Author preprint of:
Barthel, J., Roşie, R. (2021). NIKE from Affine Determinant Programs. In Qiong Huang and Yu Yu, editors, *Provable and Practical Security - ProvSec 2021*, pages 98–115. Lecture Notes in Computer Science, vol 13059. Springer, Cham.
https://doi.org/10.1007/978-3-030-90402-9_6

NIKE from Affine Determinant Programs

Jim Barthel[1] and Răzvan Roşie[2]

[1] University of Luxembourg, jim.barthel@uni.lu
[2] JAO, Luxembourg, rosie@jao.eu

Abstract. A multi-party non-interactive key-exchange (NIKE) scheme enables N users to securely exchange a secret key K in a non-interactive manner. It is well-known that NIKE schemes can be obtained assuming the existence of indistinguishability obfuscation (iO).
In this work, we revisit the original, iO-based, provably-secure NIKE construction by Boneh and Zhandry, aiming to simplify it. The core idea behind our protocol is to replace the functionality of the obfuscator with the one of an affine determinant program (ADP). Although ADPs have been designed with the purpose of attaining indistinguishability obfuscation, such implication is left open for general circuits.
The ingredients enabling to prove the security of our scheme stem into a more careful analysis of the branching programs needed to build ADPs. In particular, we show:
1. An intuitive indistinguishability notion defined for ADPs of puncturable pseudorandom functions (PRFs) is sufficient to prove security for NIKE.
2. A set of simple conditions based on ADP's branching program topology that are sufficient for proving indistinguishability of ADPs. We leave open the question of finding ADPs satisfying them.

Keywords: NIKE, branching programs, affine determinant programs .

1 Introduction

Key-exchange [8] is arguably the simplest public-key cryptographic protocol, and probably one of the most used in real world applications. Intriguingly, since its introduction for the case of two parties, and the advances in exchanging keys between three parties [14], few progress has been achieved in obtaining provable, non-interactive key-exchange protocols between multiple parties (at least four for the problem at hand).

In the last decade, it has been shown that the existence of secure advanced cryptographic primitives, such as multilinear maps [9] or indistinguishability obfuscation (iO) [1] would imply the existence of such non-interactive key-exchange protocols [5]. Until recently, the security of the former cryptographic primitives was less understood, and the problem of building NIKE schemes remained open. A stream of recent works [3,11,12] culminated with the breakthrough result that iO can be obtained from well-understood assumptions [13]. Such results posit the problem of obtaining NIKE in the realizable landscape.

In our work, we propose a new instantiation of the NIKE scheme put forth by Boneh and Zhandry [5], while replacing the circuit to be iO-obfuscated with an affine determinant program [3]. To this end, we proceed with a brief description of the scheme we use, followed by the techniques that allow to plug in the ADP.

1.1 Prior Work on NIKE

NIKE from iO. Boneh and Zhandry [5] put forth a simple NIKE protocol, which can be described as follows: assume that N participants into the protocol have access to a public set of parameters pp. Each participant computes and publishes terms that are designated to the remaining N-1 entities. Finally, each party u, by knowing its own secret key $\mathsf{sk}^{(u)}$ as well as the N-1 terms published by other parties, is able to compute the exchanged key.

Concretely, imagine the exchanged key is retrieved as the output of a specific circuit $\mathscr{C}$ applied over the input domain $\{0,1\}^{N\times h'}$ where h' denotes the length of each $\mathsf{sk}^{(u)}$. Essentially, $\mathscr{C}$ does two things: (1) performs a check that someone using the circuit is authorized to evaluate the key exchange function; (2) computes the result of a (puncturable) PRF over the joint inputs, then outputs K as an exchanged key (see Figure 1).

$\mathcal{C}_{\mathsf{pPRF.k}}\left(\mathsf{sk}^{(u)}, u, \overline{\mathsf{sk}^{(1)}}, \ldots, \overline{\mathsf{sk}^{(N)}}\right)$:

if $\mathsf{PRG}(\mathsf{sk}^{(u)}) = \overline{\mathsf{sk}^{(u)}}$:
    $K \leftarrow \mathsf{pPRF}\left(\mathsf{pPRF.k}, \overline{\mathsf{sk}^{(1)}}||\ldots||\overline{\mathsf{sk}^{(N)}}\right)$
    return K
return $\perp$

Fig. 1. Each party u knows its own private key $\mathsf{sk}^{(u)}$ and releases $\overline{\mathsf{sk}^{(u)}} \leftarrow \mathsf{PRG}\left(\mathsf{sk}^{(u)}\right)$. On input $u, \mathsf{sk}^{(u)}$, and the released values $\left\{\overline{\mathsf{sk}^{(v)}}\right\}_{v\in[N]}$, the circuit checks whether $\mathsf{PRG}\left(\mathsf{sk}^{(u)}\right) = \overline{\mathsf{sk}^{(u)}}$, in which case a PRF value is returned.

The verification subroutine of the circuit $\mathscr{C}$ requires to know the pre-image of some PRG value: user u is required to provide $\mathsf{sk}^{(u)}$ in order to be checked against the already published $\overline{\mathsf{sk}^{(u)}}$, where $\overline{\mathsf{sk}^{(u)}} \leftarrow \mathsf{PRG}(\mathsf{sk}^{(u)})$. If the check passes, evaluate pPRF over all published values and learn

$$K \leftarrow \mathsf{pPRF.Eval}\left(\mathsf{pPRF.k}, \overline{\mathsf{sk}^{(1)}}||\ldots||\overline{\mathsf{sk}^{(N)}}\right) .$$

For the proof, the authors require the PRG to stretch the inputs over a larger output domain, emphasizing that a length-doubling PRG suffices for this task.

The correctness of the scheme follows as all parties evaluate the pPRF in the same point and under the same key. Intuitively, security stems from the necessity to provide a preimage for PRG in order to be able to evaluate pPRF.

1.2 Our Result and Techniques

Our main result is a new instantiation of the multi-partite NIKE protocol proposed in [5]. Our key ingredients are affine determinant programs for puncturable functionalities reaching a natural indistinguishability notion.

Theorem 1 (Informal). *Assuming the existence of secure length-doubling pseudorandom generators, secure puncturable pseudorandom functions in* NC^1*, and indistinguishably-secure affine determinant programs, there exists a secure* NIKE *scheme for* N *parties.*

As our NIKE construction is close to the one in [5], we begin with the intuition for affine determinant programs (introduced in [3]) for PRFs and then present an overview of our techniques.

Affine Determinant Programs - Setup. The idea behind ADPs is to use branching programs in conjunction with decomposability. Consider a PRF keyed[3] by k and its ith bit restriction PRF^i as a function from $\{0,1\}^{k+n}$ to $\{0,1\}$. Assume the circuit representation of PRF^i is $\mathscr{C}^i$ and that it belongs to NC^1 (this is the class we are interested in). We can infer that its branching program has polynomial size [6]. For each $\mathscr{C}^i$, let $\mathbf{G}^i_{\mathsf{k}||\mathsf{input}}$ denote the adjacency matrix[4] of its branching program BP^i.

Following known techniques [10], we slightly post-process this adjacency matrix of BP^i, by removing its first column and last row, in order to obtain $\overline{\mathbf{G}}^i_{\mathsf{k}||\mathsf{input}}$. Then, we left/right multiply it with random binary invertible matrices $\mathbf{L}^i, \mathbf{R}^i$. The resulting matrix $\mathbf{T}^i_{\mathsf{k}||\mathsf{input}}$ satisfies:

$$\det \underbrace{\left(\mathbf{L}^i \cdot \overline{\mathbf{G}}^i_{\mathsf{k}||\mathsf{input}} \cdot \mathbf{R}^i\right)}_{\mathbf{T}^i_{\mathsf{k}||\mathsf{input}}} = \mathsf{BP}^i(\mathsf{k}||\mathsf{input}) = \mathscr{C}^i_{\mathsf{k}}(\mathsf{input}),$$

where "det" denotes the determinant over $\mathbb{F}_2$. The *crux* idea is to decompose *each entry* (u,v) of $\mathbf{L}^i \cdot \overline{\mathbf{G}}^i_{\mathsf{k}||\mathsf{input}} \cdot \mathbf{R}^i$ into sums of input_j-dependent monomials:

$$\mathbf{T}^i_{u,v,\mathsf{k}||\mathsf{input}} \leftarrow \mathbf{T}^i_{u,v,\mathsf{k}} + \mathbf{T}^i_{u,v,\mathsf{input}_1} + \ldots + \mathbf{T}^i_{u,v,\mathsf{input}_n} \ , \tag{1}$$

where each $\mathbf{T}^i_{u,v,\vec{0}_j}, \mathbf{T}^i_{u,v,\vec{1}_j}$ is a sum of degree-three monomials of the form: $l_\alpha \cdot g_\beta \cdot r_\gamma$. Oversimplified, to enable the simulation of $\mathbf{T}^i_{u,v,\mathsf{k}||\mathsf{input}}$ in Equation (1), we reveal all pairs $\{\mathbf{T}^i_{u,v,\vec{0}_j}, \mathbf{T}^i_{u,v,\vec{1}_j}\}$, as depicted in Figure 2.

The setup of the ADP proceeds by generating the branching programs of each boolean function PRF^i and the corresponding adjacency matrices. Then, it samples the invertible matrices $\mathbf{L}^i, \mathbf{R}^i$ to obtain the set:

$$\left\{\mathbf{T}^i_{u,v,\mathsf{k}}, \mathbf{T}^i_{u,v,\mathsf{input}_1}, \ldots, \mathbf{T}^i_{u,v,\mathsf{input}_n}\right\}$$

[3] The length of the key is k.

[4] The structure of adjacency matrix is settled by both k and input, a fact reflected in the notation $\mathbf{G}^i_{\mathsf{k}||\mathsf{input}}$.

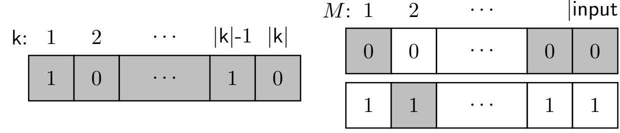

Fig. 2. An element $\mathbf{T}^i_{u,v,\mathsf{k}}$ is provided for the bits of the key k, while elements $\mathbf{T}^i_{u,v,\mathsf{input}_j}$ correspond to the bits in the binary decomposition of the message. We assume $\mathsf{input} = (0, 1, \ldots, 0, 0)$, meaning that the coloured encodings $\{\mathbf{T}^i_{u,v,\vec{0}_1}, \mathbf{T}^i_{u,v,\vec{1}_2}, \ldots, \mathbf{T}^i_{u,v,\vec{0}_{|\mathsf{input}|-1}}, \mathbf{T}^i_{u,v,\vec{0}_{|\mathsf{input}|}}\}$ are selected and are summed up with $\mathbf{T}^i_{u,v,\mathsf{k}}$.

These values are published, for all i and for all entries (u, v) and constitute the *affine determinant program* corresponding to the keyed function PRF.

Affine Determinant Programs - Evaluation. Running the ADP.Eval is straightforward. Given the input message $\mathsf{input} \coloneqq (\mathsf{input}_1, \ldots, \mathsf{input}_n)$, a first step reconstructs:

$$\mathbf{T}^i_{u,v,\mathsf{k}||\mathsf{input}} \leftarrow \mathbf{T}^i_{u,v,\mathsf{k}} + \mathbf{T}^i_{u,v,\mathsf{input}_1} + \ldots + \mathbf{T}^i_{u,v,\mathsf{input}_n} \ ,$$

by simply selecting the terms corresponding to $\mathsf{input}_1, \ldots, \mathsf{input}_n$. In this way, we recover a value $\mathbf{T}^i_{u,v,\mathsf{k}||\mathsf{input}}$ corresponding to some position (u, v).

Repeating this reconstruction for every entry (u, v), we recover the desired matrix $\mathbf{T}^i_{\mathsf{k}||\mathsf{input}}$. Then, ADP.Eval computes the determinant of $\mathbf{T}^i_{\mathsf{k}||\mathsf{input}}$ and recovers one bit in the output of $\mathscr{C}$. This step is repeated for every output bit i of $\mathscr{C}$. We stress that $\mathbf{T}^i_{\mathsf{k}||\mathsf{input}} \leftarrow \mathbf{L}^i \cdot \overline{\mathbf{G}}^i_{\mathsf{k}||\mathsf{input}} \cdot \mathbf{R}^i$, and its determinant is in fact $\mathscr{C}^i_{\mathsf{k}}(\mathsf{input})$ ($\mathscr{C}$'s i^{th} output bit) where $\overline{\mathbf{G}}^i_{\mathsf{k}||\mathsf{input}}$ is "close to" the adjacency matrix of BP.

Affine Determinant Programs - Reducing the size of the program. The size of ADPs can be improved in the following way: instead of releasing both values $\mathbf{T}^i_{u,v,\vec{0}_j}$ and $\mathbf{T}^i_{u,v,\vec{1}_j}$, we add to $\mathbf{T}^i_{u,v,\mathsf{k}}$ the sum corresponding to the *all-zero* message:

$$\mathbf{T}^i_{u,v,\mathsf{k}} + \sum_{j=1}^{n} \mathbf{T}^i_{u,v,\vec{0}_j} \ .$$

To ensure correctness, we release the difference terms corresponding to each:

$$\mathbf{T}^i_{\Delta_{u,v,j}} \leftarrow \mathbf{T}^i_{u,v,\vec{1}_j} - \mathbf{T}^i_{u,v,\vec{0}_j}, \tag{2}$$

which can be used to reconstruct the sum in Equation (1). We stress that a user could always recover the difference of monomial-sums depending on 1 and 0 in position (u, v) if it was given both $\mathbf{T}^i_{u,v,\vec{0}_j}$ and $\mathbf{T}^i_{u,v,\vec{1}_j}$. Informally, by providing the difference, we also reduce the amount of information provided to the adversary.

NIKE from IND-Secure ADPs.

We observe that ADPs corresponding to *puncturable* PRFs that enjoy a natural indistinguishability property, suffice for our proof. By indistinguishability, we mean that given two different punctured keys[5] for two different points, which are embedded in two equivalent circuits[6], the ADPs of such circuits are indistinguishable. We are able to prove NIKE security under indistinguishable ADPs only for circuits which have *identical* structure but embed different keys. This latter fact constrains us to use a punctured key in our real NIKE protocol in order to make the proof of Theorem 1 work, as usually $\mathsf{pPRF.Eval}(\cdot)$ and $\mathsf{pPRF.PuncEval}(\cdot)$ have different circuit representations, and ADP indistinguishability will not be achievable.

More interesting than the NIKE proof is the ADP indistinguishability. In the full version we provide a thorough analysis of ADPs' security. To this end, we rewrite ADPs into a form that isolates the differing variables occurring in the BP's adjacency matrices. The lack of standard complexity assumption to work with forces us to investigate the perfect security. Finally, we show that ADPs admitting BP representations having the first line set to $(0, \ldots, 0, 1)$ and where the input dependent nodes occur "after" the sensitive nodes admit perfectly secure ADPs. We leave open the problem of obtaining such BP representation.

Paper Organization. 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 3 reviews the construction of randomized encodings from branching programs and introduces the novel concept of augmented branching programs In Section 4, we describe our NIKE scheme. Section 5 describes our conditions on circuits and BPs to admit indistinguishably-secure ADPs, while in the full version we provide a detail look into ADPs and prove they achieve indistinguishability.

2 Background

Notations. We denote the security parameter by $\lambda \in \mathbb{N}^*$ and we assume it is implicitly given to all algorithms in the unary representation 1^λ. An algorithm is equivalent to a Turing machine. Algorithms are assumed to be randomized unless stated otherwise; ppt stands for "probabilistic polynomial-time" in the security parameter (rather than the total length of its inputs). Given a randomized algorithm $\mathcal{A}$ we denote the action of running $\mathcal{A}$ on input(s) $(1^\lambda, x_1, \ldots)$ with uniform random coins r and assigning the output(s) to $(y_1, \ldots)$ by $(y_1, \ldots) \leftarrow_{\$} \mathcal{A}(1^\lambda, x_1, \ldots; r)$. When $\mathcal{A}$ is given oracle access to some procedure $\mathcal{O}$, we write $\mathcal{A}^{\mathcal{O}}$. For a finite set S, we denote its cardinality by $|S|$ and the action of sampling a uniformly at random element x from X by $x \leftarrow_{\$} X$. We let bold variables such as $\vec{w}$ 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

[5] This is a significant difference to [5].

[6] The circuits are equivalent by preventing the evaluation of the pPRF at the punctured points through simple sanity checks.

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 $[k] \coloneqq \{1, \ldots, k\}$. 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. For completeness, we recall standard algorithmic and cryptographic primitives to be used. 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 Randomized Encodings

Definition 1 (Randomized Encoding Scheme). *A randomized encoding scheme* RE *for a function* $f : \{0,1\}^n \to \mathcal{Y}$ *consists of a randomness distribution* $\mathcal{R}$*, an encoding function* $\mathsf{Encode} : \{0,1\}^n \times \mathcal{R} \to \{0,1\}^\ell$*, and a decoding function* $\mathsf{Decode} : \{0,1\}^\ell \to \mathcal{Y}$*.*

A randomized encoding scheme $\mathsf{RE} \coloneqq (\mathcal{R}, \mathsf{Encode}, \mathsf{Decode})$ *should satisfy:*

- ***Correctness.*** *For any input* $M \in \{0,1\}^n$*,*

$$\Pr_{R \leftarrow \mathcal{R}}[\mathsf{Decode}(\mathsf{Encode}(M; R)) = \mathscr{C}(M)] = 1 \ .$$

- ***Security.*** *For all* $M, M' \in \{0,1\}^n$ *with* $\mathscr{C}(M) = \mathscr{C}(M')$*, the distribution of* $\mathsf{Encode}(M; R)$ *is identical to the distribution of* $\mathsf{Encode}(M'; R)$ *when sampling* $R \leftarrow_\$ \mathcal{R}$*.*

The definition of security can be relaxed, just requiring that $\mathsf{Encode}(M; R)$ *and* $\mathsf{Encode}(M'; R)$ *cannot be effectively distinguished by small circuits. Formally:*

- (s, δ)***-Security.*** *For all* $M, M' \in \{0,1\}^n$ *such that* $\mathscr{C}(M) = \mathscr{C}(M')$*, for any circuit* $\mathscr{C} : \{0,1\}^\ell \to \{0,1\}$ *of size at most* s*,*

$$\Pr_{R \leftarrow_\$ \mathcal{R}}[\mathscr{C}(\mathsf{Encode}(M; R)) = 1] - \Pr_{R \leftarrow_\$ \mathcal{R}}[\mathscr{C}(\mathsf{Encode}(M'; R)) = 1] \leq \delta.$$

2.2 Multi-Party Non-Interactive Key-Exchange

Non-interactive key-exchange (multi-partite) is a beautiful problem in cryptography. All known, provably-secure constructions rely on either multilinear maps or iO constructions. NIKE definition follows.

Definition 2. *A non-interactive key-exchange (NIKE) scheme consists in a triple of polynomial-time algorithms* $(\mathsf{Setup}, \mathsf{Publish}, \mathsf{KeyGen})$ *behaving as follows:*

- $\mathsf{pars} \leftarrow_\$ \mathsf{Setup}(1^\lambda, N)$*: given the security parameter* λ *in unary and the number of participant parties* N*, the algorithm generates the public parameters* pars*.*

– $(\mathsf{pk}^{(u)}, \mathsf{sk}^{(u)}) \leftarrow_\$ \mathsf{Publish}(\mathsf{pars}, u)$: *each party u of N taking part into the protocol derives its own secret and public keys. While $\mathsf{sk}^{(u)}$ is kept secret, $\mathsf{pk}^{(u)}$ is publicly disclosed.*
– $K \leftarrow_\$ \mathsf{KeyGen}(\mathsf{pars}, u, \mathsf{sk}^{(u)}, \mathsf{pk}^{(u)}, \mathsf{pk}^{(v\in S)})$: *the key-generation procedure corresponding to party $u \in [N]$ uses its secret key $\mathsf{sk}^{(u)}$ together with the public keys of all participants $v \in [N]$ to derive the common key K.*

The ***correctness*** *requirement states that any two parties u and v must derive the same key K:*

$$\forall (u,v) \in [N] \times [N] :$$
$$\Pr\left[K_u = K_v \;\middle|\; \begin{array}{l} K_u \leftarrow_\$ \mathsf{KeyGen}(\mathsf{pars}, u, \mathsf{sk}^{(u)}, \mathsf{pk}^{(v\in[N])}) \wedge \\ K_v \leftarrow_\$ \mathsf{KeyGen}(\mathsf{pars}, v, \mathsf{sk}^{(v)}, \mathsf{pk}^{(u\in[N])}) \end{array} \right] \in 1 - \textsc{Negl}(\lambda).$$

Security: the security experiment we present corresponds to the *static* version of the one presented in [5]. Namely, the advantage of any ppt-bounded adversary in winning the game defined in Figure 3 (left) is bounded:

$$\mathbf{Adv}^{\mathsf{IND-NIKE}}_{\mathcal{A},\mathsf{NIKE}}(\lambda) \coloneqq \left| \Pr\left[1 \leftarrow_\$ \mathsf{IND} - \mathsf{NIKE}^{\mathcal{A}}_{\mathsf{NIKE}}(\lambda) \right] - \frac{1}{2} \right| \in \textsc{Negl}(\lambda) .$$

$\underline{\mathsf{IND} - \mathsf{NIKE}^{\mathcal{A},N}_{\mathsf{NIKE}}(\lambda)}$:
$b \leftarrow_\$ \{0,1\}$
$\mathsf{pp} \leftarrow_\$ \mathsf{NIKE.Setup}(1^\lambda, N)$
for $u \in [N]$:
  $\left(\mathsf{sk}^{(u)}, \mathsf{pk}^{(u)}\right) \leftarrow_\$ \mathsf{NIKE.Publish}(\mathsf{pp}, u)$
$K \leftarrow \mathsf{NIKE.KeyGen}\left(\mathsf{pp}, 1, \mathsf{sk}^{(1)}, \{\mathsf{pk}^{(u)}\}_{u\in[N]}\right)$
if $b = 1$, **then** $K \leftarrow \{0,1\}^{|K|}$
$b' \leftarrow \mathcal{A}\left(1^\lambda, \mathsf{pp}, K, \{\mathsf{pk}^{(u)}\}_{u\in[N]}\right)$
return $(b' \stackrel{?}{=} b)$

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

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

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

2.3 Affine Determinant Programs

Section 1 provides the intuition behind Affine Determinant Programs. Below, we formalize the notion, as introduced in [3], but postpone the formal construction from randomized encodings of branching programs to Section 3.3.

Definition 3 (Affine Determinant Programs). *An affine determinant program consists of two ppt 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 $\mathscr{C} : \{0,1\}^n \to \{0,1\}$, output a program* Prog *consisting of $n+1$ square matrices $\mathbf{T}_i$ over some algebraic structure $\mathfrak{S}$ and having dimensions $\mathsf{poly}(n)$. The matrices correspond to $\mathscr{C}$.*

– $b \leftarrow \mathsf{ADP.Eval}(\mathsf{Prog}, M)$: *given the program* Prog *and some input* M*, return a value* b*.* b *is computed as the determinant of the subset sum:* $\mathbf{T}_0 + \sum_{i=1}^{n} \mathbf{T}_i^{M_i}$.

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

$$\Pr\left[\mathscr{C}(M) = \mathsf{ADP.Eval}(\mathsf{Prog}, M) \;\middle|\; \mathsf{Prog} \leftarrow_{\$} \mathsf{ADP.Setup}(1^\lambda, \mathscr{C})\right] = 1 \ .$$

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* $\forall M \in \{0,1\}^\lambda, \mathscr{C}_1(M) = \mathscr{C}_2(M)$ *it holds that*

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

The security definition above makes clear the link between the security of an iO obfuscator [1] and indistinguishability for ADPs. Trivially, a secure IND − ADP affine determinant program gives rise to an indistinguishability obfuscator for the specific class of circuits.

3 Warm-Up: ADP from Randomized Encodings

3.1 Randomized Encodings via Branching Programs

A branching programs corresponds to a sequential evaluation of a function. Depending on each input bit, a specific branch of a circuit computing the function is followed until a terminal node – 0 or 1 – is reached (we assume we only work with single bit output functions). We highlight that any function $\mathscr{C} : \{0,1\}^n \to \{0,1\}$ in NC^1 admits a polynomial size branching program representation. As a consequence of this fact, any function $\mathscr{C}' : \{0,1\}^n \to \{0,1\}^{n'}$ can be thought of as a concatenation of n' branching programs, each outputting a single bit. In an acclaimed result, Barrington [2] shows that the shorter the depth of the circuit representation of $\mathscr{C}$, the shorter the length of the branching program. In independent results, Ben-Or and Cleve [7] show a matrix-based version of Barrington's proof, where the length of the branching program is *constant* for constant depth circuits.

In this work, we consider $\mathbf{G}_M$ to be the adjacency matrix corresponding to the branching program of some $\mathscr{C} : \{0,1\}^{|M|} \to \{0,1\}$. Let, for technical reasons, the main diagonal be 1s and let each row have at most one extra 1 apart from the 1 appearing on the main diagonal. Let $\overline{\mathbf{G}}_M$ stand for the matrix obtained by removing the first column and the last row of $\mathbf{G}_M$. As shown in [10], $\mathscr{C}(M) = \mathbf{det}(\overline{\mathbf{G}}_M)$. Furthermore, two matrices $\mathbf{R}_l$ and $\mathbf{R}_r$ (sampled from a designated distribution) exist, such that the following relation holds:

$$\mathbf{R}_l \cdot \overline{\mathbf{G}}_M \cdot \mathbf{R}_r = \left(\begin{array}{c|c} \mathbf{0} & \mathscr{C}(M) \\ \hline \mathbf{I} & \mathbf{0} \end{array}\right) = \begin{pmatrix} 0 & 0 & \dots & 0 & \mathscr{C}(M) \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{pmatrix} = \overline{\mathbf{G}}_{\mathscr{C}(M)} \in \mathrm{GF}(2)^{m \times m}$$

Such a representation of $\mathscr{C}(M)$, as a product of fixed matrices $\mathbf{R}_l$ and $\mathbf{R}_r$ plays a role in the simulation security of the randomized encoding. Concretely, the value $\mathscr{C}(M)$ is given to the simulator, which, in turn, is able to simulate a product of either full-ranked matrices or of rank $m-1$, as enforced by the value of $\mathscr{C}(M)$. Therefore, this representation confers an innate randomized encoding. The decoder in the randomized encoding has to compute the determinant of $\mathbf{R}_l \cdot \overline{\mathbf{G}}_M \cdot \mathbf{R}_r$ and recover the value of $\mathscr{C}(M)$, given that $\mathbf{R}_l, \mathbf{R}_r$ are full ranked matrices.

For clarity, $\mathbf{R}_r \in \mathrm{GF}(2)^{m\times m}$ and $\mathbf{R}_l \in \mathrm{GF}(2)^{m\times m}$ have the following forms:

$$\mathbf{R}_r = \begin{pmatrix} 1\,0\,0\ldots 0\;\$ \\ 0\,1\,0\ldots 0\;\$ \\ \vdots\;\vdots\;\vdots \quad\;\; \vdots\;\vdots \\ 0\,0\,0\ldots 1\;\$ \\ 0\,0\,0\ldots 0\;1 \end{pmatrix}, \qquad \mathbf{R}_l = \begin{pmatrix} 1\;\$\;\$\ldots \$\;\$ \\ 0\;1\;\$\ldots \$\;\$ \\ \vdots\;\vdots\;\vdots \quad\;\; \vdots\;\vdots \\ 0\,0\,0\ldots 1\;\$ \\ 0\,0\,0\ldots 0\;1 \end{pmatrix}.$$

A generalization of the previous observation would use different distributions for $\mathbf{R}_l, \mathbf{R}_r$. To this end, let $\mathbf{L}$ and $\mathbf{R}$ be two matrices sampled uniformly at random from the set of invertible matrices over $\mathrm{GF}(2)^{m\times m}$ (i.e., $\mathbf{R}_l$ and $\mathbf{R}_r$ are full rank). One can express

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

Note that:

$$\begin{aligned} \mathbf{L} \cdot \overline{\mathbf{G}}_M \cdot \mathbf{R} &= (\mathbf{L}' \cdot \mathbf{R}_l) \cdot \overline{\mathbf{G}}_M \cdot (\mathbf{R}_r \cdot \mathbf{R}) = \mathbf{L}' \cdot (\mathbf{R}_l \cdot \overline{\mathbf{G}}_M \cdot \mathbf{R}_r) \cdot \mathbf{R} \\ &= \mathbf{L}' \cdot \overline{\mathbf{G}}_{\mathscr{C}(M)} \cdot \mathbf{R}' \end{aligned} \tag{3}$$

Since $\mathbf{L}'$ and $\mathbf{R}'$ are full-rank matrices, $\det\left(\mathbf{L} \cdot \overline{\mathbf{G}}_M \cdot \mathbf{R}\right) = \det\left(\overline{\mathbf{G}}_M\right) = \mathscr{C}(M)$. On a different note, we can observe that each of the $m \times m$ entries of the resulting matrix $\mathbf{T}_M \leftarrow \mathbf{L} \cdot \overline{\mathbf{G}}_M \cdot \mathbf{R}$, can be expressed as a sum of monomials of degree three. As noted in [10], while "splitting" each entry $\mathbf{T}_{i,j}$ into monomials, no monomial depends on more than one input bit of M. Also, each monomial includes one component from each of $\mathbf{L}$ and $\mathbf{R}$. Put differently, each monomial contains at most one entry from $\overline{\mathbf{G}}_M$, which is M-dependent. We return to such a representation while reaching the proof of our construction.

3.2 Augmenting NC^1 Branching Programs for Keyed Functions

This part will be used exclusively in the proof provided in the full version and readers may skip it for the moment and return to it later. In short, we introduce a method to *augment* a branching program with a set of intermediate nodes without changing the behaviour of the program, having the purpose of isolating "sensitive" variables. We use the terminology introduced in Section 3.1. To this end, consider the branching program BP corresponding to some keyed function represented by $\mathscr{C}(\mathsf{k}||M)$; its graph representation consists of two complementary

sets of nodes: one containing the nodes depending on the secret (k), and the other ones depending on the input (the message M[7]). Assume the secret key k is fixed (embedded in the circuit), a fact that settles the nodes depending on k in BP.

What we mean through an augmented branching program is an extra set of nodes that is to be added to the graph of BP. Let v denote a vertex depending on k and let u be any other vertex such that there is an arc $v \to u$ in the digraph representation of BP. We introduce an auxiliary node α between v and u. Now, v is no longer directly connected to u, but rather the link becomes $v \to \alpha \to u$. We present this pictorially in Figure 4.

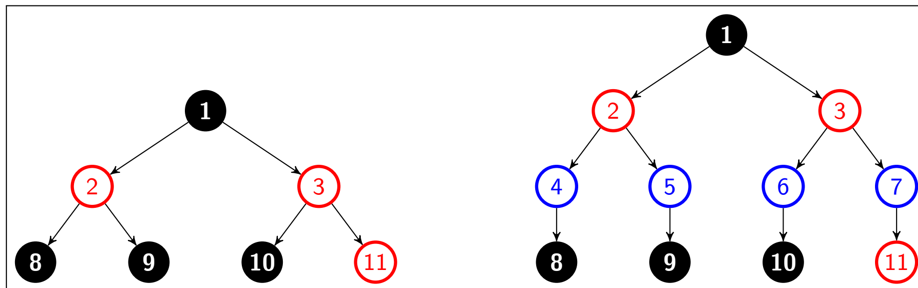

Fig. 4. Left: original branching program. Right: augmented branching program corresponding to $\mathscr{C}$. The auxiliary nodes (4-7) are depicted in blue while red nodes (2,3,11) correspond to nodes settled by the bits of the secret key of the permutation.

Definition 4 (Augmented Branching Programs for NC^1). *Let BP be the branching program corresponding to some circuit $\mathscr{C}_{\mathsf{k}} \in \mathsf{NC}^1$ that embeds k. Let $\mathcal{V}$ denote the set of vertices settled by k. For any vertex $v \in \mathcal{V}$ let u be a vertex such that there exists an arc from v to u. Define the augmented branching program ABP by extending the BP graph and introducing an intermediate vertex α on the path between any node v depending on k and any child vertex u.*

We show that augmenting a branching program preserves the behaviour (correctness) of the original branching program. It is easy to observe that while working over $\mathbb{F}_2$, computing the determinant is equivalent to computing the permanent [15] of a matrix. To deduce correctness of the output, think at the determinant as the sum of $m!$ permutations. If there exists a path from the start node to the node that represents 1, then this path, in conjunction with the 1s on the second diagonal will make one of the sums occurring in the development of the permanent be 1.

Remark 1 (Size of ABP). The size of the augmented branching program ABP is upper-bounded by $3 \times |\mathsf{BP}|$. Assuming the original branching program has $|\mathsf{BP}|$ nodes, each key-dependent node will add two other nodes. Hence the very loose bound of $3 \times |\mathsf{BP}|$.

[7] We assume that any other node (if any) that is input-independent is included in the first set.

The main advantage conferred by ABPs is a decoupling of the rows (or columns) in $\mathbf{R}$ (or $\mathbf{L}$) that depend on the sensitive input (k) from the rest of the nodes. More explicitly, when the dependency graph $\mathbf{G}$ is multiplied with $\mathbf{R}$, the lines in $\mathbf{R}$ that are triggered by the nodes depending on k are separated from the lines in $\mathbf{R}$ that are triggered by the message.

Similarly, the columns in $\mathbf{L}$ can be split in three independent sets, depending on either k, the message or the auxiliary variables (note the asymmetry to $\mathbf{R}$, where we only split the rows in twain).

3.3 ADPs for Keyed Functions from RE

We turn to the usage of randomized encodings for branching programs described in Section 3.1, preserving the goal of instantiating ADPs for keyed functions[8]. We treat each $\mathbf{T}^i_{\mathsf{k}||\mathsf{input}}$ independently, as the product of three matrices. Explicitly, this is:

$$\mathbf{T}^i_{\mathsf{k}||\mathsf{input}} \leftarrow \mathbf{L}^i \cdot \left(\overline{\mathbf{G}}^i_{\mathsf{k}||\mathsf{input}} \cdot \mathbf{R}^i\right) \quad , \quad \forall i \in [|\mathsf{input}|] \tag{4}$$

Remark 2. We remind that for the NIKE scheme, the length of input is $N \cdot h$, and we treat each bit $i \in \mu$ of the exchanged key K independently.

$\mathsf{ADP.Setup}$**:** Using the intuition provided in Section 1, we provide an explicit form for:

$$\mathbf{T}^i_{\mathsf{k}||\vec{0}} \leftarrow \mathbf{L}^i \cdot \overline{\mathbf{G}}^i_{\mathsf{k}||\vec{0}} \cdot \mathbf{R}^i \tag{5}$$

The program Prog^i will consist of $\mathbf{T}^i_{\mathsf{k}||\vec{0}}$ as well as the additional set:

$$\left\{\mathbf{T}^i_{\Delta_j} \leftarrow \mathbf{L}^i \cdot \left(\left(\overline{\mathbf{G}}^i_{\mathsf{k}||\vec{1}_j} - \overline{\mathbf{G}}^i_{\mathsf{k}||\vec{0}_j}\right) \cdot \mathbf{R}^i\right)\right\}_{j \in [N \cdot h]} \tag{6}$$

for each output bit i.

$\mathsf{ADP.Eval}$: to run Prog^i and recover the output of $\mathscr{C}^i_{\mathsf{k}}(\mathsf{input})$ proceed as follows:

$$\mathscr{C}^i_{\mathsf{k}}(\mathsf{input}) = \det\left(\mathbf{T}^i_{\mathsf{k}} + \sum_{j=1}^{N \cdot h} \mathsf{input}_j \cdot \mathbf{T}^i_{\Delta_j}\right) \tag{7}$$

4 Multi-Party NIKE via ADP

Our NIKE scheme follows from the one in [5]. The main significant difference consists in implementing a puncturable PRFs through affine determinant programs, instead of using the full power of an indistinguishability obfuscator for $\mathsf{P/poly}$[9]. The security analysis follows in Section 4.2.

---

[8] Note that we are mainly interested in ADPs for puncturable PRFs.

[9] 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.

4.1 Our NIKE Scheme

We remind below some useful notations to be used: μ stands for the length of the key to be exchanged, $\mathsf{sk}^{(u)}$ denotes the secret key corresponding to party u having length h, and $\overline{\mathsf{sk}^{(u)}}$ denotes the public key.

Definition 5 (NIKE **Scheme for N parties).** *Let* $\mathsf{PRG} : \{0,1\}^h \to \{0,1\}^{2h}$ *denote a secure pseudorandom generator, and let* $\mathsf{pPRF} : \{0,1\}^{(N\cdot 2h)+1} \to \{0,1\}^\mu$ *denote a secure puncturable pseudorandom function. Let* $\mathsf{ADP} : \{0,1\}^{N\cdot 2h} \to \{0,1\}$ *denote an affine determinant program reaching indistinguishability. Define a NIKE scheme as follows.*

- $\mathsf{NIKE.Setup}(1^\lambda, N, h, \mu)$*: the* Setup *is given a number of parties* N*, the length* μ *of the exchanged key and the length* h *of each party's secret keys. For each* $i \in [\mu]$*, initiate the public parameters* $\mathsf{pp} \leftarrow \emptyset$*. For each* $i \in [\mu]$*, repeat the following steps:*
  (1) Sample a pPRF *key and puncture it at point* $\mathbf{0}^{N\cdot 2h+1}$*:*
  $$\mathsf{k}_i \leftarrow \mathsf{pPRF.Puncture}(\mathsf{pPRF.KeyGen}(1^\lambda), \mathbf{0}^{N\cdot 2h+1}) \ .$$
  (2) Consider the following circuit $\mathscr{C}^i$*:*

$\mathscr{C}^i_{\mathsf{pPRF.k}_i}\left(u, \mathsf{sk}_i^{(u)}, \overline{\mathsf{sk}_i^{(1)}}, \ldots, \overline{\mathsf{sk}_i^{(N)}}\right)$:

if $\mathsf{PRG}\left(\mathsf{sk}_i^{(u)}\right) = \overline{\mathsf{sk}_i^{(u)}}$:
  $K \leftarrow \mathsf{pPRF}\left(\mathsf{pPRF.k}_i, 1||\overline{\mathsf{sk}_i^{(1)}}|| \ldots ||\overline{\mathsf{sk}_i^{(N)}}\right)$
  return K_i // the ith bit of K
return $\perp$

Fig. 5. 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 [5].

  (3) Instantiate an ADP from this circuit: $\mathsf{Prog}^i \leftarrow \mathsf{ADP.Setup}(1^\lambda, \mathscr{C}^i_{\mathsf{pPRF.k}_i})$.
  (4) Add to pp *the program* Prog^i*:* $\mathsf{pp} \leftarrow \mathsf{pp} \cup \mathsf{Prog}^i$ *. Publish the public parameters* pp*.*
- $\mathsf{NIKE.Publish}(\mathsf{pp}, u)$*:*
  (1) u *samples* $\{\mathsf{sk}_i^{(u)}\}_{i\in[\mu]} \leftarrow_\$ \{0,1\}^h$*.*
  (2) u *publishes the following value as her public key:*
  $\overline{\mathsf{sk}_i^{(u)}} \leftarrow \mathsf{PRG}\left(\mathsf{sk}_i^{(u)}\right)$.
  These steps are repeated for all $i \in [\mu]$*.*
- $\mathsf{NIKE.KeyGen}(\mathsf{pp}, u, \{\mathsf{sk}_i^{(u)}\}_{i\in[\mu]}, \{\mathsf{Prog}^i\}_{i\in[\mu]}, \{\overline{\mathsf{sk}_i^{(v)}}\}_{i\in[\mu], v\in[N]})$*:*
  (1) Provide to Prog^i *the input* $u, \mathsf{sk}_i^{(u)}, \{\overline{\mathsf{sk}_i^{(v)}}\}_{v\in[N]}$ *and set*
  $K_i \leftarrow \mathsf{ADP}^i.\mathsf{Eval}\left(\mathsf{Prog}^i, \left(u, \mathsf{sk}_i^{(u)}, \{\overline{\mathsf{sk}_i^{(v)}}\}_{v\in[N]}\right)\right)$.
  Repeat these steps for all $i \in [\mu]$*.*

Proposition 1. *The construction in Definition 5 is correct.*

Proof (Proposition 1). See full paper.

4.2 Security from IND-Secure ADP

The proof is structured similarly to the one in [5], up to the variation in the usage of 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. The reason behind embedding a punctured key in the real construction is that we can only prove indistinguishability for ADPs under *identically* structured branching programs. Put differently, it is usually the case that the *normal* and *punctured* evaluation procedures differ for existing pPRFs in NC^1, while we want a unique pPRF.Eval procedure (e.g., [4]).

We also stress that we consider only the static security notion (Definition 5).

Theorem 2. *Let* NIKE *be the scheme described in Section 4.1. Let* PRG *be a secure pseudorandom generator and* pPRF *denote a secure puncturable pseudorandom function. Then, the* NIKE *scheme in Section 4.1 is secure according to Definition 5.*

Proof (Theorem 2). The proof follows through a hybrid argument.

- Game_0: this is the real game, where the adversary is provided either the real exchanged key K or some value sampled uniformly at random.
- $\mathsf{Game}_{1.0}$: is identical to Game_0.
- $\mathsf{Game}_{1.u}$: in this game, we change the distribution of the published parameters pp; instead of issuing $\overline{\mathsf{sk}^{(u)}}$, as the output of the $\mathsf{PRG}(\mathsf{sk}^{(u)})$, party u samples $\overline{\mathsf{sk}^{(u)}}$ over $\{0,1\}^{2h}$. The distance to the previous game is bounded by the security of the PRG. Most importantly, the newly sampled point is not in the co-domain of the PRG with overwhelming probability, due to the PRG stretch.
- $\mathsf{Game}_{1.N}$: all public parameters $\overline{\mathsf{sk}^{(u)}}$ are sampled uniformly.
- Game_2: in this game, the original puncturable PRF key is replaced with a new one, which is punctured in the point $1||\mathsf{sk}^{(1)}||\ldots\mathsf{sk}^{(N)}$. Mind the fact that originally, the key has been punctured in the all-0 point, while the pPRF has been evaluated in an input that always began with 1 (i.e. the evaluation happened for all inputs); for the second case, we note that the PRF will not evaluate over $(\overline{\mathsf{sk}^{(1)}}||\ldots||\overline{\mathsf{sk}^{(N)}})$ as these points have no pre-image in the PRG domain. This happens thanks to the stretch of the PRG. Therefore, the two circuits are equivalent. The advantage of any adversary in noticing this game hope is negligible, down to the $\mathsf{IND}-\mathsf{ADP}$ security of our ADP,

In Game_2, we can bound the advantage of an adversary in retrieving a bit in the K by the advantage of an adversary in guessing the output of the pPRF in the challenge (punctured) point. Concretely, the pPRF game provides the reduction with a key punctured in the challenge point. This punctured key will be embedded into the circuit. The adversary is also provided with the challenge

value the pPRF game provides, which corresponds to either the real NIKE key K (i.e. the real pPRF evaluation) or a uniform value. If the adversary correctly guesses, then it wins the pPRF game. We apply the union bound, and conclude that the advantage of any ppt bounded adversary in winning the IND − NIKE game is upper bounded by:

$$\mathbf{Adv}^{\mathsf{IND-NIKE}}_{\mathcal{A},\mathsf{NIKE}}(\lambda) \leq \mu \cdot N \cdot \mathbf{Adv}^{prg}_{\mathcal{A}_1,\mathsf{PRG}}(\lambda) + \mu \cdot \mathbf{Adv}^{\mathsf{IND-ADP}}_{\mathcal{A}_2,\mathsf{ADP}}(\lambda) + \mu \cdot \mathbf{Adv}^{puncture}_{\mathcal{A}_3,\mathsf{pPRF}}(\lambda) \quad (8)$$

where the right hand side is negligible. □

5 Sufficiency Conditions for IND-Secure ADP

Section 1 provides the intuition behind Affine Determinant Programs. The security definition of ADPs makes clear the link between the security of an iO obfuscator [1] and indistinguishability for ADPs. Trivially, a secure IND − ADP affine determinant program gives rise to an indistinguishability obfuscator for our specific class of circuits.

Furthermore, we can strengthen the security definition in the following sense: for specific classes of functions, it can be the case that $\mathsf{ADP.Setup}(1^\lambda, \mathscr{C}_0; R_0) = \mathsf{ADP.Setup}(1^\lambda, \mathscr{C}_1; R_1)$. That is, for two equivalent functions, we obtain the *same* implementation under two different sets of randomness coins, namely R and R'.

Definition 6 (Colliding-Secure 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 unique R_1 such that:*

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

5.1 Admissible Classes of Functions for Matrix-Based ADPs

A relevant theory should link affine determinant programs to existing problems in cryptographic landscape. Such problems are, for instance, multi-party non-interactive key-exchange schemes or indistinguishability obfuscation. The two primitives can be obtained if there exist obfuscation for puncturable pseudorandom functions, as shown by Boneh and Zhandry for the case of NIKE, and by Pass *et al.* for the case of iO (via XiO).

What we require for Matrix-Based ADPs. Given these observation, our main goal would be to achieve IND − ADP (in fact PS−ADP) branching program-based ADPs for relevant puncturable functions. We state informally the requirements we have over the admissible classes. The requirements are enforced either by the envisioned applications or by the envisioned proof technique.

Requirement 1: The first requirement concerns the depth of circuits. We need circuits to admit branching programs. Thus we need circuits in NC^1.

Requirement 1 *Let $\mathfrak{C}_{d,k+n}$ denote a class of circuits of depth d and input length $k+n$. A necessary condition for $\mathfrak{C}_{d,k+n}$ to admit a matrix-based ADP implementation is*

$$d \in O(\log(k+n)) \ .$$

Requirement 2: Considering that our envisioned applications are built over pseudorandom functions, we are interested in ADPs for keyed functions. Thus we can think at their inputs as concatenation of "$\mathsf{k}||\mathsf{input}$". Our function that is modeled by some circuit can be described as:

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

Requirement 2 *Let $\mathfrak{C}_{d,k+n}$ denote a class of circuits of depth d and input length n. A necessary condition for $\mathfrak{C}_{d,k+n}$ to admit an $\mathsf{PS-ADP}$-secure implementation is that every $\mathscr{C} \in \mathfrak{C}_{d,k+n}$ models a two input function $\mathscr{C} : \{0,1\}^{k+n} \to \{0,1\}$.*

Requirement 3: We only consider functions that are non-constant under different keys. This requirement is motivated by the need to use invertible matrices in our proof[10]. Formally, the condition becomes:

Requirement 3 *Let $\mathfrak{C}_{d,k+n}$ denote a class of circuits of depth d and input length n. A necessary condition for $\mathfrak{C}_{d,k+n}$ to admit a $\mathsf{PS-ADP}$-secure implementation is that: for every $\mathscr{C} \in \mathfrak{C}_{d,k+n}$ modelling some f, there exists a ppt algorithm $\mathcal{R}$ such that:*

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

The condition should be read as: there exists a ppt procedure able to find some key and some input such that $f(\mathsf{k},\mathsf{input}) = 1$. We **stress** that $\mathcal{R}$ is **not** required to sample uniformly at random the input point, nor the key k.

Requirement 4: In words, we would like to have an efficient procedure $\mathcal{R}$ capable of generating two keys (k,k') such that $f(\mathsf{k},X) = f(\mathsf{k}',X)$. Formally:

Requirement 4 *Let $\mathfrak{C}_{d,k+n}$ denote a class of circuits of depth d and input length n. A necessary condition for $\mathfrak{C}_{d,k+n}$ to admit a $\mathsf{PS-ADP}$-secure implementation is that: for every $\mathscr{C} \in \mathfrak{C}_{d,k+n}$ modelling some f, there exists a ppt algorithm $\mathcal{R}$ such that $\forall \ \mathsf{input} \in \mathcal{X}$,*

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

Remark 3. $\mathcal{R}$ is not required to sample the keys uniformly at random.

Requirement 5: Another requirement enforced by our security proof in the full version is that the first *line* in matrix $\overline{\mathbf{G}}_{\mathsf{k}||\vec{0}}$ has the form

$$(0, 0, 0, *, *, \ldots, *)$$

[10] We want the function to be puncturable: under two different keys to obtain the same result under multiple points. Ideally, the punctured point will be excluded from the input space.

From a **high-level** point of view, we can translate this into:

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

An astute reader may observe that for a f fulfilling the constraint above, some branching program can be found such that the first line is not $(0,0,0,*,*,\ldots,*)$. However, without loss of generality, we assume this is not the case.

Requirement 5 *Let $\mathfrak{C}_{d,k+n}$ denote a class of circuits of depth d and input length n. A necessary condition for $\mathfrak{C}_{d,k+n}$ to admit a $\mathsf{PS}-\mathsf{ADP}$-secure implementation is that: for every $\mathscr{C} \in \mathfrak{C}_{d,k+n}$ modelling some f, there exists an efficiently computable key k such that:*

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

Requirement 6: Finally, we are left with the ordering of variables. It is necessary that BP nodes depending on inputs have greater order numbers compared to the nodes depending on k. In layman's terms, the nodes depending on input in a branching program are "below" the ones depending on k.

Requirement 6 *Let $\mathfrak{C}_{d,k+n}$ denote a class of circuits of depth d and input length n. A necessary condition for $\mathfrak{C}_{d,k+n}$ to admit an $\mathsf{PS}-\mathsf{ADP}$-secure implementation is that any index of a node depending on input is greater than any index of a node depending on k.*

Definition 7 (Admissible Class for ADPs via Branching Programs). *Let $\mathfrak{C}_{\lambda,d,|\mathsf{k}|+n}$ 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.*

5.2 Our Claim

We state below our claim in terms of the function belonging to the class $\mathfrak{C}_{d,k+n}$ mentioned above. The proof is provided in the full version.

Theorem 3 ($\mathsf{IND}-\mathsf{ADP}$ programs for ADP admissible functions). *Let $\mathfrak{C}_{d,k+n}$ denote a class of circuits of depth d and input length n which is admissible according to Definition 7. Then, there exists an ADP that reaches $\mathsf{PS}-\mathsf{ADP}$-security with respect to every single $\mathscr{C}$ in the admissible class $\mathfrak{C}_{d,k+n}$. Let pPRF denote a puncturable PRF admitting circuits in NC^1. Let $\mathscr{C}^i$ denote the circuit described in Section 4.1. Let k and k' be two pPRF keys punctured respectively in point $0||0||\ldots||0$ and in some random point $1||\$||\ldots||\$$. Then, the distributions of $\mathsf{ADP}^i(\mathscr{C}^i_{\mathsf{k}})$ and $\mathsf{ADP}^i(\mathscr{C}^i_{\mathsf{k}'})$ are identical.*

Acknowledgements. Jim Barthel was supported in part by the Luxembourg National Research Fund through grant PRIDE15/10621687/SPsquared. Răzvan Roşie was supported in part by ERC grant CLOUDMAP 787390 and is thankful to Hart Montgomery and Arnab Roy for extremely valuable discussions and for a large number of ideas in this work.

References

1. 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.
2. 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.
3. 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.
4. 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.
5. 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.
6. Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 509–539. Springer, Heidelberg, August 2016.
7. Richard Cleve. Towards optimal simulations of formulas by bounded-width programs. *Computational Complexity*, 1(1):91–105, 1991.
8. Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.
9. 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.
10. Yuval Ishai. Randomization techniques for secure computation. *Secure Multi-party Computation*, 10:222, 2013.
11. Aayush Jain, Huijia Lin, Christian Matt, and Amit Sahai. How to leverage hardness of constant-degree expanding polynomials overa $\mathbb{R}$ to build $i\mathcal{O}$. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 251–281. Springer, Heidelberg, May 2019.
12. Aayush Jain, Huijia Lin, and Amit Sahai. Simplifying constructions and assumptions for $i\mathcal{O}$. Cryptology ePrint Archive, Report 2019/1252, 2019. https://eprint.iacr.org/2019/1252.
13. 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.
14. Antoine Joux. A one round protocol for tripartite Diffie-Hellman. *Journal of Cryptology*, 17(4):263–276, September 2004.
15. Leslie G Valiant. The complexity of computing the permanent. *Theoretical computer science*, 8(2):189–201, 1979.