsf{Enc}\big(\mathsf{mpk}, (\mathsf{m}_1, i); \mathsf{pPRF}(\mathsf{k}, 1||i)\big) \ , & \text{if } i < \tau \\ c, \text{ where } c \leftarrow \mathsf{sFE}_\ell.\mathsf{Enc}\big(\mathsf{mpk}, (\mathsf{m}_1, i); \mathsf{pPRF}(K, 1||i)\big), & \text{if } i = \tau \\ \mathsf{sFE}_\ell.\mathsf{Enc}\big(\mathsf{mpk}, (\mathsf{m}_0, i); \mathsf{pPRF}(\mathsf{k}, 1||i)\big) \ , & \text{otherwise} \end{cases}$$

It is easy to see that this last setting corresponds to $b = 1$ in the s-IND-FE-CPA experiment defined in Definition 1.

Transition between Game_j *and* Game_{j+1}. We show below that hybrids Game_j and Game_{j+1} are indistinguishable, for all $j \in \{0, \ldots, \tau-1\}$. To this end, we introduce sub-hybrids. The high-level strategy is to switch the element c (corresponding to $i = \tau - j$) from m_0 to m_1.

First, we first switch the punctured k from the usual one, punctured in a random point $0||\$ \ldots \$$ to one punctured in $1||(\tau - j)$. The transition is based on the IND-ADP security of each of the ν ADPs.

Second, we rely on the pPRF security, and switch the randomness used to compute the challenge element c (corresponding to $i = j$) to some random value.

Third, we rely on the s-IND-FE-CPA security of the sFE, and obtain a genuine ciphertext corresponding to m_1, which will be used to replace c. Then we switch back using pPRF security to a crafted element, and then we switch back to a punctured key in each of the ν ADP.

– $\mathsf{Game}_j^{(0)}$: is identical to Game_j:
– $\mathsf{Game}_j^{(\ell)}$: in this game, a pPRF key k is sampled. The value R is obtained as $R \leftarrow \mathsf{pPRF.Eval}(\mathsf{k}, \tau - j)$. Then, the punctured key k^* is obtained, but it is punctured in $1||(\tau - j)$, rather than in some random point $0|| \ldots$. The circuit to be obfuscated by the ℓ^{th} ADP is set to:

$$\overline{\mathbf{C}}^{(\ell)}_{\mathsf{pPRF}.K^*,\mathsf{m}_0,\mathsf{m}_1,\tau-j}(i) \coloneqq$$
$$\begin{cases} \mathsf{sFE}_\ell.\mathsf{Enc}\big(\mathsf{mpk}, (\mathsf{m}_0, i); \mathsf{pPRF}(K^*, 1||i)\big) \ , & \text{if } i < \tau - j \\ c, \text{ where } c \leftarrow \mathsf{sFE}_\ell.\mathsf{Enc}\big(\mathsf{mpk}, (\mathsf{m}_0, i); R\big), & \text{if } i = \tau - j \\ \mathsf{sFE}_\ell.\mathsf{Enc}\big(\mathsf{mpk}, (\mathsf{m}_1, i); \mathsf{pPRF}(K^*, 1||i)\big) \ , & \text{otherwise} \end{cases}$$

To justify the transition between $\mathsf{Game}_j^{(\ell-1)}$ and $\mathsf{Game}_j^{(\ell)}$ for any $\ell \in [\nu]$, we rely on IND-ADP security. Observe the circuit in position ℓ has the same behaviour in both games: for the case $(i \neq j)$ the evaluation under the punctured key is the same, as the key is punctured in the "challenge" point $1||\tau - j$. For the case $i = \tau - j$, mind the fact that $R = \mathsf{pPRF}(K, \tau - j)$, i.e. the normal key is used and not the punctured one. Thus, the two circuits are equivalent, and IND-ADP security can be employed. Note that the reduction can sample $(\mathsf{mpk}, \mathsf{msk})$ and can issue functional keys when queried by the cFE adversary. Clearly, any PPT distinguisher between the two games can break the IND-ADP-security.

- $\mathsf{Game}_{j.1}^{(\nu)}$: in this game, all ADPs corresponding to input j make use of the punctured key.
  Now, the pPRF key k^* is embedded, as well as a value R which is sampled uniformly at random. The circuit to be obfuscated by the ℓ^{th} ADP is set identically as per the previous game.
  To justify the transition from $\mathsf{Game}_{j}^{(\nu)}$, we rely on the pPRF indistinguishability in the punctured point. The simulator get the punctured key K^* and the value R, which is generated via the pPRF (as per the previous game), of generated uniformly at random. Distinguishing the settings implies an adversary against the pPRF. For completeness, observe that our reduction can sample msk and mpk and respond with a functional key when queried by the cFE adversary.
- $\mathsf{Game}_{j.2}^{(\nu)}$: in this game, the value of c is replaced with $\mathsf{sFE.Enc}\left(\mathsf{mpk}, (\mathsf{m}_1, i); R\right)$. This happens for all ADPs.
  To justify the transition from $\mathsf{Game}_{j.1}^{(\nu)}$, we rely on the s-IND-FE-CPA security of the underlying functional encryption scheme. The reduction sends m_0 and m_1, and receives the "challenge" ciphertext c, which encrypts either m_0 or m_1. When queried by the cFE adversary, the reduction will forward the request for $\mathscr{C}'$ defined in Definition 4 to the sFE oracle, get the functional key, and forward it to the cFE adversary. Also, observe the reduction can build all the ν circuits with the bits of c embedded, and apply the obfuscator.
- $\mathsf{Game}_{j.3}^{(\nu)}$: is the inverse of $\mathsf{Game}_{j.1}^{(\nu)}$. We switch back the value of R to a value generated by the normal key of the pPRF. The game transition is identical and we defer it here.
- $\mathsf{Game}_{j.4}^{(\ell)}$: is the inverse of $\mathsf{Game}_{j}^{(\ell)}$. We switch back to using the pPRF keys punctured in $0||\$\ldots\$$, using the IND-ADP property. Let the last game in the subhybrid series be $\mathsf{Game}_{j,4}^{\nu}$.
- *Transition* $\mathsf{Game}_{j.4}^{(\ell)}$ *to* Game_{j+1}: the difference between these two games ($j \in \{0, \ldots, \tau-1\}$) is summarized below.

$$\overline{\mathbf{C}}^{(\ell)}_{\mathsf{pPRF}.K^*,\mathsf{m}_0,\mathsf{m}_1,\tau-j}(i) \coloneqq$$
$$\begin{cases} \mathsf{sFE}_\ell.\mathsf{Enc}\big(\mathsf{mpk}, (\mathsf{m}_0, i); \mathsf{pPRF}(K^*, 1||i)\big)\ , & \text{if } i < \tau - j \\ c,\ \text{where } c \leftarrow \mathsf{sFE}_\ell.\mathsf{Enc}\big(\mathsf{mpk}, (\mathsf{m}_1, i); R\big), & \text{if } i = \tau - j \\ \mathsf{sFE}_\ell.\mathsf{Enc}\big(\mathsf{mpk}, (\mathsf{m}_1, i); \mathsf{pPRF}(K^*, 1||i)\big)\ , & \text{otherwise} \end{cases}$$

and

$$\overline{\mathbf{C}}^{(\ell)}_{\mathsf{pPRF}.K^*,\mathsf{m}_0,\mathsf{m}_1,\tau-j-1}(i) \coloneqq$$
$$\begin{cases} \mathsf{sFE}_\ell.\mathsf{Enc}\big(\mathsf{mpk}, (\mathsf{m}_0, i); \mathsf{pPRF}(K^*, 1||i)\big)\ , & \text{if } i < \tau - j - 1 \\ c,\ \text{where } c \leftarrow \mathsf{sFE}_\ell.\mathsf{Enc}\big(\mathsf{mpk}, (\mathsf{m}_0, i); R\big), & \text{if } i = \tau - j - 1 \\ \mathsf{sFE}_\ell.\mathsf{Enc}\big(\mathsf{mpk}, (\mathsf{m}_1, i); \mathsf{pPRF}(K^*, 1||i)\big)\ , & \text{otherwise} \end{cases}$$

To justify the transition between the two settings, first observe they are functionally equivalent on all inputs: for $i \in \{1, \tau - j - 1\}$ and $i > \tau - j$,

this is clear. For the case $i = \tau - j$, observe that both games issue an sFE ciphertext for m_0.
Therefore, we rely on $\mathsf{IND\text{-}ADP}$-security to justify the transition.

– *Transition between* Game_τ *and* $\mathsf{Game}_{\mathit{final}}$: There is a single, final transition to be justified. To do this, observe that the two circuits will both issues ciphertexts corresponding to m_1, for all inputs $i \in \{1, \ldots, \tau\}$. We rely again, on the $\mathsf{IND\text{-}ADP}$ security, as m will be one of the nimv values. The distance between these two games is bounded by $\nu \cdot \mathbf{Adv}^{\mathsf{IND\text{-}ADP}}_{\mathcal{A},\mathsf{ADP}}(\lambda)$, where $\mathcal{A}$ represents the cFE adversary.

We apply the union bound, and conclude that the advantage of any PPT bounded adversary in winning the $\mathsf{s\text{-}IND\text{-}FE\text{-}CPA}$ game is upper bounded by:

$$\begin{aligned}\mathbf{Adv}^{\mathsf{s\text{-}IND\text{-}FE\text{-}CPA}}_{\mathcal{A},\mathsf{cFE}}(\lambda) \leq & (2\tau\nu + \tau + \nu) \cdot \mathbf{Adv}^{\mathsf{IND\text{-}ADP}}_{\mathcal{A}_1,\mathsf{ADP}}(\lambda) + \\ & \tau \cdot \mathbf{Adv}^{\mathsf{s\text{-}IND\text{-}FE\text{-}CPA}}_{\mathcal{A}_2,\mathsf{sFE}}(\lambda) + 2\tau \cdot \mathbf{Adv}^{\mathsf{puncture}}_{\mathcal{A}_3,\mathsf{pPRF}}(\lambda)\end{aligned} \tag{16}$$

Weakly sublinear compactness. The second part that needs to be proved is the ciphertext compactness. The length of our cFE is simply

$$\begin{aligned}\nu \cdot |\mathsf{ADP}| \leq \mathsf{poly}(\lambda, n) &= \mathsf{poly}(\lambda, n, \log_2(s)) \cdot |\mathsf{ADP}| = \mathsf{poly}(\lambda, n, \log_2(s)) \cdot \mathsf{poly}(\lambda, n) \\ &= \mathsf{poly}(\lambda, n) \cdot \log_2^c(s) \leq \mathsf{poly}(\lambda, n) \cdot s^{(1-\epsilon)}\end{aligned} \tag{17}$$

Note that $\frac{\log_2^c(s)}{s^{(1-\epsilon)}} = \left(\frac{\log_2(s)}{s^{(1-\epsilon)/c}}\right)^c$ and $\lim_{s\to\infty} \frac{\log_2(s)}{s^{(1-\epsilon)/c}} = 0$ for any $(1-\epsilon)/c > 0$. This completes the proof. □

4 Sufficiency Conditions for IND-Secure ADP

In this part we review the requirements that suffice for having $\mathsf{IND\text{-}ADP}$-secure affine determinant programs. We recap a series of results from [7], and complement some of the them.

It is by now clear the link between the security of affine determinant programs and that of indistinguishability obfuscators[5]. Namely, an ADP reaching $\mathsf{IND\text{-}ADP}$ security provides a direct iO. As seen in Appendix B, it is hard to obtain $\mathsf{IND\text{-}ADP}$ for all polynomial sized circuits.

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 7 (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 8 (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 1 (PS-ADP $\implies$ 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 1). 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. □

We also prove an expected result, namely that not all circuits are PS-ADP admissible.

Black-Box Separation? Ideally, we would like to show a generic separation. Namely not all equivalent circuits admit PS-ADP implementation. We cannot do this in a generic way, without further assuming several properties. For instance, one can assume the size of the ADP depends on the size of the circuit, which looks like a reasonable assumption. If this is the case, we can prove the following black box separation with respect to ADPs.

Proposition 2 (Separation). *Assume that* $|\mathsf{ADP}|$ *for some circuit $\mathscr{C}$ depends on the size of $\mathscr{C}$, namely $|\mathsf{ADP}| = g(|\mathscr{C}|)$. There exist equivalent circuits that do not admit* PS-ADP *secure implementations.*

Proof. Consider two functionally equivalent circuits $\mathscr{C}_0$ and $\mathscr{C}_1$ such that $g(|\mathscr{C}_0|) \neq g(|\mathscr{C}_1|)$. Sample some randomness term R and build the ADP corresponding to $\mathscr{C}_0$. Clearly, there is no randomness term R' that will provide the same ADP corresponding to $\mathscr{C}_1$ due to the differing dimensions of underlying matrices. Note that under our assumption, the size of and ADP is a function on the circuit to be "obfuscated", and *not* a function on its size. □

4.1 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. Still. we 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.

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

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

Condition 2: the main set of applications that are envisioned are built on multivariate functions. As mentioned in Section 3.1, the functions we deal with have inputs and nimv variables. Thus, we can always regard them as functions over "nimv||inp". Hence, the function that is modelled by some circuit can be described as:

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

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

Condition 3: the proof given in Appendix A 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 $\mathfrak{C}_{d,|nimv|+n}$ stand for a class of circuits of depth d and input length n. A condition for $\mathfrak{C}_{d,|nimv|+n}$ to admit a* PS-ADP*-secure implementation is that: for every $\mathscr{C} \in \mathfrak{C}_{d,|nimv|+n}$ implementing some f, there exists a* PPT *algorithm $\mathcal{R}$ such that:*

$$\Pr\left[f(|nimv|, \mathsf{inp}) = 1 | (|nimv|, \mathsf{inp}) \leftarrow \mathcal{R}(1^\lambda, f)\right] > \frac{1}{\mathsf{poly}(|nimv| + n)} \ . \quad (18)$$

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 $\mathcal{R}$ 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 1 (Is statistical iO *possible?).* Goldwasser and Rothblum [21] 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 $\mathcal{R}$ that can generate a pair $(\mathsf{nimv}, \mathsf{nimv}')$ such that $f(\mathsf{nimv}, X) = f(\mathsf{nimv}', X)$.

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

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

Remark 2. As for the previous condition, we stress that $\mathcal{R}$ is not required to sample the keys uniformly at random.

<u>**Condition 5:**</u> The proof provided in Appendix A asks that the the first *line* in the modified adjacency matrix $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$ of the BDD 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 $\mathfrak{C}_{d,|\mathsf{nimv}|+n}$ denote a class of circuits of depth d and input length n. A condition for $\mathfrak{C}_{d,|\mathsf{nimv}|+n}$ to admit a* PS-ADP*-secure implementation is that: for every $\mathscr{C} \in \mathfrak{C}_{d,|\mathsf{nimv}|+n}$ modelling some f, there exists and is "easy to find"* nimv *such that:*

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

<u>**Condition 6:**</u> The last condition is related to the ordering of nodes depending on nimv and inp, which means a condition on the topology of BDD. Let $\mathcal{D}$ be the set of nodes having their values settled by nimvs or auxiliary nodes. Let $\mathcal{I}$ be the set of nodes depending on inputs (including flares). Let $\mathcal{P}$ denote a path originating in some $v \in \mathcal{D}$. The condition states that if $u, v \in \mathcal{P}$ and $u \in \mathcal{D}$ and $v \in \mathcal{I}$, then $u \prec v$. In rough terms, it must be the case the nodes depending on input –i.e. nodes in $\mathcal{I}$ – should occur "below" the ones in $\mathcal{D}$, on any path, where $\mathcal{I}$ includes flare nodes.
Condition 12 *Let $\mathfrak{C}_{d,|\mathsf{nimv}|+n}$ stand for a class of circuits of depth d with input length n. A condition for $\mathfrak{C}_{d,|\mathsf{nimv}|+n}$ 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 $|\mathsf{nimv}|$, on any local path starting from some $|\mathsf{nimv}|$-dependent vertex and including an input-dependent vertex.*

Definition 9 (Admissible Class for ADPs via BDDs). *Let $\mathfrak{C}_{\lambda,d,|\mathsf{nimv}|+n}$ denote a class of circuits, such that $d \in \log(n)$. We call this class admissible if the requirements (1),(2),(3),(4),(5) and (6) stated above hold.*

4.2 Our Claim

Theorem 4 (IND-ADP encodings admissible functions). *Let $\mathfrak{C}_{d,|\mathsf{nimv}|+n}$ denote a class of circuits of depth d and input length n which is admissible according to Definition 9. Then, there exists an* ADP *that reaches* PS-ADP*-security*

with respect to every single $\mathscr{C}$ in the admissible class $\mathfrak{C}_{d,|nimv|+n}$. Let pPRF *denote a puncturable and* sFE *denote a succinct functional encryption scheme, both admitting circuits in* NC^1*. Let* $\mathscr{C}^{(\ell)}_{\mathsf{sFE.mpk},\mathsf{m},\mathsf{pPRF.k},\rho}$ *denote the circuit described in Definition 4. Let* $nimv \coloneqq (\mathsf{pPRF.k}, \mathsf{m}, \rho)$ *and* $nimv' \coloneqq (\mathsf{pPRF.k}', \mathsf{m}', \rho')$ *be two sets of non-input malleable variables. Then, the distributions of* $\mathsf{ADP.Setup}(\mathscr{C}^{(\ell)}_{nimv})$ *and* $\mathsf{ADP.Setup}(\mathscr{C}^{(\ell)}_{nimv'})$ *are identical.*
The full proof is provided in detail in Appendix A, but we sketch the main ideas in what follows.
Proof (Sketch). We provide below the main ideas behind the proof. The first thing to exploit is the fact that auxiliary nodes introduced in the augmentation step (Definition 6) are used to split the matrix $\mathbf{L}$ used to randomize $\mathbf{G}_{\mathsf{nimv}||0}$:

$$\mathbf{L} = \mathbf{L}_{\mathsf{nimv}} + \mathbf{L}_{\mathsf{inp}} + \mathbf{L}_{\mathsf{aux}} \tag{19}$$

and similarly for $\mathbf{R}$ into:

$$\mathbf{R} = \mathbf{R}_{\mathsf{nimv}} + \widetilde{\mathbf{R}_{\mathsf{nimv}}}\ . \tag{20}$$

Fortunately, it is easy to observe the following relations between the decompositions of $\mathbf{L}$ and $\mathbf{R}$:

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 $j \neq j'$: $\mathbf{L}_{\mathsf{inp}_{j'}} \cdot \left(\mathbf{G}_{\mathsf{inp}_j} \cdot \widetilde{\mathbf{R}_{\mathsf{nimv}}}\right) = 0$.

Having this representation, the next step is to rewrite the PS-ADP explicitly:

$$\begin{cases} (\mathbf{L}_{\mathsf{nimv}} + \mathbf{L}_{\mathsf{aux}} + \mathbf{L}_{\mathsf{inp}}) \cdot \overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}} \cdot \left(\mathbf{R}_{\mathsf{nimv}} + \widetilde{\mathbf{R}_{\mathsf{nimv}}}\right) = (\mathbf{L}'_{\mathsf{nimv}} + \mathbf{L}'_{\mathsf{aux}} + \mathbf{L}'_{\mathsf{inp}}) \cdot \overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}} \cdot \left(\mathbf{R}'_{\mathsf{nimv}} + \widetilde{\mathbf{R}_{\mathsf{nimv}}}'\right) \\ \quad \text{and} \\ \mathbf{L}_{\mathsf{inp}_j} \cdot \left(\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{1}} - \overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}\right) \cdot \widetilde{\mathbf{R}_{\mathsf{nimv}j}} = \mathbf{L}'_{\mathsf{inp}_j} \cdot \left(\overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{1}} - \overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}}\right) \cdot \widetilde{\mathbf{R}_{\mathsf{nimv}j}}' \end{cases}$$

After post-processing the equation above we get that:

$$\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}} \cdot \mathbf{A} + \mathbf{B} \cdot \overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}} = 0$$

where

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

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

It is easy to observe that up to this point we need to make use of all of the first 4 requirements. The last part of the proof requires the last 2 conditions.

We then consider the multiplication $\overline{\mathbf{G}}^{-1}_{\mathsf{nimv}||\vec{0}} \cdot \mathbf{S}$. The trick we employ is to force an invariant to "go up" and "go left" in a set of successive steps, in order to prove that line j of $\overline{\mathbf{G}}^{-1}_{\mathsf{nimv}||\vec{0}}$ multiplied with $\mathbf{S}$ is 0. We need the above condition to realize $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}} \cdot \mathbf{A} + \mathbf{B} \cdot \overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}} = 0$. Our invariant "traverses" up to the upper-left corner of $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$ and uses the topology of the matrix to derive a contradiction: we will assume that this product can be zero.

In more detail, 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 get 1 – we follow the arcs "upwards" and the *flares* will prevent us for going up and help to derive a contradiction. A bit more detailed if the inner product between line j and column $c(\neq j)$ of $\mathbf{S}$ is different by zero, we consider the product between line j and column c 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 BDD arcs in reverse order, up to the point where 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. As 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 BDD.

The final parts of the poof (Appendices E and F) are related to the BDD representation of functions as g_k (Appendix D). Again, we want that locally, the nodes depending on nimvs to occur before the nodes depending on inputs inp.

To this end we turn to BDD representations corresponding to function g_k, which composes a puncturable pPRF and Binomial/Gaussian noise extractor. The gist is to see that we can obtain an augumented representation in polynomial time by introducing flares depending on the first input bit. Such nodes act as switches and provide a nice alternative to node reordering [11]. To introduce such "dummy nodes" in the local BDD corresponding to g, we need to observe that we have to "interrupt" an entire state, as opposed to the previous case. While this is always possible, we may end up with a new binary decision diagram having $O(v^2)$ nodes, given the original one had v vertices. Still, ignoring the computational costs, this hammered approach is enough for the problem at hand.

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.

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

3. Shweta Agrawal and Ishaan Preet Singh. Reusable garbled deterministic finite automata from learning with errors. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *ICALP 2017*, volume 80 of *LIPIcs*, pages 36:1–36:13. Schloss Dagstuhl, July 2017.
4. Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 308–326. Springer, Heidelberg, August 2015.
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, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic PRFs and their applications. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 410–428. Springer, Heidelberg, August 2013.
11. 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.
12. 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.
13. 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.
14. 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.
15. Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 476–493. Springer, Heidelberg, August 2013.

16. 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.
17. Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013.
18. Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.
19. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. On the cryptographic applications of random functions. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 276–288. Springer, Heidelberg, August 1984.
20. 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.
21. 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.
22. Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 89–98. ACM Press, October / November 2006. Available as Cryptology ePrint Archive Report 2006/309.
23. 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.
24. 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.
25. Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. Cryptology ePrint Archive, Report 2020/1003, 2020. https://eprint.iacr.org/2020/1003.
26. Huijia Lin. Indistinguishability obfuscation from SXDH on 5-linear maps and locality-5 PRGs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 599–629. Springer, Heidelberg, August 2017.
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. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23. Springer, Heidelberg, May / June 2010.
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. Leslie G Valiant. The complexity of computing the permanent. *Theoretical computer science*, 8(2):189–201, 1979.
32. Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982.
33. Li Yao, Yilei Chen, and Yu Yu. Cryptanalysis of candidate obfuscators for affine determinant programs. Cryptology ePrint Archive, Report 2021/1684, 2021. https://ia.cr/2021/1684.

A Sufficient Condition for Compact FE from Standard Assumptions

In this part, we make use of the augmented binary decision diagrams described in Section 3.1. We treat each $\mathbf{T}^i$ independently, as the product of three matrices. Explicitly, this is:

$$\mathbf{T}^i_{\mathsf{nimv}||\mathsf{inp}} \leftarrow \mathbf{L}^i \cdot (\overline{\mathbf{G}}^i_{\mathsf{nimv}||\mathsf{inp}} \cdot \mathbf{R}^i) \ , \forall i \in [\nu] \ . \tag{21}$$

where ν denotes the output length of the function to be evaluated (in our case, the length of the sFE ciphertext).

Decomposition of $\overline{\mathbf{G}}^i_{\mathsf{nimv}||\mathsf{inp}}$. Each of the m lines in $\overline{\mathbf{G}}^i_{\mathsf{nimv}||\mathsf{inp}}$ depend on either: (1) the embedded key nimv, (2) the input inp, or (3) an auxiliary node a. Moreover, each line of $\overline{\mathbf{G}}^i_{\mathsf{nimv}||\mathsf{inp}}$ 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 3.1 have exactly *one*, *fixed*, *known* entry set to 1. For line j in $\overline{\mathbf{G}}^i_{\mathsf{nimv}||\mathsf{inp}}$ let x^1_j denote the node that is set iff the input is 1 and x^0_j the one set iff the input is 0[12].

Below, a pictorial perspective is presented for the rows of $\overline{\mathbf{G}}^i_{\mathsf{nimv}||\mathsf{inp}}$, and later adapted for $\mathbf{R}^i$, $\mathbf{L}^i$. 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. The same colouring scheme is applied to columns in $\mathbf{L}^i$. The point of this pictorial description is that only entries under the same colour are to be multiplied. A bit oversimplified, multiplying different coloured vectors from $\mathbf{L}^i$ and $\overline{\mathbf{G}}^i_{\mathsf{nimv}||\mathsf{inp}}$ result in nil elements. We formally describe the decomposition of matrices and the associated properties in Proposition 4.

Consider the following example:

$$\overline{\mathbf{G}}^i_{\mathsf{nimv}||\mathsf{inp}} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & x^0_1 & 0 & \dots 0 & 0 & x^1_1 \\ \mathbf{1} & 0 & x^0_2 & x^1_2 & 0 & 0 & 0 & \dots 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 & 1 & 0 & \dots 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & \dots 0 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots 0 & x^1_i & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots 0 & \mathbf{1} & 0 \end{pmatrix} \tag{22}$$

We rewrite the matrix $\overline{\mathbf{G}}^i_{\mathsf{nimv}||\mathsf{inp}}$ 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):

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

[12] We avoid writing the output bit i as a superscript in x_j to keep the notation cleaner.

$$\underbrace{\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots 0 & 0 & 0 \\ 0 & 0 & x_2^0 & x_2^1 & 0 & 0 & 0 & \dots 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots 0 & x_i^1 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots 0 & 0 & 0 \end{pmatrix}}_{\mathbf{G}^i_{\mathsf{nimv}}} + \underbrace{\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & x_1^0 & 0 & \dots 0 & 0 & x_1^1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots 0 & 0 & 0 \end{pmatrix}}_{\mathbf{G}^i_{\mathsf{inp}}} +$$

$$\underbrace{\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & \dots 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots 0 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots 0 & 0 & 0 \end{pmatrix}}_{\mathbf{G}^i_{\mathsf{aux}}}$$

This intuition is formalized through the following relation:

$$\overline{\mathbf{G}}^i_{\mathsf{nimv}||\mathsf{inp}} = \mathbf{J} + \mathbf{G}^i_{\mathsf{nimv}} + \mathbf{G}^i_{\mathsf{inp}} + \mathbf{G}^i_{\mathsf{aux}} \ . \tag{23}$$

Decomposition of $\mathbf{R}^i$. By multiplying $\overline{\mathbf{G}}^i_{\mathsf{nimv}||\mathsf{inp}}$ on the right with $\mathbf{R}^i$, we observe that each row of $\overline{\mathbf{G}}^i_{\mathsf{nimv}||\mathsf{inp}} \cdot \mathbf{R}^i$ 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 $\mathbf{G}^i_{\mathsf{nimv}}$ denoted by $\widetilde{\mathbf{R}_{\mathsf{nimv}}}^i$

and the remaining part denoted by $\mathbf{R}^i_{\mathsf{nimv}}$.

$$\mathbf{R}^i = \underbrace{\begin{pmatrix} 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ r_{3,1} & r_{3,2} & r_{3,3} & \dots & r_{3,m} \\ r_{4,1} & r_{4,2} & r_{4,3} & \dots & r_{4,m} \\ 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 0 \\ r_{m-1,1} & r_{m-1,2} & r_{m-1,3} & \dots & r_{m-1,m} \\ 0 & 0 & 0 & \dots & 0 \end{pmatrix}}_{\mathbf{R}^i_{\mathsf{nimv}}} + \underbrace{\begin{pmatrix} r_{1,1} & r_{1,2} & r_{1,3} & \dots & r_{1,m} \\ r_{2,1} & r_{2,2} & r_{2,3} & \dots & r_{2,m} \\ 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ r_{6,1} & r_{6,2} & r_{6,3} & \dots & r_{6,m} \\ r_{7,1} & r_{7,2} & r_{7,3} & \dots & r_{7,m} \\ r_{8,1} & r_{8,2} & r_{8,3} & \dots & r_{8,m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ r_{m-1,1} & r_{m-1,2} & r_{m-1,3} & \dots & r_{m-1,m} \\ 0 & 0 & 0 & \dots & 0 \\ r_{m,1} & r_{m,2} & r_{m,3} & \dots & r_{m,m} \end{pmatrix}}_{\widetilde{\mathbf{R}_{\mathsf{nimv}}}^i}$$

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

$$\mathbf{R}^i = \mathbf{R}^i_{\mathsf{nimv}} + \widetilde{\mathbf{R}_{\mathsf{nimv}}}^i \ . \tag{24}$$

Remark 3. The lines triggered by nimv in $\overline{\mathbf{G}}^i_{\mathsf{nimv}||\mathsf{inp}}$ are to be multiplied with the entries in $\mathbf{R}^i_{\mathsf{nimv}}$, while *all* the remaining entries (i.e. blue and green) in $\overline{\mathbf{G}}^i_{\mathsf{nimv}||\mathsf{inp}}$ are to be multiplied with $\widetilde{\mathbf{R}_{\mathsf{nimv}}}^i$.

Decomposition of $\mathbf{L}^i$. The final thing to notice concerns the multiplication with $\mathbf{L}^i$. Most importantly, we ensure that the elements depending on nimv within Equation (23) are multiplied with a specific set of elements in $\mathbf{L}^i$. Put differently, the columns in $\mathbf{L}^i$ can be split in three complementary sets: one used to multiply rows in $\overline{\mathbf{G}}_{\mathsf{nimv}||\mathsf{inp}}$ triggered by nimv, the other used to multiply rows depending on inp and one used to multiply entries depending on aux.

Elements of this latter set are not multiplied with any entries depending on inp. Let $\mathbf{L}^i$ be:

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

$$\underbrace{\begin{pmatrix} 0 & 0 & l_{1,3} & l_{1,4} & \dots 0 & .. & l_{1,m-1} & 0 \\ 0 & 0 & l_{2,3} & l_{2,4} & \dots 0 & \dots & l_{2,m-1} & 0 \\ 0 & 0 & l_{3,3} & l_{3,4} & \dots 0 & \dots & l_{3,m-1} & 0 \\ 0 & 0 & l_{4,3} & l_{4,4} & \dots 0 & \dots & l_{4,m-1} & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & l_{m,3} & l_{m,4} & \dots 0 & \dots & l_{m,m} & 0 \end{pmatrix}}_{\mathbf{L}^i_{\mathsf{aux}}}$$

Similarly to Equations (23) and (24), we write:

$$\mathbf{L}^i = \mathbf{L}^i_{\mathsf{nimv}} + \mathbf{L}^i_{\mathsf{inp}} + \mathbf{L}^i_{\mathsf{aux}} \tag{25}$$

A.1 Rewriting the ADP

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

Remark 4. As $\mathbf{G}^i_{\mathsf{nimv}}$ and $\widetilde{\mathbf{R}_{\mathsf{nimv}}}^i$ are coloured differently, their product is the zero matrix. This happens because each entry of $\overline{\mathbf{G}}^i_{\mathsf{nimv}||\vec{0}}$ settled by nimv is multiplied exclusively with entries from $\mathbf{R}^i_{\mathsf{nimv}}$. Each entry in $\mathbf{G}^i_{\mathsf{inp}}$ is multiplied exclusively with rows of $\widetilde{\mathbf{R}_{\mathsf{nimv}}}^i$. Each entry in $\mathbf{G}^i_{\mathsf{aux}}$ is multiplied exclusively with rows in $\widetilde{\mathbf{R}_{\mathsf{nimv}}}^i$. This happens because there is no direct arc among auxiliary nodes.

Using these notations, observe the following condition holds:

$$\begin{aligned} \overline{\mathbf{G}}^i_{\mathsf{nimv}||\vec{0}} \cdot \mathbf{R}^i &= \left(\mathbf{J} + \mathbf{G}^i_{\mathsf{nimv}} + \mathbf{G}^i_{\mathsf{aux}} + \sum_{j=1}^{n} \mathbf{G}^i_{\mathsf{inp}_j}\right) \cdot \left(\mathbf{R}^i_{\mathsf{nimv}} + \widetilde{\mathbf{R}_{\mathsf{nimv}}}^i\right) \\ &= \mathbf{J} \cdot \mathbf{R}^i + \mathbf{G}^i_{\mathsf{nimv}} \cdot \mathbf{R}^i_k + \mathbf{G}^i_{\mathsf{aux}} \cdot \widetilde{\mathbf{R}_{\mathsf{nimv}}}^i + \\ &+ \quad \sum_{j=1}^{n} \mathbf{G}^i_{\mathsf{inp}_j} \cdot \widetilde{\mathbf{R}_{\mathsf{nimv}}}^i \end{aligned} \tag{26}$$

In Equation (26), we denote by $\mathbf{G}^i_{\mathsf{inp}_j}$ the submatrix of $\mathbf{G}^i_{\mathsf{inp}}$ depending on inp_j. We do the same with the left matrix $\mathbf{L}^i$; we split its column in three complementary sets, denoted $\mathbf{L}^i_{\mathsf{nimv}}$, $\mathbf{L}^i_{\mathsf{aux}}$, $\mathbf{L}^i_{\mathsf{inp}}$: $\mathbf{L}^i = \mathbf{L}^i_{\mathsf{nimv}} + \mathbf{L}^i_{\mathsf{aux}} + \sum_{j=1}^{n} \mathbf{L}^i_{\mathsf{inp}_j}$.

Proposition 4 (Relations). *Let $\mathbf{L}^i, \mathbf{R}^i, \overline{\mathbf{G}}^i$ be defined as in Equation (23), Equation (24), Equation (25). The following relations hold:*

1. $\mathbf{L}^i_{\mathit{nimv}} \cdot \left(\mathbf{G}^i_{\mathit{aux}} \cdot \widetilde{\mathbf{R}_{\mathit{nimv}}}^i\right) = 0$.
2. *For all $j \in [n]$:* $\mathbf{L}^i_{\mathit{nimv}} \cdot \left(\mathbf{G}^i_{\mathsf{inp}_j} \cdot \widetilde{\mathbf{R}_{\mathit{nimv}}}^i\right) = 0$.

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

Proof (Proposition 4 - Sketch). The proof relies on the colouring properties of the matrices described in Equation (23), Equation (24), Equation (25). Namely multiplying matrices of different colours results in a zero matrix.

The previous orthogonality relations essentially state that products involving mismatched indices (e.g. aux, nimv) are nil. Using these properties we rewrite:

$$\begin{aligned}\mathbf{T}^i_{\mathsf{nimv}||\vec{0}} &= \mathbf{L}^i \cdot \overline{\mathbf{G}}^i_{\mathsf{nimv}||\vec{0}} \cdot \mathbf{R}^i \\ &= \mathbf{L}^i \cdot \mathbf{J} \cdot \mathbf{R}^i + \\ &\quad \Big(\mathbf{L}^i_{\mathsf{nimv}} + \mathbf{L}^i_{\mathsf{aux}} + \sum_{j=1}^{n} \mathbf{L}^i_{\mathsf{inp}_j}\Big) \cdot \\ &\quad \Big(\mathbf{G}^i_{\mathsf{nimv}} \cdot \mathbf{R}^i_{\mathsf{nimv}} + \mathbf{G}^i_{\mathsf{aux}} \cdot \widetilde{\mathbf{R}_{\mathsf{nimv}}}^i + \sum_{j=1}^{n} \mathbf{G}^i_{\vec{0}_j} \cdot \widetilde{\mathbf{R}_{\mathsf{nimv}}}^i\Big) \\ &= \mathbf{L}^i \cdot \mathbf{J} \cdot \mathbf{R}^i + \mathbf{L}^i_{\mathsf{nimv}} \cdot \mathbf{G}^i_{\mathsf{nimv}} \cdot \mathbf{R}^i_{\mathsf{nimv}} + \\ &\qquad \mathbf{L}^i_{\mathsf{aux}} \cdot \mathbf{G}^i_{\mathsf{aux}} \cdot \widetilde{\mathbf{R}_{\mathsf{nimv}}}^i + \\ &\qquad \sum_{j=1}^{n} \mathbf{L}^i_{\mathsf{inp}_j} \cdot \mathbf{G}^i_{\vec{0}_j} \cdot \widetilde{\mathbf{R}_{\mathsf{nimv}}}^i \end{aligned} \tag{27}$$

To build the entire ADP representation, we release the additional set:

$$\Big\{\mathbf{L}^i_{\mathsf{inp}_j} \cdot (\mathbf{G}^i_{\vec{1}_j} - \mathbf{G}^i_{\vec{0}_j}) \cdot \widetilde{\mathbf{R}_{\mathsf{nimv}}}^i\Big\}_{j\in[n], i\in[\nu]} \tag{28}$$

for each output bit i. Note that the output of $\mathscr{C}^i_{\mathsf{nimv}}(\mathsf{inp})$ can be obtained as:

$$\begin{aligned}\mathscr{C}^i_{\mathsf{nimv}}(\mathsf{inp}) &= \det(\mathbf{T}^i_{\mathsf{nimv}||\vec{0}}) \\ &= \det\Big(\mathbf{L}^i \cdot \mathbf{J} \cdot \mathbf{R}^i + \mathbf{L}^i_{\mathsf{nimv}} \cdot \mathbf{G}^i_{\mathsf{nimv}} \cdot \mathbf{R}^i_{\mathsf{nimv}} + \\ &\qquad \mathbf{L}^i_{\mathsf{aux}} \cdot \mathbf{G}^i_{\mathsf{aux}} \cdot \widetilde{\mathbf{R}_{\mathsf{nimv}}}^i + \sum_{j=1}^{n} \mathbf{L}^i_{\mathsf{inp}_j} \cdot \mathbf{G}^i_{\vec{0}_j} \cdot \widetilde{\mathbf{R}_{\mathsf{nimv}}}^i + \\ &\qquad \sum_{j=1}^{n} \mathbf{L}^i_{\mathsf{inp}_j} \cdot (\mathbf{G}^i_{\vec{1}_j} - \mathbf{G}^i_{\vec{0}_j}) \cdot \widetilde{\mathbf{R}_{\mathsf{nimv}}}^i\Big) \end{aligned} \tag{29}$$

Remark 5. Observe that the newly added set from Equation (28) is independent of nimv.

Deriving an indistinguishability proof is challenging from the following perspective: we cannot rely on indistinguishability between the newly formed system and the uniform distribution, as we compute the determinant to recover $\mathscr{C}_i$ and the "$det(\cdot)$" function is biased over $\mathbb{F}_2$ (see Appendix C.2). Furthermore, except for multivariate assumptions, no cryptographic average-case hardness assumptions seem helpful. We are only left with approaching the program from a purely algebraic point of view, and derive unconditional security.

A.2 ADPs for Admission Circuit Classes are IND-Secure

We now prove our central result that concerns puncturable pseudorandom functions (Definition 2). We show that any circuit computing a pPRF (or outputting some predefined value if a sanity-check condition fails[13]) admits an indistinguishable secure ADP via the construction descried in Section 3.3. This fills the missing gap in the proof of Theorem 4, Section 4.

In fact, we show a very strong guarantee: given two punctured keys nimv and nimv', punctured in some points X and X', and considering a circuit domain that excludes both X and X' – namely $\mathscr{C}_{\mathsf{nimv}}(\mathsf{m}) = \mathscr{C}_{\mathsf{nimv}'}(\mathsf{m}), \forall \mathsf{m} \in \mathcal{M}$ – we show that there exists randomness terms $(\mathbf{L}, \mathbf{R})$ and $(\mathbf{L}', \mathbf{R}')$ such that:

$$\mathsf{ADP}(\mathscr{C}^i_{\mathsf{nimv}}) = \mathsf{ADP}(\mathscr{C}^i_{\mathsf{nimv}'}) \ .$$

In some sense, an adversary cannot guess to whom it corresponds the ADP: to nimv of 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** cFE **Circuits).** *Let* pPRF *denote a puncturable* PRF *admitting circuits in* NC^1. *Let* $\mathscr{C}^i$ *denote the circuit described in Section 4.1. Let* nimv *and* nimv' *be two sets of non-input dependent mutable variables. Then, the distributions of* $\mathsf{ADP}^i(\mathscr{C}^i_{\mathsf{nimv}})$ *and* $\mathsf{ADP}^i(\mathscr{C}^i_{\mathsf{nimv}'})$ *are identical.*

Proof (Theorem 5). We show that for any two sets of variables nimv and nimv' we have that $\mathsf{ADP.Setup}(1^\lambda, \mathscr{C}_{\mathsf{nimv}}) = \mathsf{ADP.Setup}(1^\lambda, \mathscr{C}_{\mathsf{nimv}'})$. That is, for any admissible pair $(\mathbf{L}, \mathbf{R})$, there exists $(\mathbf{L}', \mathbf{R}')$ such that the following system holds:

$$\begin{cases} (\mathbf{L}_{\mathsf{nimv}} + \mathbf{L}_{\mathsf{aux}} + \mathbf{L}_{\mathsf{inp}}) \cdot \overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}} \cdot \left(\mathbf{R}_{\mathsf{nimv}} + \widetilde{\mathbf{R}_{\mathsf{nimv}}}\right) = \\ \qquad = \left(\mathbf{L}'_{\mathsf{nimv}} + \mathbf{L}'_{\mathsf{aux}} + \mathbf{L}'_{\mathsf{inp}}\right) \cdot \overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}} \cdot \left(\mathbf{R}'_{\mathsf{nimv}} + \widetilde{\mathbf{R}_{\mathsf{nimv}}}'\right) \\ \quad \text{and} \\ \mathbf{L}_{\mathsf{inp}_j} \cdot \left(\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{1}} - \overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}\right) \cdot \widetilde{\mathbf{R}_{\mathsf{nimv}_j}} = \\ \qquad = \mathbf{L}'_{\mathsf{inp}_j} \cdot \left(\overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{1}} - \overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}}\right) \cdot \widetilde{\mathbf{R}_{\mathsf{nimv}_j}}' \end{cases} \tag{30}$$

[13] By sanity check we mean, for example, that $\mathsf{PRG}(\mathsf{sk}) = \overline{\mathsf{sk}}$.

Step 1. In the equation above we set $\mathbf{L}_{\mathsf{inp}} = \mathbf{L}'_{\mathsf{inp}}$ and $\widetilde{\mathbf{R}_{\mathsf{nimv}}} = \widetilde{\mathbf{R}_{\mathsf{nimv}}}'$. This is the choice we make to simplify the proof, by eliminating the need to treat the second part of the system above independently. We consider the equation above, simplified, once again, in a more compact form:

$$\begin{aligned} \mathbf{L} \cdot \overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}} \cdot \mathbf{R} = \mathbf{L}' \cdot \overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}} \cdot \mathbf{R}' &\iff \\ \overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}} \cdot \left(\mathbf{R} \cdot (\mathbf{R}')^{-1}\right) = \left(\mathbf{L}^{-1} \cdot \mathbf{L}'\right) \cdot \overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}} &\iff \\ \overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}} \cdot \mathbf{A} + \mathbf{B} \cdot \overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}} = 0 & \end{aligned} \tag{31}$$

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

$$\mathbf{A} \cdot \mathbf{R}' = \mathbf{R} \ . \tag{32}$$

Step 2 - Constrains on A. Looking at Equation (32) and correlating it with the fact that $\widetilde{\mathbf{R}_{\mathsf{nimv}}}_{\mathsf{inp}} = \widetilde{\mathbf{R}_{\mathsf{nimv}}}'_{\mathsf{inp}}$, 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.

Claim (Step 2). Let $\mathbf{A}, \mathbf{R}, \mathbf{R}'$ denote matrices over $\mathbb{F}_2^{m \times m}$ such that $\mathbf{A} \cdot \mathbf{R}' = \mathbf{R}$. Let $\mathfrak{W} \subseteq [m]$ and let $\widetilde{\mathbf{R}_{\mathsf{nimv}}}$ denote a submatrix of both $\mathbf{R}$ and $\mathbf{R}'$ consisting of rows indexed by $j \in \mathfrak{W}$. Then, each row $j \in \mathfrak{W}$ of $\mathbf{A}$ has 1 in position j and all entries set to 0.

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

$$\mathbf{a}_j = (0, 0, \dots, 1, 0, 0, \dots, 0) \ ,$$

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

Similarly, we can set the relevant columns in $\mathbf{B}$ to be taken from the identity matrix.

Step 3 - Simplification. To further simplify the proof, we simply set $\mathbf{B} \leftarrow \mathbf{I}_m$. Clearly, as $\mathbf{L} = \mathbf{L}'$, we do not have an issue with setting the relevant lines to satisfy Equation (31).

Now, we are left to show in Equation (31) that 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

$$\mathbf{A} \leftarrow \mathbf{I}_m + \overline{\mathbf{G}}^{-1}_{\mathsf{nimv}||\vec{0}} \cdot \mathbf{S} \tag{33}$$

where $\mathbf{S} \leftarrow \overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}} - \overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$. Note that in Equation (33), we assume the existence of a nimv that makes $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$ invertible.

Step 4 - Why Augmentation of BDDs Matters. The trick is to 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 key:

$$\mathbf{S} \leftarrow \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots 0 & 0 & 0 \\ 0 & 0 & 1^0 & 1^1 & 0 & 0 & 0 & \dots 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 1^0 & 0 & \dots 0 & 1^1 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots 0 & 0 & 0 \end{pmatrix} \tag{34}$$

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

Proof. This follows from the usage of auxiliary nodes, where we direct each node depending on nimv to a dummy node having its *in*- and *out*- degree set to 1. Another way to look into this is to observe that key-dependent lines are colored in red will imply a linear combination over the lines of $\mathbf{R}_{\mathsf{nimv}}$, and are not combined 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 $\mathbf{A}$ described in Equation (33), after we made the substitution $\mathbf{B} \leftarrow \mathbf{I}_m$ in Equation (31). We want that line j in $\mathbf{A}$ (that is triggered by some bit j in inp, with $j \in \mathfrak{W}$) to be the same as the line j of $\mathbf{I}_m$ – i.e. to contain 1 only in position j (as formalized in the previous *Claim*).

The previous statement implies that line j in $\overline{\mathbf{G}}^{-1}_{\mathsf{nimv}||\vec{0}}$ multiplied with all columns in $\mathbf{S}$ should give 0. $\mathbf{S}$ is a very sparse matrix (as seen above), and each column contains at most a single 1. Let $\mathbf{g}_j$ denote the j^{th} row of $\overline{\mathbf{G}}^{-1}_{\mathsf{nimv}||\vec{0}}$ and let $\mathbf{h}$ stand for some column of $\mathbf{S}$. Assume there exists an index $\ell \in [m]$ such that $\mathbf{h}$ has 1 in position ℓ for some $\mathbf{h}$ a non-zero column of $\mathbf{S}$. We distinguish two cases:

Case 1. If the element in position ℓ in $\mathbf{g}_j (j \in \mathfrak{W})$ is already 0, then it is perfectly fine, as this line multiplied with column $\mathbf{h}$ of $\mathbf{S}$ will give a zero, and things work as expected.

Case 2. If the element in position ℓ in $\mathbf{g}_j$ of $\overline{\mathbf{G}}^{-1}_{\mathsf{nimv}||\vec{0}}$ is 1, then the multiplication with column of $\mathbf{S}$ seems problematic. We show below a contradiction, implying that entry ℓ in $\mathbf{g}_j$ cannot be 1, for any position ℓ that is set to 1 in any non-zero column $\mathbf{h}$ of $\mathbf{S}$.

We show a simple condition on the topology of the circuit for this thing not to happen.

If $\mathbf{g}_j$ has 1 in position ℓ, we show the very first line of $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$ is $(0, 0, \ldots, 1)$.

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 $\mathbf{g}_j$ will get multiplied with columns of $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$, and in all but one cases the result must be 0.

Why the flares matter. Assume for simplicity that $j >$ the index of the current $\mathbf{h}$[14].

When we multiply $\mathbf{g}_j$ with some column of $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$ that depends on nimv – that is the column $\mathbf{h}$ 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.

"Go" left. Now we multiply $\mathbf{g}_j$ with the column of $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$ that has 1 in position ℓ, but **due to the second diagonal**[15]. It also must be the case the product between $\mathbf{g}_j$ and this current column is 0.

"Go" up. Since the previous product is 0 it must be the case that the current column has (at least) a 1 in its upper part, and $\mathbf{g}_j$ has a 1 in its left part.

The previous two steps are repeated, maintaining an invariant that enforces the left part (with respect to the current index) of $\mathbf{g}_j$ to contain a 1.
In the end, we can reach either a flare node or the leftmost column of $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$.

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 $\mathbf{g}_j$ multplied 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 reach the leftmost column of $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$, which has its entries $(0, 1, 0, \ldots)^\top$, while $\mathbf{g}_j$ 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.

[14] The other case is considered below.

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

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 binary decision diagram.

As $\mathbf{R}$ can always be sampled, and the same holds for $\mathbf{B}$, we have a satisfying assignment for Equation (30). 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 $(\mathbf{L}', \mathbf{R}')$. Thus, it follows that the two distributions are identical and therefore the implementation is $\mathsf{IND\text{-}ADP}$-secure.

B Background: Security of ADP Obfuscators for General Circuits

The work of Bartusek *et al.*[8] considers an obfuscator starting from the following affine determinant program:

$$\mathsf{ADP} \coloneqq \{\mathbf{T}_0, \mathbf{T}_1, \ldots, \mathbf{T}_n\} \ , \quad \mathbf{T}_i \in \mathbb{F}_p^{m \times m} \ \ \forall i \in \{0\} \cup [n] \tag{35}$$

From a top-down perspective, their proposed construction obfuscates a circuit by composing 4 main steps:

$$\mathsf{Rand}\left(\mathsf{BlockDiag}\left(\mathsf{Rand}\left(\mathsf{EvenNoise}\left(\mathsf{RLS}\left(\mathsf{ADP}\right)\right)\right)\right)\right) \tag{36}$$

We present each transformation, starting with the innermost to be applied on the affine determinant program.

Random Local Substitutions (RLS). Random local substitutions work by adding extra "dummy" nodes in the adjacency graph of the binary decision diagram, and thus increasing its size. The intuition behind the technique proposed in [8] is as follows: between any pair of adjacent vertices v_j and v_k add an intermediary vertex $v_{j,k}$. Now, consider the adjacency matrix of this local sub graph containing the vertices $v_j, v_k, v_{j,k}$, and observe the following matrices will provide a path between v_j and v_k, assuming one already exists:

$$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} .$$

In a similar way, [8] remarks that a set of various matrices can be made used to deal with nodes depending on input bit i or with terminal nodes.

Small Even Noise (EvenNoise). Small-noise matrices of even value, denoted $2\mathbf{E}_i$ are added to each $\mathbf{T}_i$. If the resulting ADP from the previous step is $\{\mathbf{T}_0, \mathbf{T}_1, \ldots, \mathbf{T}_n\}$, then it becomes $\{\mathbf{T}_0 + 2 \cdot \mathbf{E}_0, \mathbf{T}_1 + 2 \cdot \mathbf{E}_1, \ldots, \mathbf{T}_n + 2 \cdot \mathbf{E}_1\}$. The argument that substantiates this claim is the following:

$$\mathsf{poly}(\mathsf{inp}_1, \ldots, \mathsf{inp}_n) \equiv \mathsf{poly}(\mathsf{inp}_1 + 2e_1, \ldots, \mathsf{inp}_n + 2e_n) \mod 2$$

for any polynomial $\mathsf{poly} : \mathbb{Z}^n \to \mathbb{Z}$.

Randomization (Rand). Consider the previous ADP as a set of matrices $\{\mathbf{T}_0, \mathbf{T}_1, \dots, \mathbf{T}_n\}$ of dimension m, sample uniformly at random $\mathbf{L}$ and $\mathbf{R}$ over $\mathbb{F}_p^{m\times m}$ such that their determinant is 1. The outcome is the following ADP: $\{\mathbf{L}\cdot\mathbf{T}_0\cdot\mathbf{R}, \mathbf{L}\cdot\mathbf{T}_1\cdot\mathbf{R}, \dots, \mathbf{L}\cdot\mathbf{T}_n\cdot\mathbf{R}\}$. The underlying observation is that the determinant of the sums of matrices is not influenced by the determinants of $\mathbf{L}$ and $\mathbf{R}$ (occurring as common factors in sums).

Block-Diagonal Matrices (BlockDiag). Sample two sets of $n+1$ matrices having determinant 1 and denote them $\{\mathbf{F}_i\}_{i\in\{0,\dots,n\}}$ and $\{\mathbf{H}_i\}_{i\in\{0,\dots,n\}}$. Let $\mathsf{BlockDiag}(\mathbf{M}_1, \dots, \mathbf{M}_n)$ stand for the matrix having $(\mathbf{M}_1, \dots, \mathbf{M}_n)$ on its block diagonal, the remaining elements being filled by 0. Consider the previous ADP as a set of matrices $\{\mathbf{T}_0, \mathbf{T}_1, \dots, \mathbf{T}_n\}$ of dimension m. The outcome is the following ADP: $\mathsf{BlockDiag}(\mathbf{T}_0, \mathbf{F}, \dots, \mathbf{F}_n)$,

$\{\mathsf{BlockDiag}(\mathbf{T}_i, \mathbf{O}, \dots, \mathbf{G}_i - \mathbf{F}_i, \dots, \mathbf{O}), \}_{i\in[n]}$. It is easy to observe that adding matrices in different positions will switch the corresponding $\mathbf{F}_i$ to $\mathbf{G}_i$ in block i (say we number the blocks from 0).

The Attack in [33]. Yao, Chen and Yu propose a neat attack that exploits the mechanism of injecting randomness through random local substitution in [8]: they observe that some matrices are not affected by RLS, a fact that leaks information aside from determinant. To their credit Yao, Chen and Yu[33] propose a patch of the original ADP scheme by Bartusek.

We will not discuss their patch herein, but we rather question more seriously the security of obfuscators based on ADPs. Below we provide an ancillary result for PRFs.

C Remaining Definitions

C.1 Puncturable PRFs

Pseudorandom functions (PRF) [19] are basic cryptographic primitives having their output distributions computationally indistinguishable from the uniform distribution. The security notion they must fulfil is defined in Figure 3 (right). A puncturable pseudorandom function pPRF[30] is identical in syntax to a standard PRF, except for the existence of a puncturing algorithm. A formal definition follows.

Definition 10 (Puncturable PRFs). *A puncturable pseudorandom function* pPRF *is a triple of algorithms* (pPRF.Setup, pPRF.Eval, pPRF.Puncture) *such that:*

- $K_{\mathsf{pPRF}} \leftarrow_{\$} \mathsf{pPRF.Setup}(1^\lambda)$*: samples* K_{pPRF} *uniformly at random over the key space.*
- $K^*_{\mathsf{pPRF}} \leftarrow \mathsf{pPRF.Puncture}(K_{\mathsf{pPRF}}, \mathsf{m}^*)$ *: given* K_{pPRF} *and a point* m^* *in the input space, a punctured key* K^*_{pPRF} *is obtained.*
- $Y \leftarrow \mathsf{pPRF.Eval}(K_{\mathsf{pPRF}}, \mathsf{m})$*: identical to* PRF*'s evaluation.*

*The **correctness** requirement states that for* $\mathsf{m}^* \in \mathcal{M}$*, for all* $K_{\mathsf{pPRF}} \in \mathcal{K}$ *and for all* $\mathsf{m} \neq \mathsf{m}^* \in \mathcal{M}$*, we have that:*

$$\mathsf{pPRF.Eval}(K_{\mathsf{pPRF}}, \mathsf{m}) = \mathsf{pPRF.Eval}(K^*_{\mathsf{pPRF}}, \mathsf{m}) \ ,$$

where $K^*_{\mathsf{pPRF}} \leftarrow \mathsf{pPRF.Puncture}(K_{\mathsf{pPRF}}, \mathsf{m}^*)$*.*

We also require the output distribution of the pPRF *to be computationally indistinguishable from the uniform distribution. Moreover, we require that even in the presence of the punctured key* K^*_{pPRF}*, a* PPT *adversary cannot distinguish between* $\mathsf{pPRF.Eval}(K_{\mathsf{pPRF}}, \mathsf{m}^*)$ *and* $R \leftarrow_{\$} \mathcal{R}$*.*

C.2 Full-Rank Matrix over $\mathbb{F}_2$

We remind here that the probability of a matrix $\mathbf{R}$, sampled uniformly at random over $\mathbb{F}_2^{m\times m}$ to be invertible tends to $0.288788\ldots$.

Indeed, based on the linear independence requirements, there are $2^m - 2^{i-1}$ possibilities to sample the i^{th} row if the previous rows have already been determined. Thereby, the resulting probability satisfies

$$\Pr\left[\exists \mathbf{R}^{-1} : \mathbf{R} \leftarrow_{\$} \mathbb{F}_2^{m\times m}\right] = \prod_{i=1}^{m} \left(1 - \frac{1}{2^i}\right) \ .$$

Letting m tend to infinity and using the Pentagonal number theorem finally yields the desired limit value.

C.3 Learning with Errors

The Learning With Error (LWE) search problem [29] asks for the secret vector $\mathbf{s}$ over $\mathbb{Z}_q^n$ given a set of noisy vectors of the form $\mathbf{A}^\top \cdot \mathbf{s} + \mathbf{e}$, where $\mathbf{A}$ denotes a randomly sampled matrix over $\mathbb{Z}_q^{n\times m}$, while $\mathbf{e}$ is a small error term sampled from an appropriate distribution χ. Roughly speaking, the decision version of the problem asks to distinguish between the distribution of the LWE problem as opposed to the uniform distribution.

Definition 11 (Learning with Errors). *For an integer* $q = q(\lambda) \geq 2$ *and an error distribution* $\chi = \chi(\lambda)$ *over* $\mathbb{Z}_q$*, the decision learning with errors problems is to distinguish between the following pairs of distributions:*

$$\{(\mathbf{A}, \mathbf{A}^\top \cdot \mathbf{s} + \mathbf{e})\} \quad \textit{and} \quad \{(\mathbf{A}, \mathbf{u})\}$$

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

C.4 Further Computational Hardness Assumptions

The (search) Learning With Errors (LWE) problem [29] asks for the secret vector $\mathbf{s}$ over $\mathbb{Z}_q^n$ given a polynomial-size noisy vector of the form $\mathbf{A}^\top \cdot \mathbf{s} + \mathbf{e}$, where

$\mathbf{A}$ denotes a randomly sampled matrix over $\mathbb{Z}_q^{n\times m}$, while $\mathbf{e}$ is a small error term sampled through an appropriate distribution χ defined over $\mathbb{Z}_q^m$. Roughly speaking, the decision version of the problem, asks to distinguish between the distribution of the LWE problem as opposed to the uniform distribution. See Appendix C for a complete definition. Later, Lyubashevsky *et al.* [28] proposed a version over quotient rings (of polynomial rings). Let $R = \mathbb{Z}[X]/(X^N + 1)$ for N a power of 2, while $R_q \coloneqq R/qR$ for a safe prime q satisfying $q = 1 \bmod 2N$.

Definition 12 (Ring LWE). *For $s \leftarrow_\$ R_q$, given a polynomial number of samples that are either: (1) all of the form $(a, a\cdot s+e)$ for some $a \leftarrow_\$ \mathcal{R}_q$ and $e \leftarrow_\chi R_q$ or (2) all uniformly sampled over R_q^2; the (decision) $\mathsf{RLWE}_{q,\phi,\chi}$ states that a* PPT*-bounded adversary can distinguish between the two settings only with negligible advantage.*

C.5 (Decomposable) Functional Encryption Schemes

Decomposable Functional Encryption. Up to this point, the existing proposals for building FE schemes for general circuits depend on expensive encryption algorithms. Agrawal and Singh in [3] put forward the notion of *decomposable functional encryption* wherein chunks of the plaintext are encrypted independently. Such a technique is suitable for parallelization, as well as for online/offline encryption.

C.6 Reducing the Security of ADPs to Multivariate Systems

Motivated by the findings in [33], we show a distinctive feature of plain affine determinant programs (Section 2.4) with respect to PRFs. Namely, we can reduce the problem of finding the embedded secret key to the one of solving a (non-standard) system of multivariate equations. We present first the assumption and then the reduction. Thus, instead of assessing security for all polynomial-sized circuits, we discuss a smaller, but still practically relevant family of functions.

A New Computational Hardness Assumption. The way to judge this hardness assumption is the following: think at the binary decision diagram corresponding to some PRF, which is known to provide a secure randomized encoding as long as it is evaluated in a single input (say the all-0 input). Once we enable evaluation in multiple points, one has to provide more data – $\mathbf{T}_{\Delta_j}$. The assumption claims that even in the presence of such data it is hard to solve the system: i.e. to entirely recover the unknowns elements of $\mathbf{L}, \mathbf{R}, \overline{\mathbf{G}}$ (notations in Section 2.4).

Definition 13 (MV Assumption). *Let $\mathsf{PRF} : \{0,1\}^k \times \{0,1\}^{|\mathsf{inp}|} \rightarrow \{0,1\}$ denote a secure pseudorandom function in NC^1. Let $\mathscr{C}_{\mathsf{k}}(\cdot)$ denote the circuit representation with respect to PRF. Let $\overline{\mathbf{G}}_{\mathsf{k}||\mathsf{inp}}$ be obtained by removing the first column and the last row from the binary decision diagram corresponding to $\mathscr{C}_{\mathsf{k}}(\cdot)$.*

Let $\mathbf{L}$ *and* $\mathbf{R}$ *denote two randomly sampled invertible matrices over* $\mathbb{F}_2^{m\times m}$*. Then, the following multivariate system:*

$$\left(\mathbf{L}\cdot\overline{\mathbf{G}}_{\mathsf{k}||\vec{0}}\cdot\mathbf{R},\quad \left\{\mathbf{L}\cdot\left(\mathbf{G}_{\mathsf{k}||\vec{1}_j}-\mathbf{G}_{\mathsf{k}||\vec{0}_j}\right)\cdot\mathbf{R}\right)\right\}_{j\in[\mathsf{inp}]}\right)$$

cannot be solved in polynomial time by any computationally bounded PPT *adversary* $\mathcal{A}$.

Remark 6. Unfortunately we cannot rely on indistinguishability between the newly formed system and the uniform distribution, as we compute the determinant to recover the output of $\mathscr{C}_{\mathsf{k}}$ and the "$det(\cdot)$" function is biased over $\mathbb{F}_2$ (as detailed in Appendix C.2).

We provide a reduction to the computational (i.e. search) assumption we introduced in Definition 13. For simplicity we will call the assumption MV, standing for multivariate system of equations.

Theorem 6 (MV $\Rightarrow$ Hard to Extract k**).** *The advantage of any* PPT *bounded adversary* $\mathcal{A}$ *in recovering* k *is bounded by:*

$$\Pr\left[\mathsf{k}\leftarrow_{\$}\mathcal{A}\left(\begin{array}{c}\mathbf{L}\cdot\overline{\mathbf{G}}_{\mathsf{k}||\vec{0}}\cdot\mathbf{R},\\ \left\{\mathbf{L}\cdot\left(\mathbf{G}_{\mathsf{k}||\vec{1}_j}-\mathbf{G}_{\mathsf{k}||\vec{0}_j}\right)\cdot\mathbf{R}\right\}_{j\in[\mathsf{inp}]}\end{array}\right)\right]\leq\mathbf{Adv}_{\mathcal{R}}^{MV}(\lambda)\ .$$

Proof (Theorem 6). We will argue that extracting k is equivalent to fully solving the system of equations. One implication is trivial.

For the other case, we take the contrapositive. Suppose an adversary $\mathcal{A}$ recovers k with non-negligible advantage. Thus $\mathcal{A}$ has full knowledge on the topology of $\mathbf{G}_{\mathsf{k}||\mathsf{inp}}$, for any input of its choice. If this is the case, then, there exists an algorithm (reduction) $\mathcal{R}$ such that $\mathcal{R}$ solves entirely the system of equations. Concretely, $\mathcal{R}$ is given as input the following system[16]:

$$\left(\mathbf{L}\cdot\overline{\mathbf{G}}_{\mathsf{k}||\vec{0}}\cdot\mathbf{R},\left\{\mathbf{L}\cdot\left(\mathbf{G}_{\mathsf{k}||\vec{1}_j}-\mathbf{G}_{\mathsf{k}||\vec{0}_j}\right)\cdot\mathbf{R}\right)\right\}_{j\in[\mathsf{inp}]}\right)$$

$\mathcal{R}$ runs the adversary $\mathcal{A}$ w.r.t. this system and obtains k. Next, $\mathcal{R}$ proceeds by computing[17]:

$$\begin{aligned}\mathbf{W}&=(\mathbf{L}\cdot\overline{\mathbf{G}}_{\mathsf{k}||\vec{0}}\cdot\mathbf{R})\cdot(\mathbf{L}\cdot\overline{\mathbf{G}}_{\mathsf{k}||\vec{1}}\cdot\mathbf{R})^{-1}\\&=(\mathbf{L}\cdot\overline{\mathbf{G}}_{\mathsf{k}||\vec{0}}\cdot\mathbf{R})\cdot(\mathbf{R}^{-1}\cdot\overline{\mathbf{G}}_{\mathsf{k}||\vec{1}}^{-1}\cdot\mathbf{L}^{-1})\\&=\mathbf{L}\cdot\overline{\mathbf{G}}_{\mathsf{k}||\vec{0}}\cdot\overline{\mathbf{G}}_{\mathsf{k}||\vec{1}}^{-1}\cdot\mathbf{L}^{-1}\ .\end{aligned}\tag{37}$$

[16] For the sake of clarity, we will drop for the rest of this section the superscript i from our matrices.

[17] We assume that $\overline{\mathbf{G}}_{\mathsf{k}||\vec{0}}$ and $\overline{\mathbf{G}}_{\mathsf{k}||\vec{1}}^{-1}$ are invertible; otherwise two other points can be considered instead of $\vec{0},\vec{1}$.

Then, since

$$\mathbf{L} \cdot \overline{\mathbf{G}}_{\mathsf{k}||\vec{0}} \cdot \overline{\mathbf{G}}^{-1}_{\mathsf{k}||\vec{1}} = \mathbf{W} \cdot \mathbf{L} \iff \mathbf{W} \cdot \mathbf{L} + \mathbf{L} \cdot \overline{\mathbf{G}}_{\mathsf{k}||\vec{0}} \cdot \overline{\mathbf{G}}^{-1}_{\mathsf{k}||\vec{1}} = 0 \;, \tag{38}$$

$\mathcal{R}$ can determine $\mathbf{L}$ by solving this Sylvester equation over $\mathbb{F}_2$.

In more details, assuming that $\chi_{\mathbf{W}}$ and $\chi_{\mathbf{G}}$ are the characteristic polynomials of $\mathbf{W}$ and $\overline{\mathbf{G}}_{\mathsf{k}||\vec{0}} \cdot \overline{\mathbf{G}}^{-1}_{\mathsf{k}||\vec{1}}$ respectively from Equation (38), $\mathcal{R}$ can set $\mathbf{L}$ as:

$$\mathbf{L} \leftarrow \begin{pmatrix} \mathbf{e}_0 \ldots \mathbf{e}_{d-1} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{f}_0 \\ \vdots \\ \mathbf{f}_{d-1} \end{pmatrix} \;, \tag{39}$$

where $\mathbf{e}_i$ denotes a (column) vector and $\mathbf{f}_i$ denotes a row vector. Because the system is built starting from a solution, we know there exists an irreducible polynomial p of degree d that is a factor of both $\chi_{\mathbf{W}}$ and $\chi_{\mathbf{G}}$. Starting from this observation, we can set $\begin{pmatrix} \mathbf{e}_0 \ldots \mathbf{e}_{d-1} \end{pmatrix}$ and $\begin{pmatrix} \mathbf{f}_0 \\ \vdots \\ \mathbf{f}_{d-1} \end{pmatrix}$ to satisfy:

$$\mathbf{W} \cdot \begin{pmatrix} \mathbf{e}_0 \ldots \mathbf{e}_{d-1} \end{pmatrix} = \begin{pmatrix} \mathbf{e}_0 \ldots \mathbf{e}_{d-1} \end{pmatrix} \cdot \mathbf{C} \tag{40}$$

and

$$\begin{pmatrix} \mathbf{f}_0 \\ \vdots \\ \mathbf{f}_{d-1} \end{pmatrix} \cdot \overline{\mathbf{G}}_{\mathsf{k}||\vec{0}} \cdot \overline{\mathbf{G}}^{-1}_{\mathsf{k}||\vec{1}} = \mathbf{C} \cdot \begin{pmatrix} \mathbf{f}_0 \\ \vdots \\ \mathbf{f}_{d-1} \end{pmatrix} \tag{41}$$

where $\mathbf{C}$ is the Frobenius companion matrix of the factor p.

Directly $\mathcal{R}$ can set $\mathbf{e}_i$ in Equation (40) in the following manner: we first sample $\mathbf{e}_0$ in the nullspace of $p(\mathbf{W})$, and then set $\mathbf{e}_i \leftarrow \mathbf{W}^i \cdot \mathbf{e}_0$. Here by $\mathbf{W}^i$ for the i-th power of matrix $\mathbf{W}$. Similarly, we set $\mathbf{f}_i$ in Equation (41). In this way one can recover $\mathbf{L}$ from Equation (39). Getting $\mathbf{R}$ can be done identically by $\mathcal{R}$.

As we know a priori a solution to the system exists, one has to take several such Sylvester equations and intersect the set of possible candidates for $\mathbf{L}$ and $\mathbf{R}$, in order to prune the set of potential solutions. Thus, an adversary can fully solve the system by knowing k.

D Decomposable Functional Encryption

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 he data-independent part grows quadratically with 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[18], namely the one introduced by Agrawal and Rosen in [2], due to 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 general classes of functionalities, thus building up on the works of [20,1]. As a prime element of novelty, the supported class of functions are now described by arithmetic, rather than Boolean circuits[19]. Second, the size of the ciphertexts in their construction 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.

Roadmap. 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 first describe the interface of a decomposable Lin-FE scheme, and then we introduce the scheme for circuits in NC^1.

D.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 $\mathcal{R}_p^w$, where the algebraic ring $\mathcal{R}_p$ 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.

[18] From now on, when referring to [2] we mean their NC^1 construction.
[19] Any Boolean circuit can be simulated by an arithmetic circuit over GF(2).

– $\mathsf{Lin\text{-}FE.Enc}(\mathsf{mpk}, \mathbf{x})$: 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, $\mathsf{CT}_{ind} \leftarrow_\$ \mathbf{Com}(\mathsf{pk}_{ind}, R)$ is a commitment to the random coins used by the scheme. The following ciphertext is returned:

$$\big(\mathsf{CT}_1, \ldots, \mathsf{CT}_w, \mathsf{CT}_{ind}\big)$$

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

– $\mathsf{Lin\text{-}FE.}\mathbf{dec}(\mathsf{sk}_f, \mathsf{CT})$: the decryption first computes a "functional ciphertext" through a public procedure Eval_{CT}:

$$\mathsf{CT}_{\langle \mathbf{y}, \mathbf{x} \rangle} \leftarrow_\$ \mathsf{Eval}_{CT}\big((\mathsf{CT}_1, \ldots, \mathsf{CT}_w), \mathbf{y}\big)$$

The procedure returns $\text{Decode}(K_\mathbf{y}, \mathsf{CT}_{\langle \mathbf{y}, \mathbf{x} \rangle}, \mathsf{CT}_{ind})$.

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:

$$\mathsf{CT}_\mathbf{x} \leftarrow \big(\mathsf{CT}_1, \ldots, \mathsf{CT}_w, \mathsf{CT}_{ind}\big)$$

$$\mathsf{mpk} \leftarrow \big(\mathsf{pk}_1, \ldots, \mathsf{pk}_w, \mathsf{pk}_{ind}\big)$$

Additionally, one may write that:

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

D.2 Regev Encodings

First, we informally recall the simple symmetric encryption scheme presented in [13]. Here, "s" stands for a RLWE secret acting as a secret key, while a and r are the random mask and noise:

$$\begin{aligned} c_1 &\leftarrow a && \in \mathcal{R}_p \\ c_2 &\leftarrow a \cdot s + 2 \cdot r + x && \in \mathcal{R}_p \end{aligned} \tag{42}$$

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 $\mathcal{E}^i \colon \mathcal{R}_{p_{i-1}} \to \mathcal{R}_{p_i}$ such that

$$\mathcal{E}^i(x) = a^i \cdot s + p_{i-1} \cdot e^i + x \quad \in \mathcal{R}_{p_i} \ . \tag{43}$$

D.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 $x_i \in \mathcal{R}_{p_0}$ independently. The result is

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

where $\mathcal{E}^1$ is the map $\mathcal{E}^1 \colon \mathcal{R}_{p_0} \to \mathcal{R}_{p_1}$ defined in Equation (43). 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 $\mathcal{E}^1(x_i)$), and obtains for the left branch:

$$\begin{aligned} \mathcal{E}^2(P) = \mathcal{E}^2(\mathcal{E}^1(x_i)) = a^2_{1,i} \cdot s + p_1 \cdot e^2_{1,i} + \\ (a^1_{1,i} \cdot s + p_0 \cdot e^1_{1,i} + x_i) \in \mathcal{R}_{p_2} \end{aligned} \tag{44}$$

and for the right branch:

$$\begin{aligned} \mathcal{E}^2(P \cdot s) = \mathcal{E}^2(\mathcal{E}^1(x_i) \cdot s) = a^2_{2,i} \cdot s + p_1 \cdot e^2_{2,i} + \\ (a^1_{1,i} \cdot s + p_0 \cdot e^1_{1,i} + x_i) \cdot s \in \mathcal{R}_{p_2} \end{aligned} \tag{45}$$

for some $(a^2_{1,i}, a^2_{2,i}) \leftarrow_{\$} \mathcal{R}^2_{p_1}$ and noise terms $(e^2_{1,i}, e^2_{2,i}) \leftarrow_{\chi} \mathcal{R}_{p_2}$. 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 6.

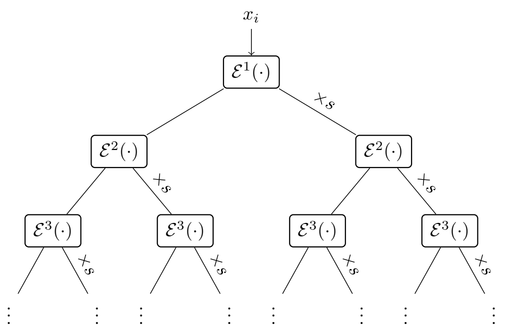

Fig. 6. The tree obtained from encoding x_i in a recursive manner.

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 7). In addition, Level 1 also contains an encoding of s, i.e., $\mathcal{E}^1(s)$, while Level i (for $2 \le i \le d$) also contains $\mathcal{E}^i(s^2)$. 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.

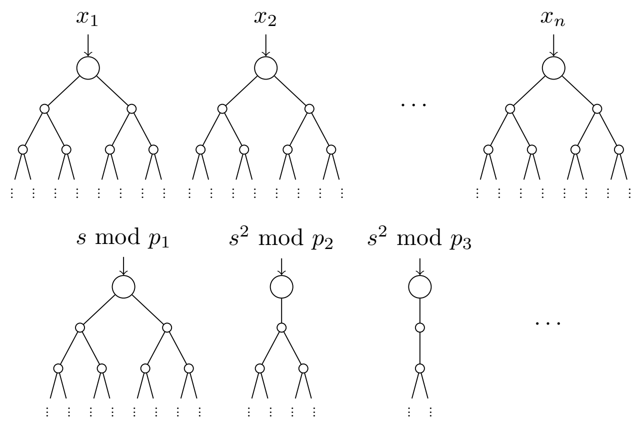

Fig. 7. A high level view of the encodings: each input x_i is recursively encrypted, the resulting ciphertext having a "tree-like" structure. Circular encryptions of s are also provided. The "additive" layers are excluded from the picture.

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 $\mathsf{Lin\text{-}FE}$. Namely, an extra component of the form:

$$\mathbf{d} \leftarrow \mathbf{w} \cdot s + p_{d-1} \cdot \eta \tag{46}$$

is provided as CT_{ind}, where $\eta \hookleftarrow_{\chi} \mathcal{R}_{p_d}$ denotes a noisy term and $\mathbf{w}$ is part of mpk. We will refer to this component from Equation (46) as a"Type-3" ciphertext.

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

$$L^{i+1} = |\mathbf{a}^{i+1}| = 2 \cdot |\mathbf{a}^i| + 1 \tag{47}$$

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 $\left\{\mathcal{E}^1(s), \mathcal{E}^2(s^2), \ldots, \mathcal{E}^d(s^2)\right\}$.

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

– The $\mathsf{Eval}_{PK}(\mathsf{mpk}, f)$ procedure uses mpk to compute PK_f. In a similar way, $\mathsf{Eval}_{CT}(\mathsf{mpk}, \mathsf{CT}, f)$ 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 H.

– **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:

$$\mathsf{CT}_{f(\mathbf{x})} \leftarrow PK_f \cdot s + \sum_{i=0}^{d-1} p_i \cdot \eta^{i+1} + f(\mathbf{x}) \ . \tag{48}$$

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

$$PK_f \cdot s + p_{d-1} \cdot \eta' \ . \tag{49}$$

Finally, $f(\mathbf{x})$ is obtained by subtracting (49) from (48) and repeatedly applying the *mod* operator to eliminate the noise: $\pmod{p_{d-1}} \ \ldots \ \pmod{p_0}$.

A more compact description is given in Appendix G.

D.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:

$$\mathsf{CT}_{f(x)} = P_f(\vec{\mathsf{CT}}^1, \ldots \vec{\mathsf{CT}}^{d-1}) + Lin_f(\vec{\mathsf{CT}}^d) \ . \tag{50}$$

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.

E The Normal and CRT Embeddings

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

$$\mathcal{R}/\mathcal{I}_1 \times \ldots \times \mathcal{R}/\mathcal{I}_n.$$

Given two elements over $\mathcal{R}_1$ – say $a = a_0 + a_1 \cdot x + \ldots a_{n-1} \cdot x^{n-1}$ and $b = b_0 + b_1 \cdot x + \ldots b_{n-1} \cdot x^{n-1}$, 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

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

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

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

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

$$\begin{aligned} a \rightarrow \Big(&(a_0 + a_1 \cdot \theta_1 + \ldots + a_{N-1} \cdot \theta_1^{N-1})/\mathcal{I}_1, \\ &\ldots \\ &(a_0 + a_1 \cdot \theta_i + \ldots + a_{N-1} \cdot \theta_i^{N-1})/\mathcal{I}_i, \\ &\quad \ldots \\ &\quad (a_0 + a_1 \cdot \theta_N + \ldots + a_{N-1} \cdot \theta_N^{N-1})/\mathcal{I}_N \Big) \end{aligned} \tag{51}$$

where $\theta_i \coloneqq \theta^{2 \cdot i - 1}$ 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 coordinates are 0. That is, $\alpha_i = (0, 0, \ldots, 0, \alpha_{i,n})$ and thus the product $\alpha_i \cdot s$ w.r.t. the CRT embedding reveals $\alpha_i \cdot s = (0 \cdot s_1, \ldots, 0 \cdot s_{n-1}, \alpha_{i,n} \cdot s_n) = (0, \ldots, 0, \alpha_{i,n} \cdot s_n)$.

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

$$a = a_0 + 0 \cdot x + \ldots 0 \cdot x^{n-1}$$

has the corresponding embedding as:

$$\overline{a} = (a_0/\mathcal{I}_1, \ldots, a_0/\mathcal{I}_i, \ldots a_0/\mathcal{I}_N)$$

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 map to itself in the coefficients of the CRT embedding.

F AR17's Encryption is IND-ADP Admissible

We detail on the track that is presented in Section 1.3. To show the IND-ADP security and from there the indistinguishability of the compact functional encryption scheme, we add constraints on BDDs. We assume the BDD has polynomial size[20] and it supports keyed bivariate functions on nimvs and inps. We also assume that the function is not constant zero (it's a ciphertext) and there are two different nimvs under which the BDDs are equivalent.

To recap, we have one BDD for every bit of the resulting ciphertext in Equation (74). We have three main classes of ciphertexts:

- **Type 1**: data-dependent ciphertexts, which are elements depicted in Figure 6.
- **Type 2**: secret-dependent ciphertexts, encrypting the randomness used to encrypt the tree.
- **Type 3**: the $\mathbf{w}$-dependent ciphertext (Equation (46)), which is used during the key derivation procedure.

We proceed to analyse the BDD implementation of each type ciphertext. Section 1.3 provides insights for the simplest case of the (succinct) data-dependent ciphertext component. Type 2 ciphertexts are almost identical, 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 1.3.

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

F.1 ADP-admissible BDDs 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:

$$\begin{aligned}\mathbf{d} \leftarrow & \Big(\sum_{j=0}^{N-1} w^d_{(j)} \cdot X^j\Big) \cdot \Big(\sum_{j=0}^{N-1} s^d_{(j)} \cdot X^j\Big) + \\ & p_{d-1} \cdot \Big(\sum_{j=0}^{N-1} \eta^d_{(j)} \cdot X^j\Big)\end{aligned} \tag{52}$$

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

Thus, in the CRT embedding, the ciphertext becomes:

$$\begin{aligned}\Big(&\overline{w_{(0)}} \cdot \overline{s_{(0)}} + p_{d-1} \cdot \overline{\eta_{(0)}}, \ldots, \\ &\quad \overline{w_{(j)}} \cdot \overline{s_{(j)}} + p_{d-1} \cdot \overline{\eta_{(j)}}, \ldots, \\ &\qquad \overline{w_{(N-1)}} \cdot \overline{s_{(N-1)}} + p_{d-1} \cdot \overline{\eta_{(N-1)}}\Big)\end{aligned} \tag{53}$$

In Equation (53), 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 binary decision diagrams producing ciphertexts with respect to the CRT embedding.

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

$$\overline{w_{(j)}} \cdot \overline{s_{(j)}} + p_{d-1} \cdot \overline{\eta_{(j)}} \ . \tag{54}$$

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

$$\overline{\eta_{(j)}} = \Big(\sum_{k=0}^{N-1} \eta_{(k)} \cdot \theta^{k\cdot(2j-1)}\Big) \mod \mathcal{I}_j \ . \tag{55}$$

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 (55) can be written explicitly as:

$$\overline{\eta_{(j)}} = \Big(\sum_{k=0}^{N-1} \big(\sum_{u=0}^{2\sigma^2-1} (b_{u,k} - b'_{u,k})\big) \cdot \theta^{k\cdot(2j-1)} \Big) \mod \mathcal{I}_j \ . \tag{56}$$

To further simplify notation, we rewrite Equation (54) as:

$$g(\mathsf{inp}) + \sum_{k=0}^{N-1} h_k(\mathsf{inp}) \tag{57}$$

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

The "shape" of BDDs. The BDD outputting bit ℓ of Equation (54) can be represented as an "addition with carry" BDD which: first chains the addition of lowest bitwise outputs of Equation (57), which represents the carry, then based on carry continues to add the second bits (another batch of BDD) up until it outputs the desired bit ℓ.

If one has to visualize the BDD, it is like a long path which, at 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 BDDs", which are detailed below ad ignored for the moment.

At this stage, we present our first result:

Theorem 7. *$\mathfrak{C}_{d,|\mathsf{nimv}|+n}$ denote a circuit class fulfilling conditions 1 → 4 in described in Theorem 1 and $\mathscr{C}$ one circuit in this class. Let* BDD *denote $\mathscr{C}$'s binary decision diagram that uses inner* BDD*. Let each inner* BDD *have, on any path, the nodes depending on* nimvs *indexed by a lower integer than nodes depending on* inp*. Let* BDD *outputs a bit of a Type 3 ciphertext. Then, there exists* BDD′ *that is* IND-ADP *admissible with respect to $\mathscr{C}$.*

Proof (Proof Theorem 7).

The proof is depicted in Figure 8. We inject dummy nodes between any two inner BDD. A dummy node v will always depend on the first (dummy) input bit and connect each possible output state from an inner BDD to the "terminal 1" node. More specifically, we do this by modifying the matrix $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$ and adding lines corresponding to the dummy nodes. As $v \to Terminal(1)$, such lines will have one only on the second diagonal and a final 1 in the last position.

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 $\overline{\mathbf{G}}^{-1}_{\mathsf{nimv}||\vec{0}}$

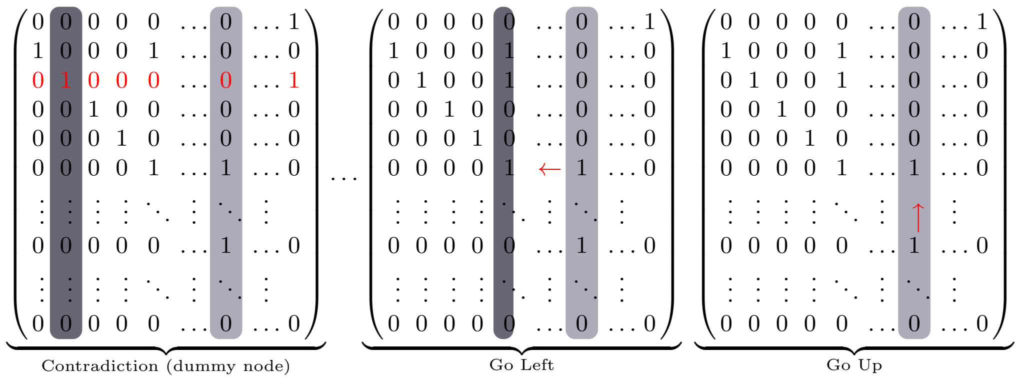

Fig. 8. Type 3's base matrix ($\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$): Introducing a dummy node and reaching contradiction. Read from right to left.

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 dummy 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 arc to follow upwards, because the dummy 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.

To re-enable the flow in the BDD (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. This completes the proof.

The previous result assumes that the inner BDD are well ordered in the sense that nodes depending on inputs occur after nodes depending on nimvs. We show here how such BDDs can be crafted.

Inner BDDs. To compute the bits in functions $g(\mathsf{inp})$ and $h_k(\mathsf{inp})$ from Equation (57), we assume the existence of another set of inner BDDs, say $\mathcal{S} := \{\mathsf{BDD}_g[0], \mathsf{BDD}_{h_1}[0], \ldots, \mathsf{BDD}_{h_k}[0], \ldots, \mathsf{BDD}_g[\ell], \mathsf{BDD}_{h_1}[\ell], \ldots, \mathsf{BDD}_{h_k}[\ell]\}$. We want that internally, they are ADP admissible. We show below how they can be transformed in such BDDs.

From a high level, the idea is very similar: what we want is to group nodes into batches: each batch should have nodes depending on nimvs in front of nodes depending on inps.

Theorem 8. *$\mathfrak{C}_{d,|nimv|+n}$ denote a circuit class fulfilling conditions 1 → 4 in Theorem 1 and let $\mathscr{C}$ be one circuit in this class. Let* BDD$''$ *denote $\mathscr{C}$ binary decision built using an inner* BDD$'$ *that have nodes depending on* nimvs *with a lower index that nodes depending on* inp *and outputs a bit of a Type 3 ciphertext.*

Then, there exists a different representation of BDD', *say* BDD *that makes* BDD'' *admissible with respect to* $\mathscr{C}$.

Proof (Proof Theorem 8).
The proof shares the injection of dummy nodes technique with the proof of Theorem 7. We use the following trick: we know BDD' constitutes a *sequential* representation of a circuit. Thus we order the nodes in the inner BDD' 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 nodes. Suppose that inp-dependent nodes $\{u_1, \ldots, \}$ are before nimv nodes $\{t_1, \ldots\}$. We will introduce a set of "dummy" 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 for the previous case, we do this by changing $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$ and inserting lines corresponding to the set of dummy nodes. As $v \to Terminal(1)$, these corresponding row has one only on the second diagonal and in its last position.

As stated before, the crux point for proving $\mathsf{IND\text{-}ADP}$ security is the following relation:

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

from Appendix A.

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 $\times$ column-j will give 0, for all $j \neq u_1$. The problematic case occurs if $j = u_1$.

If we consider BDD', since $j > u_1$, and since we follow the arcs in BDD', we can reach the case $j = u_1$, given that u_1 may be connected to t_1. However, in BDD, this is no longer the case. The newly injected dummy 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 will never occur.

As for the previous case, to re-enable the flow in the BDD (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. This completes the proof.

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

F.2 ADP-admissible BDDs for Type 1 Ciphertexts

The BDD representation of Type 1 ciphertexts using the CRT Embedding. The simple case of ciphertexts in R_{p_1} was treated in Section 1.3. 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 $a^i \cdot s^i + p_{i-1} \cdot e^i + s^{i-1} \cdot \mathsf{CT}^{i-1}$m, where CT^{i-1} denotes the lower level ciphertext. If we expand the expression above, we obtain:

$$\begin{aligned} \mathsf{CT}^i = & a^i \cdot s^i + p_{i-1} \cdot e^i + \\ & \sum_{k=1}^{i-1} \Big(a^k \cdot s^k \cdot P^k + p_{k-1} \cdot e^k \cdot P^k \Big) + \mathsf{m}_i \cdot P^1 \end{aligned} \tag{58}$$

having

$$P^k = \prod_{k \le j < i} s_j^{b_j} \ , \tag{59}$$

where b_i can be 1 or 0 depending on the chosen path in the ciphertext "tree" described in Figure 6.

We reqrite Equation (58) using the CRT embedding and obtain the following representation for the j^{th} component within the CRT embedding:

$$\begin{aligned} \mathsf{CRT}(\mathsf{CT}^i)_{(j)} \leftarrow & \overline{a^i_{(j)}} \cdot \overline{s^i_{(j)}} + p_{i-1} \cdot \overline{e^i_{(j)}} + \\ & \sum_{k=1}^{i-1} \Big(\overline{a^k}_{(j)} \cdot \overline{s^k}_{(j)} \cdot \overline{P^k}_{(j)} + \overline{p_{k-1}}_{(j)} \cdot \overline{e^k}_{(j)} \cdot \overline{P^k}_{(j)} \Big) \\ & + \mathsf{m}_i \cdot \overline{P^1}_{(j)}, \end{aligned} \tag{60}$$

where we note that

$$\overline{P^k}_v = \prod_{k \le j < i} \overline{s_j^{b_j}}_v \ , \tag{61}$$

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

$$\overline{e^i_{(j)}} = \Big(\sum_{r=0}^{N-1} e^i_{(r)} \cdot \theta^{r \cdot (2j-1)} \Big) \mod \mathcal{I}_j \ , \tag{62}$$

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

$$\overline{e^k_{(j)}} = \Big(\sum_{r=0}^{N-1} e^k_{(r)} \cdot \theta^{r \cdot (2j-1)} \Big) \mod \mathcal{I}_j \ , \tag{63}$$

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, write noise explicitly as:

$$\overline{e^i_{(j)}} = \Big(\sum_{r=0}^{N-1} \big(\sum_{u=0}^{2\sigma^2-1} (b_{u,r} - b'_{u,r}) \big) \cdot \theta^{r \cdot (2j-1)} \Big) \mod \mathcal{I}_j \ . \tag{64}$$

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

Although Equation (60) comes with a higher computational complexity, its corresponding CRT embedding can still be simplified and rewritten as:

$$m(\mathsf{inp}) + \sum_{k=0}^{N-1} g_k(\mathsf{inp}) + \sum_{k=0}^{N-1} h_k(\mathsf{inp}) \tag{65}$$

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 BDDs for Type 1 ciphertexts. As for the simpler case (type 3), the BDD outputting bit ℓ^{th} of Equation (65) 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 binary decision diagrams. But the high level idea stays the same: chains the addition of the binary decision diagrams outputting the rightmost bits (form the carry). As for type 3, based on on carry, choose a path in the tree and continue with adding the carry to the second bits (another batch of BDD) or not. This process is repeated until one obtains the desired bit of the j^{th} CRT representation of any ciphertext.

To prove the BDD that outputs one bit of a Type 1 ciphertext (its CRT representation) isIND-ADP admissible, we first proceed with the higher level approach. Essentially, we restate a previous result which assumes admissible inner BDDs.

Theorem 9. $\mathfrak{C}_{d,|\mathsf{nimv}|+n}$ *denote a circuit class fulfilling conditions 1 → 4 in Theorem 1 and* $\mathscr{C}$ *one circuit in this class. Let* BDD *denote* $\mathscr{C}$*'s binary decision diagram that uses inner* BDD*. Let each inner* BDD *have, on any path, the nodes depending on* nimvs *indexed by a lower integer than nodes depending on* inp*. Let* BDD *outputs a bit of a Type 1 ciphertext. Then, there exists* BDD$'$ *that is* IND-ADP *admissible with respect to* $\mathscr{C}$*.*

Proof (Proof Theorem 9). The proof is literally identical to the one used for Type 3 ciphertexts, except for a different using different BDDs.

The previous result assumes that the inner BDDs 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 BDDs. The shape of inner BDDs is radically changed from Type 3. We first discuss the newly added complexity ad see how this translates into the form of BDDs. The most convoluted terms are the P terms. They represent product 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. Luckily, each components can be, in fact, an inner BDD. Thus, it can be shown trivially that such nested BDDs 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 dummy nodes and ensure that on every paths containing nimv-dependent nodes, all inp-nodes occur after the nimv nodes.

Theorem 10. *$\mathfrak{C}_{d,|\mathsf{nimv}|+n}$ denote a circuit class fulfilling conditions 1 → 4 in Theorem 1 and let $\mathscr{C}$ be one circuit in this class. Let BDD'' denote $\mathscr{C}$ binary decision built using an inner BDD' that have nodes depending on nimvs with a lower index that nodes depending on inp and outputs a bit of a Type 1 ciphertext. Then, there exists a different representation of BDD', say BDD that makes BDD'' admissible with respect to $\mathscr{C}$.*

Proof (Proof Theorem 8). 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 (which is guaranteed for a puncturable PRF). Once the binary decision diagram has been transformed to use such building blocks, we can apply the same technique as in Theorem 10 that involves adding dummy 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 BDD (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 BDDs can be transformed into some in which condition (6) holds with locality, and thus the BDD is ADP admissible.

F.3 ADP-admissible BDDs for Type 2 Ciphertexts

The BDD 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 treat exclusively the case for encryption only s, and not $s \cdot s$, at any level (see also the discussion in Appendix J on getting rid of circular security at the expense of having the function in NC^0).

A Type 2 ciphertext can be written as:

$$\begin{aligned} \mathsf{CT}^i =& a^i \cdot s^i + p_{i-1} \cdot e^i + \\ & \sum_{k=1}^{i-1} \Big(a^k \cdot s^k \cdot P^k + p_{k-1} \cdot e^k \cdot P^k \Big) + \\ & s^1 \cdot P^1 \end{aligned} \tag{66}$$

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

$$P^k = \prod_{k \le j < i} s_j^{b_j} \ , \tag{67}$$

With respect to the CRT embedding, Equation (66)'s j^{th} component becomes:

$$\begin{aligned}\mathsf{CRT}(\mathsf{CT}^i)_{(j)} \leftarrow & \overline{a^i_{(j)}} \cdot \overline{s^i_{(j)}} + p_{i-1} \cdot \overline{e^i_{(j)}} + \\ & \sum_{k=1}^{i-1} \Big(\overline{a^k}_{(j)} \cdot \overline{s^k}_{(j)} \cdot \overline{P^k}_{(j)} + \overline{p_{k-1}}_{(j)} \cdot \overline{e^k}_{(j)} \cdot \overline{P^k}_{(j)} \Big) \\ & + s_i^1 \cdot \overline{P^1}_{(j)},\end{aligned} \tag{68}$$

where we note that

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

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

$$\overline{e^i_{(j)}} = \Big(\sum_{r=0}^{N-1} \big(\sum_{u=0}^{2\sigma^2-1} (b_{u,r} - b'_{u,r}) \big) \cdot \theta^{r\cdot(2j-1)} \Big) \mod \mathcal{I}_j \ . \tag{70}$$

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:

$$s(\mathsf{inp}) + \sum_{k=0}^{N-1} g_k(\mathsf{inp}) + \sum_{k=0}^{N-1} h_k(\mathsf{inp}) \tag{71}$$

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 BDDs for Type 2 ciphertexts. The BDD outputting bit ℓ^{th} of Equation (71) 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 (65). Still, the extra multiplication can be done in polynomial time in size of the modulus q, using the same approach.

To prove that a BDD outputting one bit of a Type 2 ciphertext in the CRT representation isIND-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 result.

Theorem 11. $\mathfrak{C}_{d,|\mathsf{nimv}|+n}$ *denote a circuit class fulfilling conditions 1 → 4 in Theorem 1 and* $\mathscr{C}$ *one circuit in this class. Let* BDD *denote* $\mathscr{C}$*'s binary decision diagram that uses inner* BDD*. Let each inner* BDD *have, on any path, the nodes depending on* nimvs *indexed by a lower integer than nodes depending on* inp*. Let* BDD *outputs a bit of a Type 2 ciphertext. Then, there exists* BDD′ *that is* IND-ADP *admissible with respect to* $\mathscr{C}$*.*

Proof (Proof Theorem 11). The proof is identical to the one used for Type 3 ciphertexts, except for a different using different BDDs.

Inner BDDs for Type 2 Ciphertexts. The shape of inner BDDs 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. We proceed directly with the formal statement, and observe the proofs is identical to the previous cases.

Theorem 12. $\mathfrak{C}_{d,|\mathsf{nimv}|+n}$ *denote a circuit class fulfilling conditions 1 → 4 in Theorem 1 and let* $\mathscr{C}$ *be one circuit in this class. Let* BDD′′ *denote* $\mathscr{C}$ *binary decision built using an inner* BDD′ *that have nodes depending on* nimvs *with a lower index that nodes depending on* inp *and outputs a bit of a Type 2 ciphertext. Then, there exists a different representation of* BDD′*, say* BDD *that makes* BDD′′ *admissible with respect to* $\mathscr{C}$*.*

Proof (Proof Theorem 12). The proof is virtually identical to Type 1 and 2 ciphertexts.

The puncturable PRF functionality. In all parts of out 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 $g_k(\cdot), h_k(\cdot), s(\cdot), m(\cdot)$.

F.4 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 $(0, 0, \ldots 0, 1)$, which means it always sets the base matrix's determinant to 1. Equivalently stated, the function always evaluates to 1 when evaluated on input $(0|| * \ldots *)$.

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 handle. What i the first column (corresponding to the first input bit) multiplied with some column equates

1. Then we need to follow the path in the BDD upwards. As stated, fortunately, we interrupt it.

In the event that we have the first input bit followed right after it a nimv node,w e can safely change the BDD by introducing dummy computations with inputs 2 or 2 followed again by input 1-dependent nodes to break the digraph's flow.

G 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.

$$\begin{aligned} &\textbf{for } k \leftarrow 1, \ldots, (d-1): \\ &\quad \mathbf{a}^k \leftarrow_\$ \mathcal{R}_{p_k}^{L^k} \end{aligned} \tag{72}$$

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:

$$\begin{aligned} \mathsf{mpk} &\leftarrow (\mathbf{a}^1, \ldots, \mathbf{a}^d, \mathbf{w}) \\ \mathsf{msk} &\leftarrow \mathsf{Lin\text{-}FE.msk} \end{aligned} \tag{73}$$

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

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

Invoke KeyGen to obtain K_f[21] and return it:

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

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

$$\mathsf{CT}_i^k \leftarrow_\$ \mathcal{E}^k(x_i), \ \forall k \in [d] \tag{74}$$

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

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

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

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

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

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

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

H 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. $c^k = \mathcal{E}^k(f^k(\mathbf{x}))$ and the public key or label of an encoding $\mathcal{E}^k(\cdot)$ by $\mathsf{PK}(\mathcal{E}^k(\cdot))$. 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): $\mathsf{PK}(c_\ell^2) \leftarrow u_i^1 u_j^1 \mathsf{PK}(\mathcal{E}^2(s^2)) - u_j^1 \mathsf{PK}(\mathcal{E}^2(c_i^1 s)) - u_i^1 \mathsf{PK}(\mathcal{E}^2(c_j^1 s))$.
   - 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:

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

$\mathsf{Eval}_{\mathsf{CT}}(\cup_{t\in[k]}\mathsf{CT}^t, \ell)$ 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:

1. Addition Level:
   - If $k = 1$ (base case): $c_\ell^1 \leftarrow \mathcal{E}^1(x_i) + \mathcal{E}^1(x_j)$.
   - 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): $c_\ell^2 \leftarrow c_i^1 c_j^1 \mathcal{E}^2(s^2) - u_j^1 \mathcal{E}^2(c_i^1 s) - u_i^1 \mathcal{E}^2(c_j^1 s)$.
   - 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

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

Due to space constraints, we refer the reader to [2] for complete explanations.

I The Alternative sFE Scheme from [20]

Special classes of functions, such as Boolean circuits with 1-bit of output, are functional-encryption suitable. Such FE constructions can be achieved assuming the existence of ABE (Definition 14), FHE (Definition 16) and of Garbling Schemes (Definition 15).

The construction in [20]. Goldwasser *et al.*'s proposal is to regard FE for circuits with a single-bit of output through the eyes of FHE (Figure 9). Their scheme's, *encryption procedure* proceeds as follows:

– Generates on the fly the keys for an FHE scheme – namely $(\mathsf{hpk}, \mathsf{hsk})$ – and encrypts the input m bitwise; let Ψ denote the FHE ciphertext.
– Next, Yao's garbling scheme GS is used to garble the circuit "$\mathsf{FHE.Dec}(\mathsf{hsk}, \cdot)$" and obtain two labels L_i^0, L_i^1 per input bit m_i;
– Finally, the scheme encrypts Ψ w.r.t. multiple ABE's schemes. In some sense, Ψ corresponds to an attribute: if $\mathscr{C}_f(\mathsf{m}_1, \ldots, \mathsf{m}_n) = 1$ a label L^0 is revealed. Otherwise, a label L^1 is returned (this is obtained via a two-outcome ABE).

A *functional key* for a circuit consists in an ABE key for the "FHE.Eval" circuit. The intuition is that one decrypts an ABE ciphertext with an ABE key; this corresponds to applying FHE.Eval over a FHE ciphertext. Depending on the output (which is a bit b), a label L_i^b is revealed. Once the labels are known and provided to the garbled circuit (as part of the ciphertext), the decryptor evaluates and obtains $\mathsf{FHE.Dec}(f(\Psi))$, thus yielding the expected output in a functional manner. Thus, the master keys for the FE scheme consist only of ABEs' msk and mpk. The number of ABE keys needed corresponds to the length of the FHE ciphertext.

As suggested by the authors, the result can be extended for a circuit with a constant number n of output bits almost trivially, by "replicating" the construction in [20] for each bit of output, incurring a factor n blow-up in the ciphertext length.

I.1 Attribute-Based Encryption

The *key-policy* setting ABE specifies that a decryption key must be generated for a Boolean predicate $P : \{0,1\}^\lambda \to \{0,1\}$, while a ciphertext is the encryption of a set of attributes α over $\{0,1\}^\lambda$ and of some plaintext $\mathsf{m} \in \{0,1\}^\gamma$. This is in contrast to the ciphertext policy setting. Therefore, having a decryption key enables to recover the plaintext together if $P(\alpha) = 1$, or $\perp$ otherwise.

Definition 14 (ABE [22]). *A key-policy attribute-based encryption scheme is a tuple of probabilistic polynomial time algorithms with the following behaviour:*

– $(\mathsf{mpk}, \mathsf{msk}) \leftarrow_\$ \mathsf{Setup}(1^\lambda)$*: given as input a unary representation of* λ *– the security parameter – it outputs a master public key* mpk *and a master secret key* msk.

- $\mathsf{sk}_P \leftarrow_\$ \mathsf{KGen}(\mathsf{msk}, P)$*: takes as input a master secret key as well as a policy* $P : \{0,1\}^\lambda \to \{0,1\}$*, and returns the (potentially randomized) key* sk_P *corresponding to* P*.*
- $\mathsf{CT} \leftarrow_\$ \mathsf{Enc}(\mathsf{mpk}, \alpha, \mathsf{m})$*: is a randomized procedure that encrypts the secret message* $\mathsf{m} \in \{0,1\}^\gamma$ *with respect to some attribute set* $\alpha \in \{0,1\}^\lambda$*.*
- $\mathsf{Dec}(\mathsf{sk}_P, \mathsf{CT})$*: decrypts the ciphertext* CT *using the key* sk_P *and obtains* m *if* $P(\alpha) = 1$ *or a special symbol* $\perp$*, in case the decryption procedure fails (i.e.* $P(\alpha) = 0$*).*

We say that an ABE *satisfies correctness if for all* $\mathsf{m} \in \mathcal{M}$*, for all predicates* P *and for all attributes* α *we have:*

$$\Pr\left[y = \mathsf{m} \;\middle|\; \begin{array}{l} (\mathsf{msk}, \mathsf{mpk}) \leftarrow_\$ \mathsf{ABE.Setup}(1^\lambda) \wedge \\ \mathsf{sk}_f \leftarrow_\$ \mathsf{ABE.KGen}(\mathsf{msk}, P) \wedge \\ \mathsf{CT} \leftarrow_\$ \mathsf{ABE.Enc}(\mathsf{mpk}, \mathsf{m}, \alpha) \wedge \\ y \leftarrow \mathsf{ABE.Dec}(\mathsf{CT}, \mathsf{sk}_P) \wedge P(\alpha) = 1 \end{array} \right] = 1 \ .$$

We say an attribute-based encryption is selectively secure if the advantage of any PPT *adversary* $\mathcal{A}$ *in winning the following game is negligible:* $\mathsf{m} \in \mathcal{M}$*, for all predicates* P *and for all attributes* α *we have:*

$$\Pr\left[b = b' \;\middle|\; \begin{array}{l} \alpha^* \leftarrow_\$ \mathcal{A}(1^\lambda, 1^k, 1^d) \\ (\mathsf{msk}, \mathsf{mpk}) \leftarrow_\$ \mathsf{ABE.Setup}(1^\lambda, 1^k, 1^d) \wedge \\ (\mathsf{m}_0, \mathsf{m}_1) \leftarrow_\$ \mathcal{A}(\mathsf{mpk}) \wedge \\ b \leftarrow_\$ \{0,1\} \wedge \\ \mathsf{sk}_f \leftarrow_\$ \mathcal{A}^{\mathsf{KGen}_{\mathsf{msk}}(\cdot)}(\mathsf{mpk}) \wedge \\ \mathsf{CT}^*_\alpha \leftarrow_\$ \mathsf{ABE.Enc}(\mathsf{mpk}, \mathsf{m}_b, \alpha^*) \wedge \\ b' \leftarrow \mathcal{A}^{\mathsf{KGen}_{\mathsf{msk}}(\cdot)}(\mathsf{mpk}, \mathsf{CT}^*_\alpha) \end{array} \right] = 1 \ .$$

A constraint that must be added is that $\mathcal{A}$ *does not query for keys corresponding to* $\{P : P(\alpha^*) = 1\}$*.*

I.2 Garbling Schemes.

Garbled circuits were introduced by Yao in 1982 [32] to solve the famous "Millionaires' Problem". Since then, garbled circuits became a standard building-block for many cryptographic primitives. Their definition follows.

Definition 15 (Garbling Scheme). *Let* $\{\mathscr{C}_k\}_\lambda$ *be a family of circuits taking as input* k *bits. A garbling scheme is a tuple of* PPT *algorithms* $(\mathsf{Garble}, \mathsf{Enc}, \mathsf{Eval})$ *such that:*

- $(\Gamma, \mathsf{sk}) \leftarrow_\$ \mathsf{Garble}(1^\lambda, \mathscr{C})$*: takes as input the unary representation of the security parameter and a circuit* $\mathscr{C} \in \{\mathscr{C}_k\}_\lambda$ *and outputs a garbled circuit* Γ *and a secret key* sk*.*
- $c \leftarrow_\$ \mathsf{Enc}(\mathsf{sk}, x)$*: given as input* $x \in \{0,1\}^k$ *and the secret key* sk*, the encryption procedure returns an encoding* c*.*

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

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

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

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

Fig. 9. The functional encryption scheme for boolean circuits $\mathscr{C}_f : \{0,1\}^n \rightarrow \{0,1\}$ as introduced in [20]. ℓ stands for the ciphertext's lenght of FHE, while $\mathsf{FHE.Eval}^i_f : \mathcal{K} \times \{0,1\}^{n \cdot \ell} \rightarrow \{0,1\}$ denotes the function that applies $\mathscr{C}_f$ on the encrypted input.

– $\mathscr{C}(x) \leftarrow \mathsf{Eval}(\Gamma, c)$*: the evaluation procedure receives as inputs a garbled circuit as well as an encoding of* x *returning* $\mathscr{C}(x)$*.*

We say that a garbling scheme Γ *is* ***correct*** *if for all* $\mathscr{C} : \{0,1\}^k \to \{0,1\}^\ell$ *and for all* $x \in \{0,1\}^k$ *we have that:*

$$\Pr\left[\mathscr{C}(x) = y \;\middle|\; \begin{array}{l}(\Gamma, \mathsf{sk}) \leftarrow_{\$} \mathsf{GS.Garble}(1^\lambda, C) \wedge \\ y \leftarrow \mathsf{GS.Eval}(\Gamma, \mathsf{GS.Enc}(\mathsf{sk}, x))\end{array}\right] = 1 \ .$$

Yao's Garbled Circuit [32]. An valuable way of garbled circuits is represented by the foundational proposal by Yao, who considers a family of circuits having k input wires and outputting 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 $\mathsf{Eval}(\Gamma, (L_1^{x_1}, \ldots, L_k^{x_k}))$, where x_i is the i^{th} bit of x — thus the encoding $c = (L_1^{x_1}, \ldots, L_k^{x_k})$.

I.3 Fully Homomorphic Encryption.

Fully homomorphic encryption (FHE) has been put forth with the work of Rivest, Adleman and Dertouzos and it remained an open problem until the breakthrough work of Gentry [18].

Definition 16 (FHE). *A fully-homomorphic encryption scheme, consists of a tuple algorithms* $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *such that:*

– $(\mathsf{hsk}, \mathsf{hpk}) \leftarrow_{\$} \mathsf{Setup}(1^\lambda, 1^k, 1^d)$*: a randomized algorithm returning a pair of homomorphic public and secret keys.*
– $\mathsf{CT} \leftarrow_{\$} \mathsf{Enc}(\mathsf{hpk}, \mathsf{m})$*: the randomized encryption algorithm uses* hpk *to produce a homomorphic ciphertext* CT*.*
– $\mathsf{CT}' \leftarrow \mathsf{Eval}(\mathsf{hpk}, \mathsf{CT}, f)$*: a function* f *is evaluated over the ciphertext* CT*, the resulting being another ciphertext corresponding to* $f(\mathsf{m})$*.*
– $f(\mathsf{m}) \leftarrow \mathsf{Dec}(\mathsf{hsk}, \mathsf{CT}')$*: decryption is a deterministic procedure that is given the homomorphic secret-key* hsk*, the ciphertext* CT' *and reveals* $f(\mathsf{m})$*.*

We say that a FHE *scheme is* ***perfectly correct*** *if for all* $\mathscr{C} : \{0,1\}^k \to \{0,1\}^\ell$ *of depth* d *and for all* $\mathsf{m} \in \{0,1\}^k$ *we have that the following probability is* 1*:*

$$\Pr\left[\begin{array}{l}\mathsf{FHE.Dec}(\mathsf{hsk}, \mathsf{FHE.Eval}(\mathscr{C}, \\ \quad \mathsf{FHE.Enc}(\mathsf{hpk}, \mathsf{m}))) = \mathscr{C}(\mathsf{m})\end{array} \middle| \begin{array}{l}(\mathsf{hsk}, \mathsf{hpk}) \leftarrow_{\$} \\ \quad \mathsf{FHE.Setup}(1^\lambda, 1^k, 1^d)\end{array}\right] .$$

We also require that ($\mathsf{FHE.Setup}, \mathsf{FHE.Enc}, \mathsf{FHE.Dec}$*) constitutes a semantic secure public-key encryption scheme.*

Special classes of functions, such as Boolean circuits with 1-bit of output, are functional-encryption suitable. Such FE constructions can be achieved assuming the existence of ABE (Definition 14), FHE (Definition 16) and of Garbling Schemes (Definition 15).

The construction in [20]. Goldwasser *et al.*'s proposal is to regard FE for circuits with a single-bit of output through the eyes of FHE (Figure 9). Their scheme's, *encryption procedure* proceeds as follows:

– Generates on the fly the keys for an FHE scheme – namely $(\mathsf{hpk}, \mathsf{hsk})$ – and encrypts the input m bitwise; let Ψ denote the FHE ciphertext.
– Next, Yao's garbling scheme GS is used to garble the circuit "$\mathsf{FHE.Dec}(\mathsf{hsk}, \cdot)$" and obtain two labels L_i^0, L_i^1 per input bit m_i;
– Finally, the scheme encrypts Ψ w.r.t. multiple ABE's schemes. In some sense, Ψ corresponds to an attribute: if $\mathscr{C}_f(\mathsf{m}_1, \ldots, \mathsf{m}_n) = 1$ a label L^0 is revealed. Otherwise, a label L^1 is returned (this is obtained via a two-outcome ABE).

A *functional key* for a circuit consists in an ABE key for the "$\mathsf{FHE.Eval}$" circuit. The intuition is that one decrypts an ABE ciphertext with an ABE key; this corresponds to applying $\mathsf{FHE.Eval}$ over a FHE ciphertext. Depending on the output (which is a bit b), a label L_i^b is revealed. Once the labels are known and provided to the garbled circuit (as part of the ciphertext), the decryptor evaluates and obtains $\mathsf{FHE.Dec}(f(\Psi))$, thus yielding the expected output in a functional manner. Thus, the master keys for the FE scheme consist only of ABEs' msk and mpk. The number of ABE keys needed corresponds to the length of the FHE ciphertext.

As suggested by the authors, the result can be extended for a circuit with a constant number n of output bits almost trivially, by "replicating" the construction in [20] for each bit of output, incurring a factor n blow-up in the ciphertext length.

J Conclusion and Open Problems

In this work, we explore the problem of building affine determinant programs that are indistinguishably secure. Concretely, our ADPs are built on top of binary decision diagrams that implement the encryption procedure of a single-key succinct functional encryption scheme. More specifically, we dealt with 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 BDDs, 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?

J.1 Instantiation from Standard LWE

Given that we work with AR17, can we replace RLWE with standard LWE? We certainly can.

Given that the compact FE 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 BDD 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 BDD 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.

J.2 Any Better Alternative sFE Scheme?

Finding good sFE candidates to fit into our constraints is not an easy task. A major question that can be asked is if there are any alternatives to the AR17 scheme, which seems rather convoluted. In extant literature, there exists an alternative succinct scheme [20], that is, in our opinion, even more complex to reason about.

In high level, the ciphertext structure corresponding to the sFE descried in [20] consists of (1) attribute based encryption ciphertexts, obtained on top of (2) fully homomorphic encryption and revealing (3) Yao's garbling scheme's labels. In some sense, to simulate a ciphertext, one needs to cope with an even higher amount of relatively complex cryptographic primitives. In the end, one also needs to rely to pPRFs to produce noise.

Thus, we leave this avenue for research aside, and currently focus on obtaining it from AR17.

K Previous Reviews from Asiacrypt 2022

```
Review #295A

Overall merit
-------------
3. Weak accept

Reviewer expertise
------------------
3. Knowledgeable

Paper summary
-------------
The paper focuses on achieving indistinguishability between two
    ↪ functionally equivalent programs which are similar to each
    ↪ other. To be specific, the two programs only differs in
    ↪ some hard-coded message. Since iO/XiO usually plays the
    ↪ role of hiding some hard-coded message in security proof,
    ↪ the paper argues that such indistinguishability may be
    ↪ sufficient to construct iO.

Questions/Clarifications for Authors
------------------------------------
After reading the paper, I still have some questions:
1. On page 14, why does A have full knowledge of the topology of
    ↪ G_k||input once A knows k? Do you mean that the topology is
    ↪  open information?
2. I cannot understand condition 5. You say the first line is
    ↪ (0,0,0,*,*,...,*), why do we need 3 0s? On page 39, "We
    ↪ show a simple condition", so what's the condition? The
    ↪ condition is condition 5? or the condition is the next
    ↪ sentence "the first line is (0,0,..,0,1)"? I think the
    ↪ condition 5 should be (0,0,..,0,1) as you say "On the high
    ↪ level, f(nimv,0||*****) = 1".
3. Why do we need condition 6 when go down and right?
4. Why can we always go right? What if the row has the only 1 on
    ↪ the second diagonal? For example, what if the last row is
    ↪ (0,0,...,0,1,0)?
5. Why do we want that the sums of the first and last line of R to
    ↪  be the same as the sum of the first and last line in R'?

Novelty and Conceptual Contributions
------------------------------------
```

```
I think the IND-ADP itself is the most interesting result in this
    ↪ paper. Hiding information in a program is a relatively hard
    ↪  task. One choice is to use iO, which can hide information
    ↪ "as much as possible". iO is powerful but difficult to
    ↪ construct. Therefore, some papers tried to obfuscate some
    ↪ simple functionalities such as the point function. This
    ↪ work walks a different path, it tries to protect part of
    ↪ the hard-coded information instead of "as much as possible
    ↪ ".

Technical Details
-----------------
The idea that we can substitute iO/XiO with indistinguishability
    ↪ between some pair of programs is interesting. In this work,
    ↪  IND-ADP is as efficient as iO because on inputting a
    ↪ polynomial-size program, the size of the output program is
    ↪ also polynomial. This work considers the construction of iO
    ↪  in [LPST16], which proves that XiO+LWE=>iO. And [KNT18]
    ↪ proves that SXiO+OWF=>iO, without relying on LWE. Therefore
    ↪ , maybe IND-ADP can be used to achieve a better result than
    ↪  [KNT18].
I think the separation on page 24 is not sound. In the definition
    ↪ of iO, we require that the two input programs are of the
    ↪ same size besides sharing the functionality. Moreover,
    ↪ there are many ways to pad an ADP.

Editorial Quality
-----------------
Overall, the paper is well written except for the proof of IND-ADP
    ↪ .
The Decomposition of R divides the rows into two sets: aux-related
    ↪  rows and others. But you do not make some explanation here
    ↪ . At the first glance, I cannot understand why (G^i_input+G
    ↪ ^i_aux) R^i_nimv is a zero matrix since you only mention
    ↪ that one orthogonal to G^i_nimv denoted by \widetilde{R^
    ↪ i_nimv}.
The proof on pages 40 and 41 is difficult for me to understand.
    ↪ More graphs and more explanations are better. Also, the
    ↪ usage of condition 5 and condition 6 should be marked
    ↪ apparently.


* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    ↪ * * * * *
```

```
Review #295C

Overall merit
-------------
1. Reject

Reviewer expertise
------------------
3. Knowledgeable

Paper summary
-------------
Affine determinant programs (ADPs) are the basis of a recent
    ↪ paradigm for the construction of iO candidates proposed by
    ↪ Bartusek et al., which is based on a specific set of
    ↪ randomisation operations for ADPs. Very recently, Yao et al
    ↪ . showed that in general this approach does not lead to
    ↪ secure iO schemes. In this work, the authors propose to use
    ↪  ADPs to realize exponentially efficient iO (xiO), and use
    ↪ it to implement iO via the functional encryption route to
    ↪ obfuscation, as xiO in conjunction with additional
    ↪ cryptographic standard assumptions is known to imply FE.


* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    ↪ * * * * *


Review #295B

Overall merit
-------------
1. Reject

Reviewer expertise
------------------
3. Knowledgeable

Paper summary
-------------
The authors suggest realizing xiO (which in turn implies iO) using
    ↪  the Affine determinant program (ADP) obfuscators of [5].
    ↪ While this construction is known to be insecure in general
    ↪ the authors hope that it works information-theoretically
    ↪ for a limited class C of functions. They provide a list of
    ↪ conditions on C that suffice for such a realization but do
    ↪ not provide any concrete class C that satisfies these
    ↪ conditions.

Questions/Clarifications for Authors
------------------------------------
Q1: Can you provide a non-trivial example of a family C (not
    ↪ necessarily a cryptographic one) that satisfies your
    ↪ conditions?

Q2: It is easy to satisfy some of the conditions via simple
    ↪ padding (for example, 4 can be sastisfied by adding a dummy
    ↪  bit to nimv that does not affect the computation). Please
    ↪ explain which conditions are hard to satisfy. Also, how
    ↪ should we understand condition 6? Please provide some
    ↪ intuition.

Q3: I suspect that any C that satisfies these conditions
    ↪ inherently fails to be pPRF and so the whole approach is
    ↪ doomed to fail. Indeed, given any pair of functions f1,f2
    ↪ from your class there is a short NP witness that proves
    ↪ that f1=f2 --- the witness is a pair of random strings R0,
    ↪ R1 so that the setup(f_0,R_0)=setup(f_1,R_1). This problem
    ↪ is coNP-hard even for extremely simple families of circuits
    ↪  and I doubt that one can realize pPRFs by a class for
    ↪ which coNP-hardness does not hold. In any case, it is the
    ↪ author's job to justify the plausibility of their
    ↪ assumption.

Novelty and Conceptual Contributions
------------------------------------
The paper mainly combines known results plus suggests an extremely
    ↪  strong assumption.

Technical Details
-----------------
Please clarify what is your technical contribution.

Editorial Quality
-----------------
It is very hard to follow the presentation please remove
    ↪ unnecessary details about known results and focus on the
    ↪ new statements (e.g., your construction of XiO).
```

```
Some random points (there are many others):
-footnote 6: what does "non-vanishing" mean here?
- please define "randomized encoding" and give reference to their
    ↪ origin.
- def 3: You say that ADP is a pair of algorithms but "Eval"
    ↪ algorithm is always fixed so the scheme is specified by a
    ↪ single set-up algorithm. Also, in the correctness condition
    ↪ , you use Prog(m) which is undefined. Instead, just replace
    ↪  Prog(m) by det(...) and remove Eval from the scheme.
-BDDs are never defined. Since there are several variants in the
    ↪ literature, it's best to define it properly.
-The randomization in 2.4 should be attributed to Ishai-
    ↪ Kushilevitz-2002 -- [20] is a survey.
 Also, it seems to me that most of the material in this section
     ↪ can be skipped - it's not new and if you need some
     ↪ concrete result state it as a lemma and refer to previous
     ↪ works for a proof.
-Section 3.1: It's not clear to me what are you trying to show and
    ↪  how this is relevant to the main results of the paper. The
    ↪  assumption in def 4 is "funny" and somewhat meaningless
    ↪ without specifying a concrete family of PRFs. The proof of
    ↪ Thm 1 is closely related to the proof of Applebaum-Ishai-
    ↪ Kushilevitz04 regarding the randomness reconstruction
    ↪ property of BP-based randomized encoding.

- p.19: What is P^log/poly?
-Condition 3:|nimv|->nimv
-Conditions 3-4: How is f being represented when it is given to R
    ↪ ?

Reviewer's consensus summary
----------------------------
Summary: The reviewers do not think that the contribution
    ↪ justifies acceptance. The authors are encouraged to improve
    ↪  the presentation and to provide some justification to the
    ↪ suggested approach.
```

K.1 Our Reply

Reviewer A. We thank reviewer A for his/her patience in fully reading our work and for his effort in trying to understand it.

The reviewer's first question relates to a result which for space reasons was left out of this work. To fully answer, the adversary in an *iO* game has full knowledge of both circuits, thus, it always knows the topology structure and the keys. This is the way the *iO* game is defined.

The fifth condition is needed to derive a contradiction. The bad event occurs when line j in the inverse of $\overline{\mathbf{G}}^{-1}_{\mathsf{nimv}||\vec{0}}$ multiplied with column c of $\mathbf{S}$ is not 0, where j is the index of a node that depends on input. In our proof, we derive a contradiction, and we do this by observing that every line j of $\overline{\mathbf{G}}^{-1}_{\mathsf{nimv}||\vec{0}}$ multiplied with every column $d \neq j$ of $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$ is 0; originally $d = c$ but its value will modify with each "iteration"; also one entry in each line in columnc c of $\mathbf{S}$ is part of column c of $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$. The result of the inner product between line j and column d has to be 0. If it is 1, it must be the case that line j has an extra 1 in one of its left position, and it will get multiplied with some other 1 value in the "upper" part of column d such that the result is 0 (if there is no 1, we are done and have the contradiction). But in the worst case, the adjacency matrix may have such a structure and we have to "recurse" and "go up and left" towards the starting node using the new leftmost 1 Finally, we may reach the leftmost column. When we multiply line j with leftmost column, we observe that the product will be 1 because the first column in $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}}$ has form $(0, 1, 0, \ldots, 0$ and line j will have form $(*, 1, \ldots)$. So the result will be 1 and not zero and we get the contradiction. Now, to fully answer, if the first column will not have the first elements set to zero, the result of the inner product between line j and column 1 may be 0, which will prevent us from having a proof.

We hope the above intuition explains the steps for going left and up in our work. We can provide more graphs in the full version, but this one is size restricted.

As regards to go right and go down technique, indeed, this subcase is not really not needed, and it was removed from the current submission. In essence, all that matters for the proof is that on any path including a nimv-dependent node, there is no node depending on input occurring before the nimv-node. An alternative proof can also made available, relying mainly on matrix algebra rather than the current graph-based interpretation (which we thought to be more intuitive).

Reviewer C. We would like to thank for your work and dedication in ensuring each submission enjoyed a fair review process. We are delighted to read reviews such as 295C for our submission, and to observe how nicely did the reviewer articulate his/her viewpoints and supported them with comprehensive and convincing logical arguments. It sets a very good example and a very high standard that we will follow while reviewing others' works.

Reviewer B. To answer Q1: the XOR functionality with its BDD depicted in Figure 1 easily satisfies the constraints.

To answer Q2: please try to read our proof or the above answer to reviewer A. The previous submission was in a 30 pages format and the reviewers were required to fully read the submission (including the proof, which was in the first 30 pages).

To answer Q3: algebraically it is possible to find two sets of randomness terms such that the two ADP implementations coincide, assuming their adjacency matrices will differ in a relatively small set of points. The differing points will select different lines in the randomizing matrices. The relation that has to hold is: $\overline{\mathbf{G}}_{\mathsf{nimv}||\vec{0}} \cdot \mathbf{A} + \mathbf{I}_m \cdot \overline{\mathbf{G}}_{\mathsf{nimv}'||\vec{0}} = \mathbf{O}$.

To further delve into the subject, we have provided a new part (in appendices) that investigates a succinct FE scheme fulfilling such constraints in the prior appendices.

@Non-vanishing: see this definition.
@Prog/Eval: in general, this is a good suggestion; but for the moment we still prefer a more generic definition of ADPs as perhaps we may want to instantiate them differently.
@Previous Section 3: this is not included for space reasons.
@Randomized encodings: we added the suggested reference to [IK02].