Reducing Multilinear Map Levels in Constrained Pseudorandom Functions and Attribute-based Encryption

Nishanth Chandran[1⋆] Srinivasan Raghuraman[2⋆⋆] Dhinakaran Vinayagamurthy[3⋆⋆⋆]

[1] Microsoft Research, India
[2] Indian Institute of Technology, Madras
[3] University of Toronto

Abstract. The candidate construction of multilinear maps by Garg, Gentry, and Halevi (Eurocrypt 2013) has lead to an explosion of new cryptographic constructions ranging from attribute-based encryption (ABE) for arbitrary polynomial size circuits, to program obfuscation, and to constrained pseudorandom functions (PRFs). Many of these constructions require κ-linear maps for large κ. In this work, we focus on the reduction of κ in certain constructions of access control primitives that are based on κ-linear maps; in particular, we consider the case of constrained PRFs and ABE. We construct the following objects:

- A constrained PRF for arbitrary circuit predicates based on $(n + \ell_{\mathsf{OR}} - 1)-$linear maps (where n is the input length and ℓ_{OR} denotes the OR-depth of the circuit).
- For circuits with a specific structure, we also show how to construct such PRFs based on $(n+\ell_{\mathsf{AND}} - 1)-$linear maps (where ℓ_{AND} denotes the AND-depth of the circuit).
- We then give a black-box construction of a constrained PRF for $\mathbf{NC}^1$ predicates, from any bit-fixing constrained PRF that fixes only *one* of the input bits to 1; we only require that the bit-fixing PRF have certain key homomorphic properties. This gives us a constrained PRF for $\mathbf{NC}^1$ predicates that is based only on n-linear maps, with no dependence on the predicate.
  In contrast, the previous constructions of constrained PRFs (Boneh and Waters, Asiacrypt 2013) required $(n+\ell+1)-$linear maps for circuit predicates (where ℓ is the *total* depth of the circuit) and n-linear maps even for bit-fixing predicates.
- We also show how to extend our techniques to obtain a similar improvement in the case of ABE and construct ABE for arbitrary circuits based on $(\ell_{\mathsf{OR}} + 1)-$linear (respectively $(\ell_{\mathsf{AND}} + 1)-$linear) maps.

1 Introduction

The breakthrough work on multilinear maps [GGH13a] has found tremendous applications in various areas of cryptography. It has lead to attribute-based encryption (ABE) for all polynomial size circuits [GGH$^+$13c], indistinguishability obfuscation and functional encryption for general circuits [GGH$^+$13b], constrained pseudorandom functions [BW13], and so on. Many of these constructions require κ-linear maps for large κ. Larger κ leads to more inefficient schemes and stronger hardness assumptions. In this work, we are interested in exploring the reduction of κ in such constructions – specifically, we consider the case of constrained PRFs and ABE.

Constrained Pseudorandom Functions. A pseudorandom function (PRF) is a keyed function, $F_k(x)$, that is computationally indistinguishable from a truly random function, even to an adversary who has oracle access to the function (but has no knowledge about the key k). Constrained PRFs (introduced in [BW13,BGI14,KPTZ13]), allow the owner of k to give out a constrained key k_f, for a predicate f, such that any user who has k_f can evaluate $F_k(x)$ iff $f(x) = 1$. The security requirement on all points x, such that $f(x) = 0$ is the same as that of standard PRFs.

⋆ Email: nichandr@microsoft.com
⋆⋆ Email: rsrini@cse.iitm.ac.in. Work done while at Microsoft Research, India.
⋆⋆⋆ Email: dhinakaran5@cs.toronto.edu. Work done while at Microsoft Research, India.

Boneh and Waters [BW13] show how to construct constrained PRFs for bit-fixing predicates using an $n-$linear map (where n is the input length to the PRF), and also how to construct constrained PRFs for arbitrary circuit predicates using an $(n+\ell+1)-$linear map (where ℓ is the total depth of the circuit predicate). Constrained PRFs can be used to construct broadcast encryption with small ciphertext length, identity-based key exchange, and policy-based key distribution.

Attribute Based Encryption. Attribute based encryption (ABE) [SW05] allows a more fine-grained access policy to be embedded into public-key encryption. In more detail, in ABE schemes, there is a master authority who owns sk and publishes public parameters as well as a relation $R(x,y)$. A user who encrypts a message m, creates a ciphertext under some string x (that can specify some policy), to obtain $\mathsf{Enc}_{pk}(m,x)$. The master authority can give a user a secret key sk_y. Now, this user can use sk_y to decrypt $\mathsf{Enc}_{pk}(m,x)$ and obtain m iff $R(x,y)=1$; otherwise, the user obtains no information about m. ABE, for the class of relations $R \in \mathbf{NC}^1$ can be constructed based on bilinear maps [GPSW06]. Recently, the work of [GGH$^+$13c] shows how to construct ABE for arbitrary circuits based on $(\ell+1)-$linear maps (where ℓ is the depth of the relation R when expressed as a boolean circuit), while [GVW13] also show how to construct ABE for arbitrary circuits based on the Learning with Errors (LWE) hardness problem.

1.1 Our Results

In this work, we show the following results:

- We construct constrained PRFs for arbitrary circuit predicates using an $(n+\ell_{\mathsf{OR}}-1)-$linear map, where n is the input length to the PRF and ℓ_{OR} denotes the OR-depth of the constraint f when expressed as a boolean circuit (informally, the OR-depth of a circuit is defined to be the maximum number of OR gates from input wires to the output wire along any path in the circuit). We believe that the reduction in linearity is important even in cases when it is not an asymptotic improvement as lower linearity results in a weaker hardness assumption.
- Next, we construct constrained PRFs for circuit predicates using an $(n+\ell_{\mathsf{AND}}-1)-$linear map, where ℓ_{AND} denotes the AND-depth of the constraint f (informally, the AND-depth of a circuit is defined to be the maximum number of AND gates from input wires to the output wire along any path in the circuit). Although in this construction, we require the circuit to be of a specific structure, we show that for several circuits, our construction reduces the number of levels of multilinear map needed.
- Then, we show (in a black-box manner) how to convert any bit-fixing constrained PRF that fixes only *one* bit[4] to 1 into a constrained PRF for $\mathbf{NC}^1$ circuits; we only require that the bit-fixing PRF have certain additive key-homomorphic properties. From this, we obtain a constrained PRF for all predicates $f \in \mathbf{NC}^1$ using an $n-$linear map. In particular, the number of levels in our construction has no dependence on f. We believe this construction to be of independent interest as the only known (non-trivial) constructions of constrained PRFs are based on multilinear maps.
- Finally, we show how to extend our techniques to construct ABE schemes from lesser levels of multi-linear maps.

Similar to [BW13], all our constructions are based on the κ-Multilinear Decisional Diffie-Hellman (κ-MDDH) assumption and achieve selective security (i.e., the adversary must commit to the challenge query at the beginning of the security game); as in [BW13], we can achieve standard security via complexity leveraging. We remark that our techniques can be extended to the constructions of verifiable constrained PRFs [Fuc14,CRV], thereby leading to a similar lowering of κ.

Other related works. The work of [FKPR14] considers the prefix-fixing constrained PRF from the classical GGM construction [GGM86], and shows how to avoid an exponential (in n) loss in security when going from selective security to adaptive security. Their work also shows that any "simple"' reduction, that proves full

[4] By symmetry, we can also start with a bit-fixing constrained PRF that fixes only one bit to 0.

security of the bit-fixing constrained PRF of [BW13], from a non-interactive hardness assumption, must incur an exponential security loss. The work of [HKKW] shows how to construct adaptively secure constrained PRFs for circuits from indistinguishability obfuscation in the random oracle model. More recently, key-homomorphic constrained PRFs were constructed in [BV15,BFP+15][5]. Banerjee et al. [BFP+15] also note that [BW13] is "key-homomorphic".

Security of multilinear maps. After the initial work of Garg et al. [GGH13a], Coron, Lepoint and Tibouchi proposed a multilinear maps construction over the integers [CLT13] also based on ideal lattices. But, Cheon, Han, Lee, Ryu and Stehlé [CHL+15] proposed an attack which completely broke the CLT scheme by recovering the secret parameters of the scheme in polynomial time. After some failed attempts to fix the CLT scheme by various groups, Coron et al. [CLT15] proposed another candidate construction which remains unbroken. Recently, Hu and Jia [HJ15] showed that the κ-MDDH assumption in [GGH13a] does not hold when encodings of zero are provided. Independent of these, Gentry, Gorbunov and Halevi [GGH15] proposed a multilinear maps construction based on *random* lattices but with the map defined with respect to a directed acyclic graph.

To instantiate our constructions, any multilinear maps scheme which is "secure" under the κ-MDDH assumption can be used.

1.2 Our Techniques

Our starting point is the constrained PRF construction of [BW13] for arbitrary circuit predicates. We first view this construction differently as follows. Let the PRF in [BW13] be denoted by $\mathsf{PRF}_{n+\ell}(u, x)$, where u is the key of the PRF, x, an n-bit string, is the input to the PRF, and $\mathsf{PRF}_{n+\ell}$ denotes that the PRF output is at the $(n+\ell)-$level of the multilinear map (where ℓ denotes the depth of the constraint f). Now, in order to give out a constrained key for f, we first pick a random value r_w for every wire w in the circuit. Let j denote the depth of this wire in the circuit. Now, for a given x such that $f(x) = 1$, the idea is to give a key that will enable the user to compute $\mathsf{PRF}_{n+j}(r_w, x)$ for all wires w in the circuit that evaluate to 1 on x. Doing this inductively will allow the compution of $\mathsf{PRF}_{n+\ell}(u, x)$. Let w be an output to some gate in the circuit and let $A(w), B(w)$ be the input wires corresponding to this gate. If this gate is an AND (respectively OR) gate, we give a key, that will allow a user to compute $\mathsf{PRF}_{n+j}(r_w, x)$ from the values $\mathsf{PRF}_{n+j-1}(r_{A(w)}, x)$ AND (respectively OR) $\mathsf{PRF}_{n+j-1}(r_{B(w)}, x)$.

Free AND construction. Our first observation is that for AND gates, one must be able to compute the PRF value corresponding to w wire iff one has the PRF values corresponding to *both* $A(w)$ and $B(w)$. Now, suppose the PRF under consideration is "additively homomorphic" in some sense. Then, we observe that given $\mathsf{PRF}_{n+j-1}(r_{A(w)}, x)$ and $\mathsf{PRF}_{n+j-1}(r_{B(w)}, x)$, one can compute $\mathsf{PRF}_{n+j-1}(r_w, x)$, without the need for additional keys and without jumping a level in the multilinear map as long as we set $r_{A(w)}$ and $r_{B(w)}$ to be random additive shares of r_w. Now, this ensures that AND gates are "free" in the circuit. The OR gates are handled exactly as in the case of [BW13]. This leads to a construction that only makes use of a $(n + \ell_{\mathsf{OR}} - 1)-$linear map.

While this is the main change made to the construction, the proof of security now requires attention. At a very high level, [BW13] could embed a part of the "hard problem" from the hardness assumption at every layer of the circuit as they give out keys for all gates in the circuits. In our case, we do not have that luxury. In particular, since we do not give any keys for AND gates, the structure of the hard problem may be distorted after multiple evaluations of AND gates. In order to overcome this, we must carefully give out the keys at OR levels to "reset" the problem to be of our preferred form. This enables us to then prove security.

Free OR construction. Now, suppose we turn our attention towards the OR gates alone. Note, that one must be able to compute the PRF value corresponding to wire w iff one has the PRF values corresponding to *either* $A(w)$ or $B(w)$. Now, suppose we set $r_w = r_{A(w)} = r_{B(w)}$, then this enables the computation of

[5] We note that these works are concurrent to our work, which was also submitted to TCC.

$\mathsf{PRF}_{n+j-1}(r_w, x)$ from either $\mathsf{PRF}_{n+j-1}(r_{A(w)}, x)$ or $\mathsf{PRF}_{n+j-1}(r_{B(w)}, x)$, without the need for additional keys and without jumping a level in the multilinear map. However, doing this naïvely would lead to a similar "backtracking attack" as the attack described by [GGH+13c] in the context of ABE. In more detail, note that if $A(w) = 0$ and $B(w) = 1$, one can indeed (rightly) compute $\mathsf{PRF}_{n+j-1}(r_w, x)$ from $\mathsf{PRF}_{n+j-1}(r_{B(w)}, x)$ as both $B(w)$ and w are 1. However, this also enables the (unauthorized) computation of $\mathsf{PRF}_{n+j-1}(r_{A(w)}, x)$, and if this wire had a fan-out greater than 1, this would lead to an attack on the security of the PRF. Here, we show that if the circuit had a specific structure, then such a construction can still be made to work. We show that several circuits can be converted to this form (with a polynomial blowup) that results in a reduction in the number of multilinear levels needed. We remark that for the construction (and proof) to succeed, one must carefully select the random key values on the circuit for the constrained key , starting *backwards*, from the output wire in the circuit.

$\mathbf{NC}^1$ *construction.* While we obtain our construction of constrained PRF for $\mathbf{NC}^1$ circuits by combining the above two techniques, we note that the proof of security is tricky and requires the simulator to carefully set the random keys in the simulation. In particular, let x^* be the challenge input to the PRF. Now, suppose, the simulator must give out a constrained key for a circuit f such that $f(x^*) = 0$. The simulator must choose all the random keys of the PRFs on each wire in such a way that for all wires that evaluate to 1 on x^*, the key is either chosen randomly by the simulator or can be computed from values that are chosen randomly by the simulator. We show that this can be indeed done by the simulator, thus resulting in the proof of security.

We then show how to generalize this construction to obtain a constrained PRF for $\mathbf{NC}^1$ circuits from any constrained PRF for bit-fixing predicates that fixes only *one bit* and has certain additively homomorphic properties. We believe this construction to be of independent interest as till date, constrained PRFs for any non-trivial predicate, are known only based on multilinear maps.

Finally, we show how to extend our Free AND/OR techniques to the case of ABE. This gives an ABE based on $(\ell_{\mathsf{OR}}+1)-$linear and $(\ell_{\mathsf{AND}}+1)-$linear maps respectively, improving upon the $(\ell+1)-$linear map construction of [GGH+13c].

1.3 Organization

In Section 2, we define constrained PRFs and ABE as well as state the hardness assumption that we make. We also present circuit notation that is used in the rest of the paper. In Section 3, we describe our $(n+\ell_{\mathsf{OR}}-1)-$linear map construction of constrained PRF for arbitrary circuits. We outline our $(n+\ell_{\mathsf{AND}}-1)-$linear map construction in Section 4. We present our $n-$linear map construction of constrained PRF for $\mathbf{NC}^1$ circuits in Section 5 and the black-box construction of constrained PRF for $\mathbf{NC}^1$ circuits from bit-fixing constrained PRFs in Section 6. We finally show the extension of our results to ABE in Appendix D.

2 Preliminaries

2.1 Definitions

Constrained Pseudorandom Functions. A pseudorandom function (PRF) $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$, is a deterministic polynomial (in security parameter λ) time algorithm, that on input a key $k \in \mathcal{K}$ and an input $x \in \mathcal{X}$, outputs $F(k, x) \in \mathcal{Y}$. F has a setup algorithm $\mathsf{Setup}(1^\lambda)$ that on input λ, outputs a key $k \in \mathcal{K}$.

Definition 1. *A PRF $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ is said to be constrained with respect to a set system $\mathcal{S} \subseteq \mathcal{X}$ if there is an additional key space $\mathcal{K}_c$, and there exist algorithms ($F.\mathsf{Constrain}$, $F.\mathsf{Evaluate}$) such that*

- *$F.\mathsf{Constrain}(k, S)$ is a randomized polynomial time algorithm that takes as input a PRF key $k \in \mathcal{K}$ and the description of a set $S \in \mathcal{S}$. It outputs a constrained key $k_S \in \mathcal{K}_c$ which enables the evaluation of $F(k, x)$ for all $x \in S$ and no other x;*

– *F.*$\mathsf{Evaluate}(k_S, x)$ *is a deterministic polynomial time algorithm that takes as input a constrained key* $k_S \in \mathcal{K}_c$ *and an input* $x \in \mathcal{X}$*. If* k_S *is the output of* *F.*$\mathsf{Constrain}(k, S)$ *for some* $k \in \mathcal{K}$*, then* *F.*$\mathsf{Evaluate}(k_S, x)$ *outputs* $F(k, x)$ *if* $x \in S$ *and* $\perp$ *otherwise, where* $\perp \notin \mathcal{Y}$*. We will use the shorthand* $F(k_S, x)$ *for* *F.*$\mathsf{Evaluate}(k_S, x)$*.*

The security of constrained PRFs informally states that given several constrained keys, as well as the output of the PRF on several points of the adversary's choice, the PRF looks random at all points that the adversary could not have computed himself. Let $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ be a constrained PRF with respect to a set system $\mathcal{S}$. Define two experiments Exp_0 and Exp_1. For $b \in \{0, 1\}$, Exp_b proceeds as follows:

1. First, a random key $k \in \mathcal{K}$ is chosen, and two sets $C, V \subseteq \mathcal{X}$ are initialized to $\emptyset$. C will keep track of points on which the adversary will be challenged and V will keep track of points on which the adversary can compute the PRF himself. The experiments will maintain the invariant that $C \cap V = \emptyset$.
2. The adversary is given access to the following oracles:
   – F.$\mathsf{Constrain}$: Given a set $S \in \mathcal{S}$, if $S \cap C = \emptyset$, the oracle returns F.$\mathsf{Constrain}(k, S)$ and updates $V \leftarrow V \cup S$; otherwise, it returns $\perp$.
   – F.$\mathsf{Evaluate}$: Given an input $x \in \mathcal{X}$, if $x \notin C$, the oracle returns $F(k, x)$ and updates $V \leftarrow V \cup x$; otherwise, it returns $\perp$.
   – $\mathsf{Challenge}$: Given $x \in \mathcal{X}$ where $x \notin V$, if $b = 0$, the oracle returns $F(k, x)$; if $b = 1$, the oracle returns a random (consistent) $y \in \mathcal{Y}$. C is updated as $C \leftarrow C \cup x$.
3. The adversary finally outputs a bit $b' \in \{0, 1\}$.
4. For $b \in \{0, 1\}$, define W_b to be the event that that $b' = 1$ in experiment Exp_b. The adversary's advantage $\mathsf{Adv}_{\mathcal{A},F,\mathcal{S}}(\lambda)$ is defined to be $|\Pr[W_0] - \Pr[W_1]|$.

Definition 2. *A constrained PRF* $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$*, is said to be secure, if for all PPT adversaries* $\mathcal{A}$*, we have that* $\mathsf{Adv}_{\mathcal{A},F,\mathcal{S}}(\lambda)$*, is negligible in* λ*.*

Remark. When constructing constrained pseudorandom functions, it will be more convenient to work with the definition where the adversary is allowed to issue only a single challenge query. A standard hybrid argument shows that this definition is equivalent to the one where an adversary is allowed to issue multiple challenge queries. A constrained PRF is selectively secure if the adversary commits to this single challenge query at the beginning of the experiment.

Attribute-based Encryption. An attribute-based encryption (ABE) scheme has the following algorithms:

– $\mathsf{Setup}(1^\lambda, n, \ell)$: This algorithm takes as input the security parameter λ, the length n of input descriptors in the ciphertext, and a bound ℓ on the circuit depth. It outputs the public parameters PP and the master secret key MSK.
– $\mathsf{Encrypt}(PP, x, M)$: This algorithm takes as input the public parameters, $x \in \{0, 1\}^n$ (representing the assignment of boolean variables) and a message M. It outputs a ciphertext CT.
– $\mathsf{KeyGen}(MSK, f)$: This algorithm takes as input the master secret key and a circuit f. It outputs a secret key SK.
– $\mathsf{Decrypt}(SK, CT)$: This algorithm takes as input a secret key and ciphertext and outputs either M or $\perp$.

The correctness of the ABE requires that for all messages M, for all $x \in \{0, 1\}^n$, for all depth ℓ circuits f, with $f(x) = 1$, if $\mathsf{Encrypt}(PP, x, M)$ outputs CT, and $\mathsf{KeyGen}(MSK, f)$ outputs SK, where PP and MSK were obtained as the output of $\mathsf{Setup}(1^\lambda, n, \ell)$, then $\mathsf{Decrypt}(SK, CT) = M$. The security of an ABE scheme is defined through the following game between a challenger Chall and adversary Adv as described below:

– **Setup.** Chall runs $\mathsf{Setup}(1^\lambda, n, \ell)$ and gives PP to Adv; it keeps SK to itself.
– **Phase 1.** Adv makes any polynomial number of queries for circuit descriptions f of its choice. Chall returns $\mathsf{KeyGen}(MSK, f)$.

– **Challenge.** Adv submits two equal length messages M_0 and M_1 as well as an $x^* \in \{0,1\}$ such that for all f queried in Phase 1, $f(x^*) = 0$. Chall flips a bit b and returns $CT^* = \mathsf{Encrypt}(PP, x^*, M_b)$ to Adv.
– **Phase 2.** Phase 1 is repeated with the restriction that $f(x^*) = 0$ for all queried f.
– **Guess.** Adv outputs a bit b'.

Definition 3. *The advantage of* Adv *in the above game is defined to be* $|\Pr[b' = b] - \frac{1}{2}|$. *An ABE for circuits is secure if for all PPT adversaries* Adv*, the advantage of* Adv *is negligible in the security parameter* λ. *An ABE scheme is said to be selectively secure, if* Adv *commits to* x^* *at the beginning of the security game.*

2.2 Assumptions

Leveled multilinear groups. We assume the existence of a group generator $\mathcal{G}$, which takes as input a security paramter 1^λ and a positive integer κ to indicate the number of levels. $\mathcal{G}(1^\lambda, \kappa)$ outputs a sequence of groups $\mathbb{G} = (\mathbb{G}_1, \ldots, \mathbb{G}_\kappa)$ each of large prime order $p > 2^\lambda$. In addition, we let g_i be a canonical generator of $\mathbb{G}_i$ that is known from the group's description. We let $g = g_1$. We assume the existence of a set of multilinear maps $\{e_{i,j} : \mathbb{G}_i \times \mathbb{G}_j \to \mathbb{G}_{i+j} | i, j \geq 1; i+j \leq \kappa\}$. The map $e_{i,j}$ satisfies the following relation: $e_{i,j}(g_i^a, g_j^b) = g_{i+j}^{ab}, \forall a, b \in \mathbb{Z}_p$. When the context is obvious, we will drop the subscripts i, j. For example, we may simply write $e(g_i^a, g_j^b) = g_{i+j}^{ab}$. We define the κ-Multilinear Decisional Diffie-Hellman (κ-MDDH) assumption [GGH13a] as follows:

Assumption 21 *(κ-Multilinear Decisional Diffie-Hellman: κ-MDDH) The κ-Multilinear Decisional Diffie-Hellman (κ-MDDH) problem is as follows: A challenger runs $\mathcal{G}(1^\lambda, \kappa)$ to generate groups and generators of order p. Then it picks random $c_1, \ldots, c_{\kappa+1} \in \mathbb{Z}_p$. The assumption states that given $g = g_1, g^{c_1}, \ldots, g^{c_{\kappa+1}}$, it is hard to distinguish the element $T = g_\kappa^{\prod_{j \in [\kappa+1]} c_j}$ from a random group element in $\mathbb{G}_\kappa$ with better than negligible advantage in λ.*

2.3 Circuit Notation

We will consider layered circuits, where a gate at[6] depth j will receive both of its inputs from wires at depth $j-1$. We also assume that all NOT gates are restricted to the input level. Similar to [BW13], we restrict ourselves to monotonic circuits where gates are either AND or OR gates of two inputs.[7]

Formally, our circuits will be a five tuple $f = (n, q, A, B, \texttt{GateType})$. We let n be the number of inputs and q be the number of gates. We define $\mathsf{inputs} = [n]$, $\mathsf{Wires} = [n+q]$ and $\mathsf{Gates} = [n+q] \backslash [n]$. The wire $n+q$ is designated as the output wire, outputwire. $A : \mathsf{Gates} \to \mathsf{Wires} \backslash \{\mathsf{outputwire}\}$ is a function where $A(w)$ identifies w's first incoming wire and $B : \mathsf{Gates} \to \mathsf{Wires} \backslash \{\mathsf{outputwire}\}$ is a function where $B(w)$ identifies w's second incoming wire. Finally, $\texttt{GateType} : \mathsf{Gates} \to \{\text{AND}, \text{OR}\}$ is a function that identifies a gate as either an AND gate or an OR gate. We let $w > B(w) > A(w)$. Also, define three functions: $\texttt{tot-depth}(w)$, $\texttt{AND-depth}(w)$, and $\texttt{OR-depth}(w)$ that are all 1, when $w \in \mathsf{inputs}$, and in general are equal to the number of gates (respectively AND and OR gates) on the shortest path to an input wire plus one. We let $f(x)$ be the evaluation of f on the input $x \in \{0,1\}^n$, and $f_w(x)$ be the value of the wire w on the input x.

3 A Free-AND Circuit-predicate Construction

We show how to construct a constrained PRF for arbitrary polynomial size circuit predicates, without giving any keys for AND gates, based on $\kappa = (n + \ell_{\mathsf{OR}} - 1)-$linear maps, where ℓ_{OR} denotes the OR-depth of the circuit. The starting point of our construction is the constrained PRF construction of [BW13] which is based on the ABE for circuits [GGH$^+$13c]. [BW13] works with layered circuits. For ease of exposition, we assume a layered circuit where all gates in a particular layer are of the same type (either AND or OR). Circuits have a single output OR gate. Also a layer of gates is not followed by another layer of the same type. We stress that these are only for the purposes of exposition and can be removed as outlined later on in the section.

[6] When the term depth is used, it is synonymous to the notion of tot-depth described ahead.
[7] These restrictions are mostly useful for exposition and do not impact functionality.

3.1 Construction

$F.\mathsf{Setup}(1^\lambda, n, \ell_{\mathsf{OR}})$:
The setup algorithm takes as input the security parameter λ, the bit length, n, of PRF inputs and ℓ_{OR}, the maximum OR-depth[8] of the circuit. The algorithm runs $\mathcal{G}(1^\lambda, \kappa = n + \ell_{\mathsf{OR}} - 1)$ and outputs a sequence of groups $\mathbb{G} = (\mathbb{G}_1, \ldots, \mathbb{G}_\kappa)$ of prime order p with canonical generators $g_1, \ldots, g_\kappa$, where $g = g_1$. It chooses random exponents $u \in \mathbb{Z}_p$ and $(d_{1,0}, d_{1,1}), \ldots, (d_{n,0}, d_{n,1}) \in \mathbb{Z}_p^2$ and computes $D_{m,\beta} = g^{d_{m,\beta}}$ for $m \in [n]$ and $\beta \in \{0,1\}$. It then sets the key of the PRF as:

$$k = (\mathbb{G}, p, g_1, \ldots, g_\kappa, u, d_{1,0}, d_{1,1}, \ldots, d_{n,0}, d_{n,1}, D_{1,0}, D_{1,1}, \ldots, D_{n,0}, D_{n,1})$$

The PRF is $F(k, x) = g_\kappa^{u \prod_{m \in [n]} d_{m,x_m}}$, where x_m is the m^{th} bit of $x \in \{0,1\}^n$.

$F.\mathsf{Constrain}(k, f = (n, q, A, B, \mathtt{GateType}))$:
The constrain algorithm takes as input the key k and a circuit description f. The circuit has $n + q$ wires with n input wires, q gates and the wire $n + q$ designated as the output wire.

To generate a constrained key k_f, the key generation algorithm chooses random $r_1, \ldots, r_n \in \mathbb{Z}_p$, where we think of the random value r_w as being associated with the wire w. For each $w \in [n + q - 1] \backslash [n]$, if $\mathtt{GateType}(w) = \text{AND}$, it sets $r_w = r_{A(w)} + r_{B(w)}$ (where + denotes addition in the group $\mathbb{Z}_p$); otherwise, it chooses $r_w \in \mathbb{Z}_p$ at random. Finally, it sets $r_{n+q} = u$.

The first part of the constrained key is given out as simply all $D_{i,\beta}$ for $i \in [n]$ and $\beta \in \{0,1\}$. Next, the algorithm generates key components. The structure of the key components depends on whether w is an input wire or an output of an OR gate. For AND gates, we do not need to give out any keys. The key components in each case are described below.

- *Input wire.* By convention, if $w \in [n]$, then it corresponds to the w-th input. The key component is: $K_w = g^{r_w d_{w,1}}$.
- *OR gate.* Let $j = \mathtt{OR\text{-}depth}(w)$. The algorithm chooses random $a_w, b_w \in \mathbb{Z}_p$. Then, the algorithm creates key components:

$$K_{w,1} = g^{a_w}, K_{w,2} = g^{b_w}, K_{w,3} = g_{j-1}^{r_w - a_w \cdot r_{A(w)}}, K_{w,4} = g_{j-1}^{r_w - b_w \cdot r_{B(w)}}$$

The constrained key k_f consists of all these key components along with $\{D_{i,\beta}\}$ for $i \in [n]$ and $\beta \in \{0,1\}$.

$F.\mathsf{Evaluate}(k_f, x)$:
The evaluate algorithm takes as input a constrained key k_f for the circuit f and an input $x \in \{0,1\}^n$. The algorithm first checks that $f(x) = 1$, and if not, it aborts. Consider the wire w at OR-depth j. If $f_w(x) = 1$, then, the algorithm computes $E_w = g_{n+j-1}^{r_w \prod_{m \in [n]} d_{m,x_m}}$. If $f_w(x) = 0$, then nothing is computed for that wire. The algorithm proceeds iteratively starting with computing E_1 and proceeds, in order, to compute E_{n+q}. Computing these values in order ensures that the computation on a lower-depth wire that evaluates to 1 will be defined, before the compution on a higher-depth wire. Since $r_{n+q} = u$, $E_{n+q} = g_{n+\ell_{\mathsf{OR}}-1}^{u \prod_{m \in [n]} d_{m,x_m}}$. We show how to compute E_w for all w where $f_w(x) = 1$, case-wise, according to whether the wire is an input, an OR gate or an AND gate. Define $D = D(x) = g_n^{\prod_{m \in [n]} d_{m,x_m}}$, which is computable through pairings.

- *Input wire.* Suppose $f_w(x) = 1$. Through pairing operations, the algorithm computes $g_{n-1}^{\prod_{m \in [n] \backslash \{w\}} d_{m,x_m}}$. It then computes:

$$E_w = e\left(K_w, g_{n-1}^{\prod_{m \in [n] \backslash \{w\}} d_{m,x_m}}\right) = g_n^{r_w \prod_{m \in [n]} d_{m,x_m}}$$

[8] We can define OR-depth of a circuit which is in our specified form as the number of layers comprising of OR gates, plus 1.

– *OR gate.* Let $j = \texttt{OR-depth}(w)$. The computation is performed if $f_w(x) = 1$. Note that in this case, at least one of $f_{A(w)}(x)$ and $f_{B(w)}(x)$ must be 1. If $f_{A(w)}(x) = 1$, the algorithm computes:

$$\begin{aligned} E_w &= e(E_{A(w)}, K_{w,1}) \cdot e(K_{w,3}, D) \\ &= e\left(g_{n+j-2}^{r_{A(w)} \prod_{m\in[n]} d_{m,x_m}}, g^{a_w}\right) \cdot e\left(g_{j-1}^{r_w - a_w \cdot r_{A(w)}}, g_n^{\prod_{m\in[n]} d_{m,x_m}}\right) \\ &= g_{n+j-1}^{r_w \prod_{m\in[n]} d_{m,x_m}} \end{aligned}$$

Otherwise, we have $f_{B(w)}(x) = 1$ and the algorithm computes E_w from $E_{B(w)}, K_{w,2}, K_{w,4}$ in a similar manner.

– *AND gate.* Let $j = \texttt{OR-depth}(w)$. The computation is performed if $f_w(x) = 1$. Note that in this case, $f_{A(w)}(x) = f_{B(w)}(x) = 1$. The algorithm computes:

$$E_w = E_{A(w)} \cdot E_{B(w)} = g_{n+j-1}^{r_{A(w)} \prod_{m\in[n]} d_{m,x_m}} \cdot g_{n+j-1}^{r_{B(w)} \prod_{m\in[n]} d_{m,x_m}} = g_{n+j-1}^{r_w \prod_{m\in[n]} d_{m,x_m}}$$

The procedures above are evaluated in order for all w for which $f_w(x) = 1$. Thus, the algorithm computes $E_{n+q} = g_{n+\ell_{\mathsf{OR}}-1}^{u \prod_{m\in[n]} d_{m,x_m}} = F(k, x)$.

3.2 Proof of Pseudorandomness

The correctness of the constrained PRF is verifiable in a straightforward manner. The security proof is in the selective security model (where the adversary commits to the challenge input x^* at the beginning of the game). To get full security, the proof will use the standard complexity leveraging technique of guessing the challenge x^*; this guess will cause a loss of a $1/2^n$-factor in the reduction.

Theorem 1. *If there exists a PPT adversary $\mathcal{A}$ that breaks the pseudorandomness of our circuit-predicate construction for n-bit inputs with advantage $\epsilon(\lambda)$, then there exists a PPT algorithm $\mathcal{B}$ that breaks the $\kappa = (n + \ell_{\mathsf{OR}} - 1)-$Multilinear Decisional Diffie-Hellman assumption with advantage $\epsilon(\lambda)/2^n$.*

Proof. The algorithm $\mathcal{B}$ first receives a $\kappa = (n+\ell_{\mathsf{OR}}-1)-$MDDH challenge consisting of the group sequence description $\mathbb{G}$ and $g = g_1, g^{c_1}, \ldots, g^{c_{\kappa+1}}$ along with T, where T is either $g_\kappa^{\prod_{m\in[\kappa+1]} c_m}$ or a random group element in $\mathbb{G}_\kappa$.

Setup:
It chooses an $x^* \in \{0,1\}^n$ uniformly at random. Next, it chooses random $z_1, \ldots, z_n \in \mathbb{Z}_p$ and sets $D_{m,\beta} = g^{c_m}$ when $x^*_m = \beta$ and g^{z_m} otherwise, for $m \in [n]$ and $\beta \in \{0,1\}$. This corresponds to setting $d_{m,\beta} = c_m$ when $x^*_m = \beta$ and z_m otherwise. It then sets $u = c_{n+1} \cdot c_{n+2} \cdot \ldots \cdot c_{n+\ell_{\mathsf{OR}}}$. The setup is executed as in the construction.

Constrain:
Suppose a query is made for a secret key for a circuit $f = (n, q, A, B, \texttt{GateType})$. If $f(x^*) = 1$, then $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ generates key components for every wire w, case-wise, according to whether w is an input wire or an OR gate as described below.

Input wire. By convention, if $w \in [n]$, then it corresponds to the w-th input. If $x^*_w = 1$, then $\mathcal{B}$ chooses $\eta_w = r_w$ at random. The key component is:

$$K_w = (D_{w,1})^{r_w} = g^{r_w d_{w,1}}$$

If $x^*_w = 0$, then $\mathcal{B}$ implicitly sets $r_w = c_{n+1} + \eta_w$, where $\eta_w \in \mathbb{Z}_p$ is a randomly chosen element. The key component is:

$$K_w = (g^{c_{n+1}} \cdot g^{\eta_w})^{z_w} = g^{r_w d_{w,1}}$$

OR gate. Suppose that $w \in \mathsf{Gates}$ and that $\mathtt{GateType}(w) = \text{OR}$. In addition, let $j = \mathtt{OR\text{-}depth}(w)$. In order to show that $\mathcal{B}$ can simulate all the key components, we shall additionally show the following property:

Property 1. For any gate $w \in \mathsf{Gates}$, $\mathcal{B}$ will be able to compute $g_j^{r_w}$, where $j = \mathtt{OR\text{-}depth}(w)$.

We will prove the above property through induction on the OR-depth j; doing this will enable us to prove that $\mathcal{B}$ can compute all the key components required to give out the constrained key . The base case of the input wires ($j = 1$) follows as we know that for an input wire w, $\mathcal{B}$ can compute g^{r_w}, where r_w is of the form η_w or $c_{n+1} + \eta_w$. We now proceed to show the computation of the key-components. In each case, we show that property 1 is satisfied.

CASE 1: If $f_w(x^*) = 1$, then $\mathcal{B}$ chooses $\psi_w = a_w$, $\phi_w = b_w$ and $\eta_w = r_w$ at random. Then, $\mathcal{B}$ creates key components:

$$K_{w,1} = g^{a_w}, K_{w,2} = g^{b_w}, K_{w,3} = g_{j-1}^{r_w - a_w \cdot r_{A(w)}}, K_{w,4} = g_{j-1}^{r_w - b_w \cdot r_{B(w)}}$$

By virtue of of property 1, since $\mathtt{OR\text{-}depth}(A(w)) = \mathtt{OR\text{-}depth}(B(w)) = j - 1$, by the induction hypothesis, we know that $\mathcal{B}$ can compute $g_{j-1}^{r_{A(w)}}$ and $g_{j-1}^{r_{B(w)}}$. Hence, $\mathcal{B}$ can compute the above key-components, as the remaining exponents were all chosen at random by $\mathcal{B}$. Further, since r_w was chosen at random, note that $g_j^{r_w}$ can be be computed for this wire, and hence property 1 holds for this wire as well (at OR-depth j).

CASE 2: If $f_w(x^*) = 0$, then $\mathcal{B}$ implicitly sets $r_w = c_{n+1} \cdot \ldots \cdot c_{n+j} + \eta_w$, where $\eta_w \in \mathbb{Z}_p$ is a randomly chosen element. Since η_w was chosen at random, note that $g_j^{r_w}$ can be be computed for this wire (since $g_j^{c_{n+1} \cdot \ldots \cdot c_{n+j}}$ can be computed using j pairings of g^{c_m}, $n+1 \leq m \leq n+j$), and hence property 1 holds for this wire as well. For computing the key-components, the choices of a_w and b_w are done more carefully.

1. Suppose the level before the current level consists of the inputs. $\mathcal{B}$ would know the values of $\eta_{A(w)}$ and $\eta_{B(w)}$, since for input wires, these values are always chosen at random. In this case, $\mathcal{B}$ implicitly sets $a_w = c_{n+j} + \psi_w$ and $b_w = c_{n+j} + \phi_w$, where $\psi_w, \phi_w \in \mathbb{Z}_p$ are randomly chosen elements. Then, $\mathcal{B}$ creates key components:

$$K_{w,1} = g^{c_{n+j} + \psi_w} = g^{a_w}, K_{w,2} = g^{c_{n+j} + \phi_w} = g^{b_w},$$

$$K_{w,3} = g_{j-1}^{\eta_w - c_{n+j} \cdot \eta_{A(w)} - \psi_w (c_{n+1} \cdot \ldots \cdot c_{n+j-1} + \eta_{A(w)})} = g_{j-1}^{r_w - a_w \cdot r_{A(w)}},$$

$$K_{w,4} = g_{j-1}^{\eta_w - c_{n+j} \cdot \eta_{B(w)} - \phi_w (c_{n+1} \cdot \ldots \cdot c_{n+j-1} + \eta_{B(w)})} = g_{j-1}^{r_w - b_w \cdot r_{B(w)}}$$

   $\mathcal{B}$ is able to create the last two key components due to a cancellation. Since $f_{A(w)}(x^*) = f_{B(w)}(x^*) = 0$, $\mathcal{B}$ would have set $r_{A(w)} = c_{n+1} \cdot \ldots \cdot c_{n+j-1} + \eta_{A(w)}$ and $r_{B(w)} = c_{n+1} \cdot \ldots \cdot c_{n+j-1} + \eta_{B(w)}$. Further, $g_{j-1}^{c_{n+1} \cdot \ldots \cdot c_{n+j-1}}$ can be computed using $j - 1$ pairings of g^{c_m}, $n + 1 \leq m \leq n + j - 1$.
2. Suppose the level before the current level consists of AND gates. Since $f_{A(w)}(x^*) = 0$, we have two cases: either one of $f_{A(A(w))}(x^*)$ and $f_{B(A(w))}(x^*)$ is zero, or both of them are zero. $\mathcal{B}$ sets $a_w = c_{n+j} + \psi_w$ in the former case, and $a_w = \frac{1}{2} c_{n+j} + \psi_w$ in the latter case, where $\psi_w \in \mathbb{Z}_p$ is a randomly chosen element. Similarly, since $f_{B(w)}(x^*) = 0$, we have two cases: either one of $f_{A(B(w))}(x^*)$ and $f_{B(B(w))}(x^*)$ must be zero, or both of them must be zero. $\mathcal{B}$ sets $b_w = c_{n+j} + \phi_w$ in the former case, and $b_w = \frac{1}{2} c_{n+j} + \phi_w$ in the latter case, where $\phi_w \in \mathbb{Z}_p$ is a randomly chosen element. Then, $\mathcal{B}$ creates key components:

$$K_{w,1} = g^{a_w}, K_{w,2} = g^{b_w}, K_{w,3} = g_{j-1}^{r_w - a_w \cdot r_{A(w)}}, K_{w,4} = g_{j-1}^{r_w - b_w \cdot r_{B(w)}}$$

   We now show that these components can indeed be computed in every case. Note that the first two components can be computed in every case. Consider $K_{w,3}$ (a similar argument holds for $K_{w,4}$).

   (a) Consider the first case, where one of $f_{A(A(w))}(x^*)$ and $f_{B(A(w))}(x^*)$ is zero. In particular, without loss of generality, assume that $f_{A(A(w))}(x^*) = 0$ and $f_{B(A(w))}(x^*) = 1$. Hence, $\mathcal{B}$ must have set

$r_{A(A(w))} = c_{n+1} \cdot \ldots \cdot c_{n+j-1} + \eta_{A(A(w))}$ and $r_{B(A(w))} = \eta_{B(A(w))}$. Since $A(w)$ is an AND gate, we would have $r_{A(w)} = r_{A(A(w))} + r_{B(A(w))} = c_{n+1} \cdot \ldots \cdot c_{n+j-1} + \eta_{A(A(w))} + \eta_{B(A(w))}$. Hence, we have:

$$\begin{aligned} K_{w,3} &= g_{j-1}^{\eta_w - c_{n+j}(\eta_{A(A(w))} + \eta_{B(A(w))}) - \psi_w(c_{n+1} \cdot \ldots \cdot c_{n+j-1} + \eta_{A(A(w))} + \eta_{B(A(w))})} \\ &= g_{j-1}^{r_w - a_w \cdot r_{A(w)}} \end{aligned}$$

which can be computed as follows. We know the values of $\eta_{A(A(w))}$ and $\eta_{B(A(w))}$. Further, $g_{j-1}^{c_{n+1} \cdot \ldots \cdot c_{n+j-1}}$ can be computed using $j-1$ pairings of g^{c_m}, $n+1 \leq m \leq n+j-1$. Hence the key component can be computed.

(b) Consider the second case, where $f_{A(A(w))}(x^*) = f_{B(A(w))}(x^*) = 0$. Hence, $\mathcal{B}$ must have set $r_{A(A(w))} = c_{n+1} \cdot \ldots \cdot c_{n+j-1} + \eta_{A(A(w))}$ and $r_{B(A(w))} = c_{n+1} \cdot \ldots \cdot c_{n+j-1} + \eta_{B(A(w))}$. Since $A(w)$ is an AND gate, we would have $r_{A(w)} = r_{A(A(w))} + r_{B(A(w))} = 2c_{n+1} \cdot \ldots \cdot c_{n+j-1} + \eta_{A(A(w))} + \eta_{B(A(w))}$. Hence, we have:

$$\begin{aligned} K_{w,3} &= g_{j-1}^{\eta_w - \frac{1}{2}c_{n+j}(\eta_{A(A(w))} + \eta_{B(A(w))}) - \psi_w(2c_{n+1} \cdot \ldots \cdot c_{n+j-1} + \eta_{A(A(w))} + \eta_{B(A(w))})} \\ &= g_{j-1}^{r_w - a_w \cdot r_{A(w)}} \end{aligned}$$

which can be computed as outlined in the former case.

Thus, the four key components can be given out in every case.

AND gate. We now discuss the case of the AND gate. Suppose that $w \in \mathsf{Gates}$ and that $\mathtt{GateType}(w) = \text{AND}$. In addition, let $j = \mathtt{OR\text{-}depth}(w)$. $\mathcal{B}$ implicitly sets $r_w = r_{A(w)} + r_{B(w)}$. Note that we need not choose any a_w or b_w. In fact, r_w is being chosen because the key components being given out for the OR gates involve $r_{A(w)}$, etc., which may potentially be from AND gates. Clearly, property 1 holds here as well, i.e., $g_j^{r_w} = g_j^{r_{A(w)}} \cdot g_j^{r_{B(w)}}$ can be be computed for this wire, since $g_j^{r_{A(w)}}$ and $g_j^{r_{B(w)}}$ can be computed by virtue of of property 1.

Finally, we set, for the output wire $w = n+q$, $\eta_w = 0$, so that $r_w = u$ in $\mathcal{B}$'s internal view. It is easy to see that a_w and b_w have the same distribution in the real game and the game executed by $\mathcal{B}$. In the real game, they are chosen at random and in the game executed by $\mathcal{B}$, they are either chosen at random or are values offset by some random values ψ_w and ϕ_w, respectively. For $w \in [n+q-1]$, r_w also has the same distribution in the real game and the game executed by $\mathcal{B}$. This is true, since in the real game, they are chosen so that randomness on the input wires of an AND gate add up to the randomness on its output wire, and they are chosen at random for an OR gate, while in the game executed by $\mathcal{B}$, they are chosen in the exact same way, where being "chosen at random" is either truly satisfied or are fixed values are offset by random η_w values. Now, we look at r_{n+q}. In the real game, it is a fixed value u, and in the game executed by $\mathcal{B}$, by setting $\eta_{n+q} = 0$, $r_{n+q} = c_{n+1} \cdot c_{n+2} \cdot \ldots \cdot c_{n+\ell_{\mathsf{OR}}} = u$ internally. Hence, they too have the same distribution. Hence all the parameters in the real game and game executed by $\mathcal{B}$ have the identical distribution.

Evaluate:
Suppose a query is made for a secret key for an input $x \in \{0,1\}^n$. If $x = x^*$, then $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ identifies an arbitrary t such that $x_t \neq x_t^*$. Through ℓ_{OR} pairings of g^{c_m}, $n+1 \leq m \leq n+\ell_{\mathsf{OR}}$, it computes $H = g_{\ell_{\mathsf{OR}}}^u = g_{\ell_{\mathsf{OR}}}^{c_{n+1} \cdot \ldots \cdot c_{n+\ell_{\mathsf{OR}}}}$. Then, through pairing of $D_{m,x_m} \forall m \in [n] \backslash \{t\}$, it computes $g_{n-1}^{\prod_{m \in [n] \backslash \{t\}} d_{m,x_m}}$ and raises it to $d_{t,x_t} = z_t$ to get $H' = g_{n-1}^{\prod_{m \in [n]} d_{m,x_m}}$. Finally, it computes $H'' = e(H, H') = g_{n+\ell_{\mathsf{OR}}-1}^{u \prod_{m \in [n]} d_{m,x_m}} = F(k,x)$ and outputs it. Eventually, $\mathcal{A}$ will issue a challenge input $\tilde{x}$. If $\tilde{x} = x^*$, $\mathcal{B}$ will return the value T and output the same bit as $\mathcal{A}$ does as its guess. If $\tilde{x} \neq x^*$, $\mathcal{B}$ outputs a random bit as its guess.

This completes the description of the adversary $\mathcal{B}$. We first note that in the case where T is part of a MDDH tuple, the real game and game executed by $\mathcal{B}$ have the identical distribution. Secondly, in both cases

(i.e., whether or not T is part of the MDDH tuple), as long as $\mathcal{B}$ does not abort, once again, the real game and game executed by $\mathcal{B}$ have the identical distribution, except for the output of $\mathcal{B}$ on the challenge query x^*. We now analyze the probability that $\mathcal{B}$'s guess was correct. Let δ' denote $\mathcal{B}$'s output and let δ denote whether T is an MDDH tuple or not, $\delta, \delta' \in \{0,1\}$. Now

$$\begin{aligned}\Pr[\delta' = \delta] &= \Pr[\delta' = \delta | \mathsf{abort}] \Pr[\mathsf{abort}] + \Pr[\delta' = \delta | \overline{\mathsf{abort}}] \Pr[\overline{\mathsf{abort}}] \\ &= \frac{1}{2}(1 - 2^{-n}) + \Pr[\delta' = \delta | \overline{\mathsf{abort}}] \cdot (2^{-n}) \\ &= \frac{1}{2}(1 - 2^{-n}) + \left(\frac{1}{2} + \epsilon\right) \cdot (2^{-n}) = \frac{1}{2} + \epsilon \cdot (2^{-n})\end{aligned}$$

The set of equations shows that the advantage of $\mathcal{B}$ is $\epsilon(\lambda)/2^n$. This completes the proof of the theorem, which establishes the pseudorandomness property of the construction. Hence, the constrained PRF construction for the circuit-predicate case is secure under the κ-MDDH assumption.

Removing the restrictions. The restriction that $\texttt{GateType}(n+q) = \text{OR}$ enables us to set randomness as we do in the scheme above. But this restriction can be easily removed by setting the randomness corresponding to the last level of OR gates (or the input wires in case there is no OR gate in the circuit) appropriately so that r_{n+q} ends up being u.

The restriction that a layer of gates cannot follow another layer of the same type of gates can also be overcome. The case of several consecutive layers of OR gates poses no threat since we move up one level in the multilinear maps for layers of OR gates and hence the current proof method works as is. The case of several consecutive layers of AND gates can be handled by even more careful choices of the randomness a_w and b_w. When we had only one layer of AND gate (before a layer of OR gates), for an OR gate at $\texttt{OR-depth}\ j$, we set a_w to be either $\mathbf{1} \cdot c_{n+j} + \psi_w$ or $\frac{\mathbf{1}}{\mathbf{2}} \cdot c_{n+j} + \psi_w$ depending on whether $r_{A(w)} = \mathbf{1} \cdot c_{n+1} \cdot \ldots \cdot c_{n+j-1} + \eta_{A(A(w))} + \eta_{B(A(w))}$ or $r_{A(w)} = \mathbf{2} \cdot c_{n+1} \cdot \ldots \cdot c_{n+j-1} + \eta_{A(A(w))} + \eta_{B(A(w))}$. Similarly, we set b_w in accordance with $r_{B(w)}$. Now, when there are more than one layers of AND gates consecutively, for an OR gate at $\texttt{OR-depth}\ j$ just after these AND gates, we set a_w (resp. $b(w)$) to be $\frac{1}{k} c_{n+j} + \psi_w$ where k is the coefficient of $c_{n+1} \cdot \ldots \cdot c_{n+j-1}$ in $r_{A(w)}$ (resp. $r_{B(w)}$). We present an illustration of this technique in Appendix A.

Regarding the first assumption, any layered circuit can be trivially converted into a "homogeneous" layered circuit by "splitting" each layer in the layered circuit into two layers: one with only AND gates and the other with only OR gates. This doubles the depth of the circuit. But if we are a bit more careful and do the splitting such that the odd layers are split into an AND-layer followed by an OR-layer and the even layers are split into an OR-layer followed by an AND-layer, the resulting circuit will have layers of the form (AND-OR)-(OR-AND)-(AND-OR)-$\cdots$. Now, we can merge the consecutive OR layers into a single OR layer (because our scheme supports gates with arbitrary fan-in) with just a polynomial increase in the number of wires. So, we can convert a layered circuit of depth d into a layered circuit with each layer consisting of only AND or OR gates with depth $d+1$ but with the OR-depth of the circuit being $d/2$ now. So even in the worst case we get improvements in parameters using our scheme.

4 A Free-OR Circuit-predicate Construction

In this section, we show how to construct a constrained random function for polynomial size circuit predicates of a specific form, without giving any keys for the OR gates. Once again, we base our construction on multilinear maps and on the κ-MDDH assumption; however κ in our construction will only depend on n (the size of the input to the PRF) and now, the AND-depth of the circuit (informally, this is the maximum number of AND gates from input wires to the output wire along any path). Once again, the starting point of our construction is the constrained PRF construction of Boneh and Waters [BW13] which is based on the attribute-based encryption construction for circuits [GGH$^+$13c]. We restrict the class of boolean circuits to be of a specific form. We assume layered circuits and that all gates in a particular layer are of the same type

(either AND or OR). We assume that a layer of gates is not followed by another layer of the same type of gates. We also assume that all AND gates have a fanout of 1[9].

We introduce here a "gadget" which we call a "FANOUT-gate". This is done in order to deal with OR gates in the circuit that have a fanout greater than 1. To this end, we assume that a FANOUT-gate is placed just after the OR gate under consideration. We view such OR gates also to have a fanout of 1 and without loss of generality assume that the FANOUT-gate alone has a fanout greater than 1. However, we do not treat the FANOUT-gate while calculating the total depth of the circuit, etc. It is merely a construct which allows us to deal only with OR gates having fanout 1.

4.1 Construction

The setup and the PRF construction is identical to the construction in Section 3. We now outline the constrain and evaluate algorithms.

$F.\mathsf{Constrain}(k, f = (n, q, A, B, \mathtt{GateType}))$**:**

The constrain algorithm takes as input the key k and a circuit description f. The circuit has $n + q$ wires with n input wires, q gates and the wire $n + q$ designated as the output wire. Assume that all gates have fanout 1 and that FANOUT-gates have been inserted at places where the gates have a fanout greater than 1.

To generate a constrained key k_f, the key generation algorithm sets $r_{n+q} = u$, where we think of the random value r_w as being associated with the wire w. Hence, in notation, if a gate w has fanout greater than 1, then, notation-wise, r_w would have mutliple values: one associated with each of the fanout wires of the FANOUT-gate and one associated with the wire leading out of the gate w itself. We introduce notation for the same below.

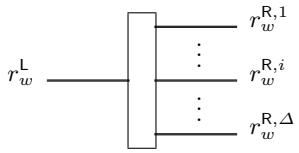

Fig. 1. FANOUT-gate

Consider a FANOUT-gate placed after wire w, as shown in Figure 1. We denote by r_w^{L} the randomness on the wire going as input to the FANOUT-gate (the actual output wire of the gate under consideration) and by $r_w^{\mathsf{R},i}$ the randomness on the ith fanout wire of the FANOUT-gate (there would be as many of these as the fanout of the gate w), where $i \in [\Delta]$ and Δ is the fanout of the wire w.

We now describe how the randomness for each wire is set. For each $w \in [n+q]\backslash[n]$, if $\mathtt{GateType}(w) = \text{OR}$, it sets $r_{A(w)} = r_{B(w)} = r_w$, otherwise, it chooses $r_{A(w)}$ and $r_{B(w)}$ at random. The case of FANOUT-gates is handled as follows. Note that the above description already takes care of setting randomness on all the fanout wires of the FANOUT-gate. The randomness for the input wire to the FANOUT-gate (the output wire of the gate with fanout greater than 1) is chosen at random. Note that this completely describes how randomness on all wires in the circuit are chosen.

The first part of the constrained key is given out as simply all $D_{i,\beta}$ for $i \in [n]$ and $\beta \in \{0, 1\}$. Next, the algorithm generates key components. The structure of the key components depends on whether w is an input wire or an output of an AND gate. For OR gates, we do not need to give out any keys, hence the name

[9] This can always be ensured for circuits that have alternating AND and OR layers. Suppose there is an AND gate with fanout $\Delta > 1$. We simply replace it with Δ AND gates having the same inputs and now we have Δ wires with the required output as before. Note that this process would have forced us to make the fanout of gates driving the AND gate to be Δ times as large, but since a gate driving an AND gate would only be an OR gate by our imposed circuit structure, this blows up the size of the circuit by only a polynomial factor.

Free-OR. But, we also need to give out special key components for the FANOUT-gates. The key components in each case are described below.

– *Input wire*
By convention, if $w \in [n]$, then it corresponds to the w-th input. The key component is:

$$K_w = g^{r_w d_{w,1}}$$

– *AND gate*
Suppose that $w \in \mathsf{Gates}$ and that $\mathtt{GateType}(w) = \text{AND}$. In addition, let $j = \mathtt{AND\text{-}depth}(w)$. The algorithm chooses random $a_w, b_w \in \mathbb{Z}_p$. Then, the algorithm creates key components:

$$K_{w,1} = g^{a_w}, K_{w,2} = g^{b_w}, K_{w,3} = g_{j-1}^{r_w - a_w \cdot r_{A(w)} - b_w \cdot r_{B(w)}}$$

– *FANOUT-gate*
Suppose that $w \in \mathsf{Gates}$, $\mathtt{GateType}(w) = \text{OR}$ and that the fanout of w is greater than 1. In addition, let $j = \mathtt{AND\text{-}depth}(w)$. In this case, a FANOUT-gate would have been placed after w. Let r_w^{L} denote the randomness on the wire going as input to the FANOUT-gate (the actual output wire of the gate under consideration) and let $r_w^{\mathsf{R},i}$ denote the randomness on the ith fanout wire of the FANOUT-gate (there would be as many of these as the fanout of the gate w). The keys given out are:

$$K_{w,w',i} = g_{j-1}^{\left(r_w^{\mathsf{R},i} - r_w^{\mathsf{L}}\right)}$$

for all $i \in [\Delta]$, where Δ is the fanout of the gate w.

The constrained key k_f consists of all these key components along with $\{D_{i,\beta}\}$ for $i \in [n]$ and $\beta \in \{0,1\}$.

$F.\mathsf{Evaluate}(k_f, x)$**:**
The evaluate algorithm takes as input a constrained key k_f for the circuit $f = (n, q, A, B, \mathtt{GateType})$ and an input $x \in \{0,1\}^n$. The algorithm first checks that $f(x) = 1$, and if not, it aborts.

Consider the wire w at AND-depth j. If $f_w(x) = 1$, then, the algorithm computes $E_w = g_{n+j-1}^{r_w \prod_{m \in [n]} d_{m,x_m}}$. If $f_w(x) = 0$, then nothing needs to be computed for that wire. The algorithm proceeds iteratively starting with computing E_1 and proceeds, in order, to compute E_{n+q}. Computing these values in order ensures that the computation on a lower-depth wire that evaluates to 1 will be defined before the computation for a higher-depth wire. Since $r_{n+q} = u$, $E_{n+q} = g_{n+\ell_{\mathsf{AND}}-1}^{u \prod_{m \in [n]} d_{m,x_m}}$.

We show how to compute E_w for all w where $f_w(x) = 1$, case-wise, according to whether the wire is an input, an OR gate, an AND gate or a fanout wire of a FANOUT-gate. Define $D = D(x) = g_n^{\prod_{m \in [n]} d_{m,x_m}}$, which is computable through n pairing operations.

– *Input wire*
By convention, if $w \in [n]$, then it corresponds to the w-th input. Suppose $f_w(x) = 1$. Through pairing operations, the algorithm computes $g_{n-1}^{\prod_{m \in [n] \setminus \{w\}} d_{m,x_m}}$. It then computes:

$$E_w = e\left(K_w, g_{n-1}^{\prod_{m \in [n] \setminus \{w\}} d_{m,x_m}}\right) = g_n^{r_w \prod_{m \in [n]} d_{m,x_m}}$$

– *OR gate*
Consider a wire $w \in \mathsf{Gates}$ with $\mathtt{GateType}(w) = \text{OR}$. The computation is performed if $f_w(x) = 1$. Note that in this case, at least one of $f_{A(w)}(x)$ and $f_{B(w)}(x)$ must be 1. Hence, we must have been able to evaluate at least one of $E_{A(w)}$ and $E_{B(w)}$. Since, for an OR gate, $r_{A(w)} = r_{B(w)} = r_w$, we have $E_w = E_{A(w)} = E_{B(w)}$, which can now be computed.

– *AND gate*
Consider a wire $w \in \mathsf{Gates}$ with $\mathtt{GateType}(w) = \text{AND}$. In addition, let $j = \mathtt{AND\text{-}depth}(w)$. The computation is performed if $f_w(x) = 1$. Note that in this case, both $f_{A(w)}(x)$ and $f_{B(w)}(x)$ must be 1. The algorithm computes:

$$\begin{aligned} E_w &= e(E_{A(w)}, K_{w,1}) \cdot e(E_{B(w)}, K_{w,2}) \cdot e(K_{w,3}, D) \\ &= e\left(g_{n+j-2}^{r_{A(w)} \prod_{m\in[n]} d_{m,x_m}}, g^{a_w}\right) \cdot e\left(g_{n+j-2}^{r_{B(w)} \prod_{m\in[n]} d_{m,x_m}}, g^{b_w}\right) \cdot \\ &\quad e\left(g_{j-1}^{r_{A(w)} - a_w \cdot r_{A(w)} - b_w \cdot r_{B(w)}}, g_n^{u \prod_{m\in[n]} d_{m,x_m}}\right) \\ &= g_{n+j-1}^{r_w \prod_{m\in[n]} d_{m,x_m}} \end{aligned}$$

– *FANOUT-gate*
Let r_w^{L} denote the randomness on the wire going as input to the FANOUT-gate (the actual output wire of the gate under consideration) and let $r_w^{\mathsf{R},i}$ denote the randomness on the ith fanout wire of the FANOUT-gate (there would be as many of these as the fanout of the gate w). The computation is performed if $f_w(x) = 1$. In coherence with the previous notation, we define the quantities E_w^{L} and $E_w^{\mathsf{R},i}$. Note that the E_w^{L} would have been computed. It then computes:

$$E_w^{\mathsf{R},i} = e\left(K_{w,w',i}, D\right) \cdot E_w^{\mathsf{L}} = g_{n+j-1}^{r_w^{\mathsf{R},i} \prod_{m\in[n]} d_{m,x_m}}$$

The procedures above are evaluated in order for all w for which $f_w(x) = 1$. Thus, the algorithm computes $E_{n+q} = g_{n+\ell-1}^{u \prod_{m\in[n]} d_{m,x_m}} = F(k, x)$.

5 Combining the Free-AND and Free-OR Techniques

In this section, we show that for the case of $\mathbf{NC}^1$, we can indeed combine the Free-AND and Free-OR techniques to obtain a construction that has Free-ANDs and Free-ORs. While the main reason that the technique works is that for $\mathbf{NC}^1$ circuits we can consider only boolean formulas, proving that our construction is secure is non-trivial (and different from the case of ABE).

5.1 An $\mathbf{NC}^1$-predicate Construction

We construct a constrained PRF for arbitrary $\mathbf{NC}^1$ circuit predicates, without giving any keys for AND as well as OR gates. Again, we base our construction on the κ-MDDH assumption; however κ in our construction will only depend on n (the size of the input to the PRF) and **not** on the circuit in any way. We will be dealing with circuits of the form described in Section 2.3.

5.2 Construction

$F.\mathsf{Setup}(1^\lambda, 1^n)$**:**
The setup algorithm that defines the master secret key and the PRF is identical to the setup algorithm from Section 3 with $\kappa = n$ instead of $n + \ell_{\mathsf{OR}} - 1$.

$F.\mathsf{Constrain}(k, f = (n, q, A, B, \mathtt{GateType}))$**:**
The algorithm sets $r_{n+q} = u$. For each $w \in [n+q]\backslash[n]$, if $\mathtt{GateType}(w) = \text{OR}$, it sets $r_{A(w)} = r_{B(w)} = r_w$, otherwise, it chooses $r_{A(w)}$ at random and sets $r_{B(w)} = r_w - r_{A(w)}$. Since the fanout of all gates is 1, for any wire $w \in [n+q]\backslash[n]$, r_w would have been uniquely set. However, since the same inputs may be re-used in multiple gates, for any wire $w \in [n]$, r_w may have multiple values (as many as the fanout of the input wire), i.e., different randomness values for each use of the input wire (to different gates). Note that this procedure sets randomness on all wires in the circuit. The first part of the constrained key (k_f) is given out as simply

all $D_{i,\beta}$ for $i \in [n]$ and $\beta \in \{0,1\}$. The remaining key components are: $K_{w,i} = g^{r_{w,i} d_{w,1}}, \forall i \in [\Delta]$, where Δ is the fanout of the input wire w.

$F.\mathsf{Evaluate}(k_f, x)$**:**
The evaluate algorithm takes as input a constrained key k_f and an input $x \in \{0,1\}^n$. The algorithm first checks that $f(x) = 1$, and if not, it aborts. Consider the wire w. If $f_w(x) = 1$, then, we show how to compute[10] $E_w = g_n^{r_w \prod_{m\in[n]} d_{m,x_m}}$, case-wise, according to whether the wire is an input, an OR gate or an AND gate.

- *Input wire.* Through pairing operations, compute $g_{n-1}^{\prod_{m\in[n]\setminus\{w\}} d_{m,x_m}}$. Then compute: $E_{w,i} = e\left(K_{w,i}, g_{n-1}^{\prod_{m\in[n]\setminus\{w\}} d_{m,x_m}}\right) = g_n^{r_{w,i} \prod_{m\in[n]} d_{m,x_m}} \forall i \in [\Delta]$, where Δ is the fanout of the input wire w.
- *OR gate.* In this case, at least one of $f_{A(w)}(x)$ and $f_{B(w)}(x)$ must be 1. Hence, we can evaluate at least one of $E_{A(w)}$ and $E_{B(w)}$. Since, for an OR gate, $r_{A(w)} = r_{B(w)} = r_w$, $E_w = E_{A(w)} = E_{B(w)}$, can now be computed.
- *AND gate.* In this case, $f_{A(w)}(x) = f_{B(w)}(x) = 1$. The algorithm computes:

$$E_w = E_{A(w)} \cdot E_{B(w)} = g_n^{r_{A(w)} \prod_{m\in[n]} d_{m,x_m}} \cdot g_n^{r_{B(w)} \prod_{m\in[n]} d_{m,x_m}} = g_n^{r_w \prod_{m\in[n]} d_{m,x_m}}$$

The procedures above are evaluated, in order, for all w for which $f_w(x) = 1$. Thus, the algorithm computes $E_{n+q} = g_n^{u \prod_{m\in[n]} d_{m,x_m}} = F(k,x)$.

5.3 Proof of Pseudorandomness

The correctness of the constrained PRF is verifiable in a straightforward manner. To show pseudorandomness, given an algorithm $\mathcal{A}$ that breaks security of the constrained PRF, we will construct algorithm $\mathcal{B}$ that breaks security of the $\kappa = n-$MDDH assumption. $\mathcal{B}$ receives a $\kappa-$MDDH challenge consisting of the group sequence description $\mathbb{G}$ and $g = g_1, g^{c_1}, \ldots, g^{c_{\kappa+1}}$ along with T, where T is either $g_\kappa^{\prod_{m\in[\kappa+1]} c_m}$ or a random group element in $\mathbb{G}_\kappa$. The security proof is in the selective security model (where the adversary commits to the challenge input x^* at the beginning of the game). To get full security, the proof will use the standard complexity leveraging technique of guessing the challenge x^*; this guess will cause a loss of a $1/2^n$-factor in the reduction. We formally show:

Theorem 2. *If there exists a PPT adversary $\mathcal{A}$ that breaks the pseudorandomness property of our $\mathbf{NC}^1$-predicate construction for n-bit inputs with advantage $\epsilon(\lambda)$, then there exists a PPT algorithm $\mathcal{B}$ that breaks the $\kappa = n-$Multilinear Decisional Diffie-Hellman assumption with advantage $\epsilon(\lambda)/2^n$.*

Proof. The algorithm $\mathcal{B}$ first receives a $\kappa = n-$MDDH challenge consisting of the group sequence description $\mathbb{G}$ and $g = g_1, g^{c_1}, \ldots, g^{c_{\kappa+1}}$ along with T, where T is either $g_\kappa^{\prod_{m\in[\kappa+1]} c_m}$ or a random group element in $\mathbb{G}_\kappa$.

Setup:
It chooses an $x^* \in \{0,1\}^n$ uniformly at random. Next, it chooses random $z_1, \ldots, z_n \in \mathbb{Z}_p$ and sets $D_{m,\beta} = g^{c_m}$ if $x^*_m = \beta$ and g^{z_m} otherwise, for $m \in [n]$ and $\beta \in \{0,1\}$. It then implicitly sets $u = c_{n+1}$. The setup is executed as in the construction.

Constrain:
Suppose a query is made for a secret key for a circuit $f = (n, q, A, B, \mathtt{GateType})$. If $f(x^*) = 1$, then $\mathcal{B}$ aborts.

[10] For input wires $w \in [n]$, we have $E_{w,i} = g_n^{r_{w,i} \prod_{m\in[n]} d_{m,x_m}}$ for all $i \in [\Delta]$, where Δ is the fanout of the input wire w. This feature has been present in our Free-OR construction as well. We pay attention to it specifically in this construction because of the absence of fanout for any wire other than the input wires.

Otherwise, $\mathcal{B}$ sets the randomness on each wire in the circuit in the following way. It sets, for the output wire $w = n + q$, $r_w = u = c_{n+1}$. For each $w \in [n+q] \backslash [n]$, if $\texttt{GateType}(w) = \text{OR}$, it sets $r_{A(w)} = r_{B(w)} = r_w$. Suppose $\texttt{GateType}(w) = \text{AND}$. If $f_w(x^*) = 1$, then $f_{A(w)}(x^*) = f_{B(w)}(x^*) = 1$ and $\mathcal{B}$ chooses $r_{A(w)}$ at random and sets $r_{B(w)} = r_w - r_{A(w)}$. Suppose $f_w(x^*) = 0$. Then we know that at least one of $f_{A(w)}(x^*)$ and $f_{B(w)}(x^*)$ must be zero. If $f_{A(w)}(x^*) = 0$, it chooses $r_{B(w)}$ at random and sets $r_{A(w)} = r_w - r_{B(w)}$, while if $f_{A(w)}(x^*) = 1$ and hence $f_{B(w)}(x^*) = 0$, it chooses $r_{A(w)}$ at random and sets $r_{B(w)} = r_w - r_{A(w)}$. As we shall see later, such a choice of randomness is critical for the security proof. Since the fanout of all gates is 1, for any wire $w \in [n+q] \backslash [n]$, r_w would have been uniquely set. However, since the same inputs may be re-used in multiple gates, for any wire $w \in [n]$, r_w may have multiple values (as many as the fanout of the input wire), i.e., different randomness values for each use of the input wire (to different gates), which we denote by $r_{w,i}$ for all $i \in [\Delta]$, where Δ is the fanout of the input wire w. Note that this procedure sets randomness on all wires in the circuit.

To show that $\mathcal{B}$ can indeed compute all the key components, our proof will follow a similar structure to the Free-OR case (Section 4). We shall prove that for all wires in the circuit, $\mathcal{B}$ can compute g^{r_w}. To prove this, we shall prove the above statement, both when the wire w is such that $f_w(x^*) = 1$ (Lemma 2), and when the wire w is such that $f_w(x^*) = 0$ (Lemma 3). To prove Lemma 2, we shall first prove the following fact (Lemma 1): consider all wires in the circuit that evaluate to 1 on x^* and consider those wires among these that have maximum total depth; then, these wires must all be input wires to AND gates.

Lemma 1. *Define:*

- $S_1 = \{w : w \in [n+q] \wedge f_w(x^*) = 1\}$
- $S_1^{\textsf{max-tot-depth}} = \{w : w \in S_1 \wedge \texttt{tot-depth}(w) \geq \texttt{tot-depth}(w') \ \forall w' \in S_1\}$

Then w is an input wire to an AND gate $\forall w \in S_1^{\textsf{max-tot-depth}}$.

Proof. This fact is very easy to easy. Clearly, $w \neq n + q$, since $f_{n+q}(x^*) = 0$ while $f_w(x^*) = 1$. Hence there exist layers of gates after the one containing w. Suppose w is an input wire to an OR gate. Since $f_w(x^*) = 1$, for some OR gate w' in the next layer of gates, $f_{w'}(x^*) = 1$. Hence, $\exists w' \in S_1$ such that $\texttt{tot-depth}(w) < \texttt{tot-depth}(w')$ which contradicts the fact that $w \in S_1^{\textsf{max-tot-depth}}$.

Lemma 2. *For any wire $w \in [n+q]$, if $f_w(x^*) = 1$, then r_w is known.*

Proof. We prove this by observing the randomness we have set on each wire, from the output wire to the input wires. From Lemma 1, we know that the first such wire we would see would be an input to an AND gate. For an input wire $A(w)$, of an AND gate, satisfying $f_{A(w)}(x^*) = 1$, first consider the case when $f_w(x^*) = 1$[11]. In this case, $\mathcal{B}$ explicitly chooses all random values associated with this gate and hence $\mathcal{B}$ chose $r_{A(w)}$. When $f_w(x^*) = 0$, note that $\mathcal{B}$ carefully chose the randomness on the input wires which may potentially evaluate to 1 on x^* at random (and set the value on the other input wire $B(w)$ based on this). Hence, if $f_{A(w)}(x^*) = 1$, $r_{A(w)}$ is known to $\mathcal{B}$. This forms the base case for the induction. Now, consider any other wire $A(w)$ such that $f_{A(w)} = 1$. Now, if $A(w)$ were an input to an AND gate, then by the same argument as above, $r_{A(w)}$ is known to $\mathcal{B}$. Suppose, $A(w)$ were an input to an OR gate w and $f_{A(w)}(x^*) = 1$, then $f_w(x^*) = 1$. By the induction hypothesis, r_w is known. We know that since w is an OR gate, $r_{A(w)} = r_w$ and hence $r_{A(w)}$ is known. This completes the proof.

Lemma 3. *For any wire $w \in [n+q]$, if $f_w(x^*) = 0$, then g^{r_w} is known.*

Proof. We can prove this by observing the randomness we have set on each wire, from the output wire to the input wires. The statement is true for the output wire $w = n + q$, since $g^{c_{n+1}}$ is known. This forms the base case. We can now argue inductively.

[11] It is true that the first such wire when we go from output to input level would be an AND gate with $f_w(x^*) = 0$. However, the discussion on the case of $f_w(x^*) = 1$ is more a general one for all AND gates in the circuit.

– Case 1: If w is an input to an OR gate w', then $r_w = r_{w'}$. If $f_{w'}(x^*) = 1$, then by Lemma 2, $r_{w'}$ is known and hence g^{r_w} is known. If $f_{w'}(x^*) = 0$, then by the induction hypothesis, $g^{r_{w'}}$ is known and hence g^{r_w} is known.
– Case 2: If w is an input to an AND gate w', then $f_{w'}(x^*) = 0$. Now, by the induction hypothesis, $g^{r_{w'}}$ is known. If $w = A(w')$, then $r_{B(w')}$ was chosen at random and is known, and hence $g^{r_w} = g^{r_{w'} - r_{B(w')}}$ is known. Suppose $w = B(w')$. If $f_{A(w')}(x^*) = 0$, r_w was chosen at random and is known, and hence g^{r_w} is known. If $f_{A(w')}(x^*) = 1$, then $r_{A(w')}$ was chosen at random and is known, and hence $g^{r_w} = g^{r_{w'} - r_{A(w')}}$ is known.

Finally, $\mathcal{B}$ generates key components for input wires $w \in [n]$. By convention, if $w \in [n]$, then it corresponds to the w-th input. If $x^*_w = 1$, then $r_{w,i}$ is known, from Lemma 2, for all $i \in [\Delta]$, where Δ is the fanout of the input wire w. The key components are: $K_{w,i} = (D_{w,1})^{r_{w,i}} = g^{r_{w,i} d_{w,1}}$, for all $i \in [\Delta]$. If $x^*_w = 0$, then $g^{r_{w,i}}$ is known, from Lemma 3, for all $i \in [\Delta]$. The key components are: $K_{w,i} = (g^{r_{w,i}})^{z_w} = g^{r_{w,i} d_{w,1}}$, for all $i \in [\Delta]$.

Evaluate:
Suppose a query is made for a secret key for an input $x \in \{0,1\}^n$. If $x = x^*$, then $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ identifies an arbitrary t such that $x_t \neq x^*_t$. Through pairing of $D_{m,x_m} \forall m \in [n] \backslash \{t\}$, it computes $g_{n-1}^{\prod_{m \in [n] \backslash \{t\}} d_{m,x_m}}$ and raises it to $d_{t,x_t} = z_t$ to get $H = g_{n-1}^{\prod_{m \in [n]} d_{m,x_m}}$. Finally, it computes $H' = e(U, H) = g_n^{u \prod_{m \in [n]} d_{m,x_m}} = F(k, x)$ and outputs it. Eventually, $\mathcal{A}$ will issue a challenge input $\tilde{x}$. If $\tilde{x} = x^*$, $\mathcal{B}$ will return the value T and output the same bit as $\mathcal{A}$ does as its guess. If $\tilde{x} \neq x^*$, $\mathcal{B}$ outputs a random bit as its guess.

This completes the description of the adversary $\mathcal{B}$. We first note that in the case where T is part of a MDDH tuple, the real game and game executed by $\mathcal{B}$ have the identical distribution. Secondly, in both cases (i.e., whether or not T is part of the MDDH tuple), as long as $\mathcal{B}$ does not abort, once again, the real game and game executed by $\mathcal{B}$ have the identical distribution, except for the output of $\mathcal{B}$ on the challenge query x^*. Similar to the analysis in Section 3, the probability that $\mathcal{B}$'s guess was correct can be shown to be $\epsilon(\lambda)/2^n$.

6 From Bit-fixing PRFs to NC^1 PRFs

In this section, we show that from any constrained PRF scheme supporting bit-fixing predicates that has certain additive homomorphic properties (let this be F_{bf}), we can construct a constrained PRF scheme supporting $\mathbf{NC}^1$ circuit predicates ($\mathsf{F}_{\mathsf{NC1}}$) in a black-box manner. We will be dealing with circuits of the form described in Section 2.3. It is sufficient if the PRF is able to fix a single bit to just one of the possibilities (i.e., either fixing the bits only to 0 or only to 1). The homomorphic properties that we require from the bit-fixing scheme are:

1. The PRF must have an additive key-homomorphism property. In other words, there exists a public algorithm $\mathsf{F}_{\mathsf{bf}}.\mathsf{KeyEval}$, such that, for all $k_1, k_2 \in \mathcal{K}$, $\mathsf{F}_{\mathsf{bf}}.\mathsf{KeyEval}$ outputs $\mathsf{F}_{\mathsf{bf}}(k_1 + k_2, x)$ on inputs $\mathsf{F}_{\mathsf{bf}}(k_1, x)$ and $\mathsf{F}_{\mathsf{bf}}(k_2, x)$.
2. Let $\mathsf{F}_{\mathsf{bf}}.\mathsf{Constrain}(k, i)$ be the constrain algorithm that takes in a key and the position of the bit to be fixed to 1.[12] An additive key-homomorphism property should also exist among the constrained keys, that is, there exists a public algorithm, $\mathsf{F}_{\mathsf{bf}}.\mathsf{AddKeys}$, such that[13], for all $k_1, k_2 \in \mathcal{K}$ and index i,

$$\mathsf{F}_{\mathsf{bf}}.\mathsf{AddKeys}(\mathsf{F}_{\mathsf{bf}}.\mathsf{Constrain}(k_1, i), \mathsf{F}_{\mathsf{bf}}.\mathsf{Constrain}(k_2, i)) = \mathsf{F}_{\mathsf{bf}}.\mathsf{Constrain}(k_1 + k_2, i)$$

[12] By symmetry, the construction also works if the constrain algorithm fixes a bit to 0.
[13] We note here that $\mathsf{F}_{\mathsf{bf}}.\mathsf{Constrain}(k, i)$ could, in general, be a randomized algorithm and in this case, we require the distributions on the left and the right of the equality to be computationally indistinguishable. For ease of exposition, we assume that $\mathsf{F}_{\mathsf{bf}}.\mathsf{Constrain}(k, i)$ is deterministic and state our results accordingly.

6.1 Construction

We follow the same template as in our $\mathbf{NC}^1$-predicate construction in Section 5.1. We observe that the component $K_{w,i}$ at the input level can be replaced with a constrained key from any bit-fixing scheme which satisfies the properties mentioned above. $\mathsf{F}_{\mathsf{bf}}, \mathsf{F}_{\mathsf{NC1}}$ denote the bit-fixing and $\mathbf{NC}^1$ schemes respectively.

$\mathsf{F}_{\mathsf{NC1}}.\mathsf{Setup}(1^\lambda, 1^n)$**:**
The setup algorithm runs $\mathsf{F}_{\mathsf{bf}}.\mathsf{Setup}(1^\lambda, 1^n)$ to get the PRF F_{bf} and key k. It sets the key as k. The keyed pseudo-random function is defined as $\mathsf{F}_{\mathsf{bf}}(k, x)$.

$\mathsf{F}_{\mathsf{NC1}}.\mathsf{Constrain}(k, f = (n, q, A, B, \mathtt{GateType}))$**:**
The constrain algorithm sets up randomness on the wires of the circuit using the procedure in the construction in Section 5.1 and computes key components for the input wires as $K_w = \mathsf{F}_{\mathsf{bf}}.\mathsf{Constrain}(r_w, w)$[14]. The constrained key k_f consists of all these key components.

$\mathsf{F}_{\mathsf{NC1}}.\mathsf{Evaluate}(k_f, x)$**:**
The algorithm first checks that $f(x) = 1$, and if not, it aborts. As in the construction in Section 5.1, for every wire w, if $f_w(x) = 1$, then, the algorithm computes $\mathsf{F}_{\mathsf{bf}}(r_w, x)$. The algorithm proceeds iteratively and computes $\mathsf{F}_{\mathsf{bf}}(r_{n+q}, x) = \mathsf{F}_{\mathsf{bf}}(k, x)$. $\mathsf{F}_{\mathsf{bf}}(r_w, x)$ can be computed, case-wise, according to whether the wire is an input, an OR gate or an AND gate.

- *Input wire*
  If $f_w(x) = 1$, it computes $\mathsf{F}_{\mathsf{bf}}(r_w, x) = \mathsf{F}_{\mathsf{bf}}.\mathsf{Eval}(K_w, x)$.
- *OR gate*
  If $f_w(x) = 1$, at least one of $f_{A(w)}(x)$ and $f_{B(w)}(x)$ must be 1. Hence, we must have been able to evaluate at least one of $\mathsf{F}_{\mathsf{bf}}(r_{A(w)}, x)$ and $\mathsf{F}_{\mathsf{bf}}(r_{B(w)}, x)$. Since, $r_{A(w)} = r_{B(w)} = r_w$, $\mathsf{F}_{\mathsf{bf}}(r_w, x) = \mathsf{F}_{\mathsf{bf}}(r_{A(w)}, x) = \mathsf{F}_{\mathsf{bf}}(r_{B(w)}, x)$, which can be computed.
- *AND gate*
  If $f_w(x) = 1$, $f_{A(w)}(x) = f_{B(w)}(x) = 1$. Hence, we must have been able to evaluate both $\mathsf{F}_{\mathsf{bf}}(r_{A(w)}, x)$ and $\mathsf{F}_{\mathsf{bf}}(r_{B(w)}, x)$. The algorithm computes $\mathsf{F}_{\mathsf{bf}}(r_w, x) = \mathsf{F}_{\mathsf{bf}}.\mathsf{KeyEval}(\mathsf{F}_{\mathsf{bf}}(r_{A(w)}, x), \mathsf{F}_{\mathsf{bf}}(r_{B(w)}x))$, since, $r_{A(w)} + r_{B(w)} = r_w$.

The procedures above are evaluated, in order, for all w for which $f_w(x) = 1$. Thus, the algorithm computes $\mathsf{F}_{\mathsf{bf}}(r_{n+q}, x) = \mathsf{F}_{\mathsf{bf}}(k, x)$.

6.2 Proof of Pseudorandomness

The correctness of the scheme is straightforward from the key-homomorphism property of the bit-fixing PRF scheme. We now prove the security.

Theorem 3. *If there exists a PPT adversary $\mathcal{A}$ that breaks the selective security of our construction for n-bit inputs supporting $\mathbf{NC}^1$-predicates with an advantage $\epsilon(\lambda)$, then there exists a PPT algorithm $\mathcal{B}$ that breaks the selective security of the underlying bit-fixing predicate construction with the same advantage $\epsilon(\lambda)$.*

Proof. Let $\mathcal{A}$ be the adversary which breaks the selective security of our $\mathbf{NC}^1$ construction. We will construct an adversary $\mathcal{B}$ which will use $\mathcal{A}$ to break the selective security of the bit-fixing construction F_{bf}. Thus, $\mathcal{B}$ plays a dual role: one as an adversary in the security game breaking the bit-fixing construction and also as a challenger in the security game breaking the $\mathbf{NC}^1$ construction.

- First $\mathcal{A}$ provides its challenge x^* to $\mathcal{B}$ which in turn forwards it to its challenger. $\mathcal{B}$ receives the public parameters of the bit-fixing scheme from its challenger along with either $\mathsf{F}_{\mathsf{bf}}(k, x^*)$ or a random value which it forwards to $\mathcal{A}$. $\mathcal{B}$ is going to answer queries as though the PRF evaluated by the $\mathbf{NC}^1$ construction is the same as that evaluated by the bit-fixing construction F_{bf} used by the challenger. When $\mathcal{A}$ asks a query f to $\mathbf{NC}^1.\mathsf{Constrain}$ oracle with $f(x^*) = 0$, $\mathcal{B}$ follows a procedure similar to the one in Section 5.1.

[14] As in Section 5.1, the fanout of the input wires can be easily incorporated.

- $\mathcal{B}$ *carefully* sets the randomness on all wires in the circuit as in the proof in Section 5.1. By virtue of this careful setting, the same properties hold: for any wire $w \in [n+q]$, if $f_w(x^*) = 1$, then r_w is known, and if $f_w(x^*) = 0$, then r_w would either be known or of the form $k + \sum r$, where each r is known. Note that $r_{n+q} = k$ which is the key of PRF used by $\mathcal{B}$ as well as $\mathcal{B}$'s challenger.
- To give out keys for the input wires, $\mathcal{B}$ does the following. For those wires w with $f_w(x^*) = 1$, r_w is known and hence $\mathcal{B}$ obtains $K_w = \mathsf{F_{bf}.Constrain}(r_w, w)$ by running $\mathsf{F_{bf}.Constrain}(r_w, w)$ by itself. For wires w with $f_w(x^*) = 0$, if r_w is known, then $\mathcal{B}$ obtains $K_w = \mathsf{F_{bf}.Constrain}(r_w, w)$ by running $\mathsf{F_{bf}.Constrain}(r_w, w)$ by itself. Otherwise, r_w is of the form $k + \sum r$, where each r is known. For each r, $\mathcal{B}$ obtains $K'_{r,w} = \mathsf{F_{bf}.Constrain}(r, w)$ by running $\mathsf{F_{bf}.Constrain}(r, w)$ by itself. Through repeated use of $\mathsf{F_{bf}.AddKeys}$, and by virtue of the homomorphism property of the constrained keys, $\mathcal{B}$ obtains $K'_{\sum r,w} = \mathsf{F_{bf}.Constrain}\left(\sum r, w\right)$. $\mathcal{B}$ then queries its challenger for the constrained key fixing the wth bit, i.e., it obtains $K'_{k,w} = \mathsf{F_{bf}.Constrain}(k, w)$ by querying its challenger. Finally, through the use of $\mathsf{F_{bf}.AddKeys}\left(K'_{k,w}, K'_{\sum r,w}\right)$, $\mathcal{B}$ obtains $K_w = \mathsf{F_{bf}.Constrain}\left(r_w, w\right)$.
- When answering $\mathcal{A}$'s queries to $\mathbf{NC}^1.\mathsf{Constrain}$, it is important to note that $\mathcal{B}$ does not query for any predicate that allows it to evaluate $F(k, x^*)$ by itself. We achieve this because all queries by $\mathcal{B}$ to the challenger, $\mathsf{F_{bf}.Constrain}(k, w)$, fix the wth bit to 1, while if the query were made, $f_w(x^*) = 0$, i.e., the wth bit of x^* is 0.

– When $\mathcal{A}$ outputs a bit b', $\mathcal{B}$ outputs the same.

In the above game, if $\mathcal{A}$ breaks the selective security of the $\mathbf{NC}^1$ construction with an advantage of $\epsilon(\lambda)$ then $\mathcal{B}$ breaks the underlying bit-fixing construction with the same advantage.

References

BFP+15. A. Banerjee, G. Fuchsbauer, C. Peikert, K. Pietrzak, and S. Stevens. Key-homomorphic constrained pseudorandom functions. In *TCC(II)*, pages 31–60, 2015.

BGG+14. D. Boneh, C. Gentry, S. Gorbunov, S. Halevi, V. Nikolaenko, G. Segev, V. Vaikuntanathan, and D. Vinayagamurthy. Fully Key-Homomorphic Encryption, Arithmetic Circuit ABE and Compact Garbled Circuits. In *EUROCRYPT*, pages 533–556, 2014.

BGI14. E. Boyle, S. Goldwasser, and I. Ivan. Functional Signatures and Pseudorandom Functions. In *Public Key Cryptography*, pages 501–519, 2014.

BV15. Z. Brakerski and V. Vaikuntanathan. Constrained key-homomorphic prfs from standard lattice assumptions - or: How to secretly embed a circuit in your PRF. In *TCC(II)*, pages 1–30, 2015.

BW13. D. Boneh and B. Waters. Constrained Pseudorandom Functions and Their Applications. In *ASIACRYPT (2)*, pages 280–300, 2013.

CHL+15. Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehlé. Cryptanalysis of the multilinear map over the integers. In *EUROCRYPT I*, pages 3–12, 2015.

CLT13. Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In *CRYPTO I*, pages 476–493, 2013.

CLT15. Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. New multilinear maps over the integers. In *CRYPTO I*, pages 267–286, 2015.

CRV. N. Chandran, S. Raghuraman, and D. Vinayagamurthy. Constrained Pseudorandom Functions: Verifiable and Delegatable. Cryptology ePrint Archive, Report 2014/522.

FKPR14. G. Fuchsbauer, M. Konstantinov, K. Pietrzak, and V. Rao. Adaptive security of constrained prfs. In *ASIACRYPT*, pages 82–101, 2014.

Fuc14. G. Fuchsbauer. Constrained Verifiable Random Functions. In *SCN*, pages 95–114, 2014.

GGH13a. S. Garg, C. Gentry, and S. Halevi. Candidate Multilinear Maps from Ideal Lattices. In *EUROCRYPT*, pages 1–17, 2013.

GGH+13b. S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate Indistinguishability Obfuscation and Functional Encryption for all Circuits. In *FOCS*, pages 40–49, 2013.

GGH+13c. S. Garg, C. Gentry, S. Halevi, A. Sahai, and B. Waters. Attribute-Based Encryption for Circuits from Multilinear Maps. In *CRYPTO (2)*, pages 479–499, 2013.

GGH15. Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In *TCC II*, pages 498–527, 2015.

GGM86. O. Goldreich, S. Goldwasser, and S. Micali. How to Construct Random Functions. *J. ACM*, 33(4):792–807, 1986.

GPSW06. V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-Based Encryption for Fine-grained Access Control of Encrypted Data. In *ACM Conference on Computer and Communications Security*, pages 89–98, 2006.

GVW13. S. Gorbunov, V. Vaikuntanathan, and H. Wee. Attribute-Based Encryption for Circuits. In *STOC*, pages 545–554, 2013.

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

HKKW. D. Hofheinz, A. Kamath, V. Koppula, and B. Waters. Adaptively Secure Constrained Pseudorandom Functions. Cryptology ePrint Archive, Report 2014/720.

KPTZ13. A. Kiayias, S. Papadopoulos, N. Triandopoulos, and T. Zacharias. Delegatable Pseudorandom Functions and Applications. In *ACM Conference on Computer and Communications Security*, pages 669–684, 2013.

SW05. A. Sahai and B. Waters. Fuzzy Identity-Based Encryption. In *EUROCRYPT*, pages 457–473, 2005.

A Dealing with consecutive AND gate Layers

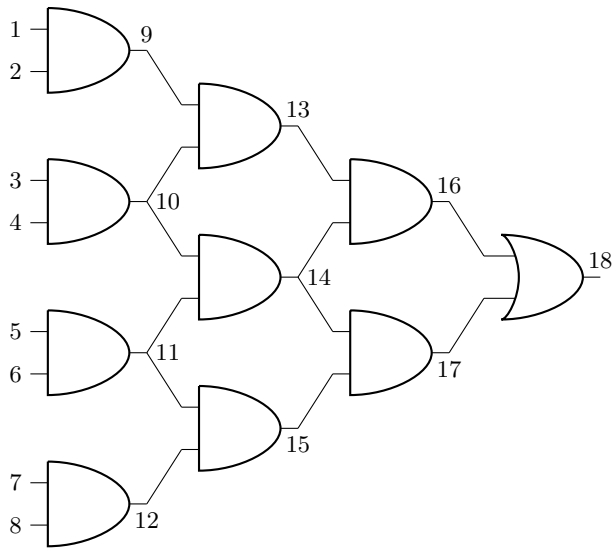

Fig. 2. Circuit with consecutive layers of AND gates

Consider the circuit shown in Figure 2. We present here a sketch of how the simulator would work for this circuit for choice $x^* = 01001111$ on which the circuit evaluates to 0. The wires are numbered as in Figure 2. The simulator sets randomness as described in the proof of Theorem 1:

- $r_w = c_{n+1} + \eta_w$ for $w \in \{1, 3, 4\}$; $r_w = \eta_w$ for $w \in \{2, 5, 6, 7, 8\}$;
- $r_9 = c_{n+1} + \eta_1 + \eta_2$, $r_{10} = 2c_{n+1} + \eta_3 + \eta_4$, $r_{11} = \eta_5 + \eta_6$, $r_{12} = \eta_7 + \eta_8$
- $r_{13} = 3c_{n+1} + \eta_1 + \eta_2$, $r_{14} = 2c_{n+1} + \eta_3 + \eta_4 + \eta_5 + \eta_6$, $r_{15} = \eta_5 + \eta_6 + \eta_7 + \eta_8$
- $r_{16} = 5c_{n+1} + \eta_1 + \eta_2 + \eta_3 + \eta_4 + \eta_5 + \eta_6$, $r_{17} = 2c_{n+1} + \eta_3 + \eta_4 + 2\eta_5 + 2\eta_6 + \eta_7 + \eta_8$
- $r_{18} = c_{n+1}c_{n+2}$

$\mathcal{B}$ computes the key components for the input wires as in the proof of Theorem 1. Key components need to be given out for the OR gate as well. To this end, $\mathcal{B}$ sets $a_{18} = \frac{1}{5}c_{n+2} + \psi_{18}$ and $b_{18} = \frac{1}{2}c_{n+2} + \phi_{18}$. Then, $\mathcal{B}$ creates key components:

$$K_{18,1} = g^{a_{18}}, K_{18,2} = g^{b_{18}}, K_{18,3} = g^{r_{18} - a_{18} \cdot r_{16}}, K_{18,4} = g^{r_{18} - b_{18} \cdot r_{17}}$$

As was shown in the proof of Theorem 1, it is easy to see that the components can indeed be computed.

When there is no OR gate following these layers of AND gates, we engineer the randomness coming into the first layer (which is either the input randomness or that from the layer of OR gates before these layers of AND gates) such that the output randomness ends up being $u = r_{n+q}$.

B Proof of pseudorandomness of the Free-OR Construction

The correctness of the constrained PRF is verifiable in a straightforward manner from the construction. The pseudorandomness property of the constrained PRF is proved ahead. The security proof is in the selective security model (where the adversary commits to the challenge input x^* at the beginning of the game). To get full security, the proof will use the standard complexity leveraging technique of guessing the challenge x^*; this guess will cause a loss of a $1/2^n$-factor in the reduction.

Theorem 4. *If there exists a PPT adversary $\mathcal{A}$ that breaks the pseudorandomness property of our circuit-predicate construction for n-bit inputs with advantage $\epsilon(\lambda)$, then there exists a PPT algorithm $\mathcal{B}$ that breaks the $\kappa = (n + \ell_{\mathsf{AND}} - 1)-$Multilinear Decisional Diffie-Hellman assumption with advantage $\epsilon(\lambda)/2^n$.*

Proof. The algorithm $\mathcal{B}$ first receives a $\kappa = (n + \ell_{\mathsf{AND}} - 1)-$MDDH challenge consisting of the group sequence description $\mathbb{G}$ and $g = g_1, g^{c_1}, \ldots, g^{c_{\kappa+1}}$ along with T, where T is either $g_\kappa^{\prod_{m\in[\kappa+1]} c_m}$ or a random group element in $\mathbb{G}_\kappa$.

Setup:
It chooses an $x^* \in \{0,1\}^n$ uniformly at random. Next, it chooses random $z_1, \ldots, z_n \in \mathbb{Z}_p$ and sets

$$D_{m,\beta} = \begin{cases} g^{c_m} & x^*_m = \beta \\ g^{z_m} & x^*_m \neq \beta \end{cases}$$

for $m \in [n]$ and $\beta \in \{0,1\}$. This corresponds to setting

$$d_{m,\beta} = \begin{cases} c_m & x^*_m = \beta \\ z_m & x^*_m \neq \beta \end{cases}$$

It then sets $u = c_{n+1} \cdot c_{n+2} \cdot \ldots \cdot c_{n+\ell_{\mathsf{AND}}}$. The setup is executed as in the construction.

Constrain:
Suppose a query is made for a secret key for a circuit $f = (n, q, A, B, \mathtt{GateType})$. If $f(x^*) = 1$, then $\mathcal{B}$ aborts.

Otherwise, $\mathcal{B}$ sets the randomness on each wire in the circuit in the following way. For the output wire $w = n+q$, it implicitly sets $r_w = u = c_{n+1} \cdot c_{n+2} \cdot \ldots \cdot c_{n+\ell_{\mathsf{AND}}}$. For each $w \in [n+q]\backslash[n]$, if $\mathtt{GateType}(w) = \mathrm{OR}$, it sets $r_{A(w)} = r_{B(w)} = r_w$. Suppose $\mathtt{GateType}(w) = \mathrm{AND}$. In addition, let $j = \mathtt{AND\text{-}depth}(w)$. If $f_{A(w)}(x^*) = 0$, it sets $r_{A(w)} = c_{n+1} \cdot \ldots \cdot c_{n+j-1} + \eta_{A(w)}$, while if $f_{A(w)}(x^*) = 1$, it sets $r_{A(w)} = \eta_{A(w)}$, where $\eta_{A(w)} \in \mathbb{Z}_p$ is a randomly chosen element. Similarly, if $f_{B(w)}(x^*) = 0$, it sets $r_{B(w)} = c_{n+1} \cdot \ldots \cdot c_{n+j-1} + \eta_{B(w)}$, while if $f_{B(w)}(x^*) = 1$, it sets $r_{B(w)} = \eta_{B(w)}$, where $\eta_{B(w)} \in \mathbb{Z}_p$ is a randomly chosen element. Suppose that $w \in \mathsf{Gates}$, $\mathtt{GateType}(w) = \mathrm{OR}$ and that the fanout of w is greater than 1. In addition, let $j = \mathtt{AND\text{-}depth}(w)$. In this case, a FANOUT-gate would have been placed after w. Let r_w^{L} denote the randomness on the wire going as input to the FANOUT-gate (the actual output wire of the gate under consideration) and let $r_w^{\mathsf{R},i}$ denote the randomness on the ith fanout wire of the FANOUT-gate (there would be as many of these as the fanout of the gate w). Since the process of setting the randomness on the wires inherently proceeds from the output gate to the inputs by description, $\mathcal{B}$ would have already set $r_w^{\mathsf{R},i}$ for all $i \in [\Delta]$, where Δ denotes the fanout of the gate. If $f_w(x^*) = 0$, it sets $r_w^{\mathsf{L}} = c_{n+1} \cdot \ldots \cdot c_{n+j} + \eta_w^{\mathsf{L}}$, while if $f_w(x^*) = 1$, it sets $r_w^{\mathsf{L}} = \eta_w^{\mathsf{L}}$, where $\eta_w^{\mathsf{L}} \in \mathbb{Z}_p$ is a randomly chosen element.

To show that $\mathcal{B}$ can indeed compute all the key components we shall prove (Lemma 7) that for all wires in the circuit, $\mathcal{B}$ can compute $g_j^{r_w}$, where j is $\mathtt{AND\text{-}depth}(w)$. To prove this, we shall prove the above statement, both when the wire w is such that $f_w(x^*) = 1$ (Lemma 5), and when the wire w is such that $f_w(x^*) = 0$ (Lemma 6). To prove Lemma 5, we shall first prove the following fact (Lemma 4): consider all wires in the circuit that evaluate to 1 on x^* and consider those wires among these that have maximum total depth; then, these wires must all be input wires to AND gates.

Lemma 4. *Define:*

- $S_1 = \{w : w \in [n+q] \wedge f_w(x^*) = 1\}$
- $S_1^{\textit{max-tot-depth}} = \{w : w \in S_1 \wedge \mathtt{tot\text{-}depth}(w) \geq \mathtt{tot\text{-}depth}(w')\ \forall w' \in S_1\}$

Then w is an input wire to an AND gate $\forall w \in S_1^{\textit{max-tot-depth}}$.

Proof. This fact is very easy to easy. Clearly, $w \neq n+q$, since $f_{n+q}(x^*) = 0$ while $f_w(x^*) = 1$. Hence $\texttt{tot-depth}(w) < \ell$ and there exist layers of gates after the one containing w. Suppose w is an input wire to an OR gate. Since $f_w(x^*) = 1$, for some OR gate w' in the next layer of gates, $f_{w'}(x^*) = 1$. Hence, $\exists w' \in S_1$ such that $\texttt{tot-depth}(w) < \texttt{tot-depth}(w')$ which contradicts the fact that $w \in S_1^{\mathsf{max\text{-}tot\text{-}depth}}$.

Lemma 5. *For any wire $w \in [n+q]$, if $f_w(x^*) = 1$, then r_w is known to $\mathcal{B}$.*

Proof. We can prove this by observing the randomness we have set on each wire, from the output wire to the input wires. From Lemma 4, the first such wire we would see would be an input to an AND gate, which by our imposed circuit structure would be an OR gate. For an input wire w, of an AND gate, satisfying $f_w(x^*) = 1$, r_w was chosen at random and hence is known. This forms the base case for the induction. The previous argument also holds for any OR gate[15]. If w were an AND gate and $f_w(x^*) = 1$, then it feeds an OR gate w' in the next layer of gates, with $f_{w'}(x^*) = 1$. By induction hypothesis, $r_{w'}$ is known. We also know that since w' is an OR gate, $r_w = r_{w'}$ and hence r_w is known. This completes the proof.

Lemma 6. *For any wire $w \in [n+q]$, if $f_w(x^*) = 0$, then $g_j^{r_w}$ is known, where $j = \texttt{AND-depth}(w)$.*

Proof. We can prove this by observing the randomness we have set on each wire, from the output wire to the input wires. The statement is true for the output wire $w = n+q$, since $g_{\ell_{\mathsf{AND}}}^{c_{n+1} \cdot c_{n+2} \cdot \ldots \cdot c_{n+\ell_{\mathsf{AND}}}}$ can be computed using ℓ_{AND} pairings of g^{c_m}, $n+1 \leq m \leq n+\ell_{\mathsf{AND}}$. This forms the base case. We can now argue inductively. If w is an input to an OR gate w', then $r_w = r_{w'}$ and $\texttt{AND-depth}(w) = j$. If $f_{w'}(x^*) = 1$, then by Lemma 5, $r_{w'}$ is known and hence $g_j^{r_w}$ is known. If $f_{w'}(x^*) = 0$, then by the induction hypothesis, $g_j^{r_{w'}}$ is known and hence $g_j^{r_w}$ is known. If w is an input to an AND gate, then $r_w = c_{n+1} \cdot \ldots \cdot c_{n+j} + \eta_w$ and $g_j^{r_w}$ can be computed since $g_j^{c_{n+1} \cdot c_{n+2} \cdot \ldots \cdot c_{n+j}}$ can be computed using j pairings of g^{c_m}, $n+1 \leq m \leq n+j$ and η_w is known[16]. Hence, this completes the proof.

Lemma 7. *For any wire $w \in [n+q]$, $g_j^{r_w}$ is known, where $j = \texttt{AND-depth}(w)$.*

Proof. The statement follows directly from Lemmas 5 and 6.

We now describe how $\mathcal{B}$ generates key components for wires w, case-wise, according to whether w is an input wire, and AND gate or a FANOUT-gate as described below.

– *Input wire*
By convention, if $w \in [n]$, then it corresponds to the w-th input. If $x_w^* = 1$, then r_w is known from Lemma 5. The key component is:

$$K_w = (D_{w,1})^{r_w} = g^{r_w d_{w,1}}$$

If $x_w^* = 0$, then g^{r_w} is known from Lemma 6. The key component is:

$$K_w = (g^{r_w})^{z_w} = g^{r_w d_{w,1}}$$

– *AND gate*
Suppose that $w \in \mathsf{Gates}$ and that $\texttt{GateType}(w) = \text{AND}$. In addition, let $j = \texttt{AND-depth}(w)$.
- If $f_w(x^*) = 1$, then r_w is known from Lemma 5. Further, $g_{j-1}^{r_{A(w)}}$ and $g_{j-1}^{r_{B(w)}}$ are known from Lemma 7. $\mathcal{B}$ chooses $\psi_w = a_w$, $\phi_w = b_w$ at random. The key components are:

$$K_{w,1} = g^{a_w}, K_{w,2} = g^{b_w}, K_{w,3} = g_{j-1}^{r_w - a_w \cdot r_{A(w)} - b_w \cdot r_{B(w)}}$$

[15] This is inclusive even of randomness on either side of potential FANOUT-gates because if the wire is 1, the randomness on the wire going as input to the FANOUT-gate (the actual output wire of the gate under consideration) is chosen at random.

[16] This is inclusive even of randomness on either side of potential FANOUT-gates because if the wire is 0, the randomness on the wire going as input to the FANOUT-gate (the actual output wire of the gate under consideration) is chosen in a similar fashion to the randomness on the output wires of the FANOUT-gate, except for the η-component which is known.

- If $f_w(x^*) = 0$, we know that at least one of $f_{A(w)}(x^*)$ and $f_{B(w)}(x^*)$ must be zero.
  - If $f_{A(w)}(x^*) = 0$, $\mathcal{B}$ implicitly sets $a_w = c_{n+j} + \psi_w$ and $b_w = \phi_w$, where $\psi_w, \phi_w \in \mathbb{Z}_p$ are randomly chosen elements.
  - If $f_{A(w)}(x^*) = 1$ and hence $f_{B(w)}(x^*) = 0$, $\mathcal{B}$ implicitly sets $a_w = \psi_w$ and $b_w = c_{n+j} + \phi_w$, where $\psi_w, \phi_w \in \mathbb{Z}_p$ are randomly chosen elements.

  Then, $\mathcal{B}$ creates key components:

$$K_{w,1} = g^{a_w}, K_{w,2} = g^{b_w}, K_{w,3} = g_{j-1}^{r_w - a_w \cdot r_{A(w)} - b_w \cdot r_{B(w)}}$$

  We now show that these components can indeed be computed in every case. Note that the first two components can be computed in both cases. Consider $K_{w,3}$.
  - Consider the former case, where $f_{A(w)}(x^*) = 0$. Hence, $\mathcal{B}$ must have set $r_{A(w)} = c_{n+1} \cdot \ldots \cdot c_{n+j-1} + \eta_{A(w)}$. Hence, we have:

$$K_{w,3} = g_{j-1}^{\eta_w - c_{n+j} \cdot \eta_{A(w)} - \psi_w (c_{n+1} \cdot \ldots \cdot c_{n+j-1} + \eta_{A(w)}) - \phi_w \cdot r_{B(w)}} = g_{j-1}^{r_w - a_w \cdot r_{A(w)} - b_w \cdot r_{B(w)}}$$

    which can be computed as follows. $\eta_{A(w)}$ is known, and from Lemma 7, $g_{j-1}^{r_{B(w)}}$ can be computed. Further, $g_{j-1}^{c_{n+1} \cdot \ldots \cdot c_{n+j-1}}$ can be computed using $j-1$ pairings of g^{c_m}, $n+1 \leq m \leq n+j-1$. Hence the key component can be computed.
  - Consider the latter case, where $f_{A(w)}(x^*) = 1$ and $f_{B(w)}(x^*) = 0$. Hence, $\mathcal{B}$ must have set $r_{B(w)} = c_{n+1} \cdot \ldots \cdot c_{n+j-1} + \eta_{B(w)}$. Hence, we have:

$$K_{w,3} = g_{j-1}^{\eta_w - c_{n+j} \cdot \eta_{B(w)} - \phi_w (c_{n+1} \cdot \ldots \cdot c_{n+j-1} + \eta_{B(w)}) - \psi_w \cdot r_{A(w)}} = g_{j-1}^{r_w - a_w \cdot r_{A(w)} - b_w \cdot r_{B(w)}}$$

    which can be computed as outlined in the former case.

– *FANOUT-gate*

Suppose that $w \in \mathsf{Gates}$, $\mathtt{GateType}(w) = \text{OR}$ and that the fanout of w is greater than 1. In addition, let $j = \mathtt{AND\text{-}depth}(w)$. In this case, a FANOUT-gate would have been placed after w. Let r_w^{L} denote the randomness on the wire going as input to the FANOUT-gate (the actual output wire of the gate under consideration) and let $r_w^{\mathsf{R},i}$ denote the randomness on the ith fanout wire of the FANOUT-gate (there would be as many of these as the fanout of the gate w). If $f_w(x^*) = 1$, then by Lemma 5, r_w^{L} and $r_w^{\mathsf{R},i}$ are known for all $i \in [\Delta]$, where Δ is the fanout of the gate w. The key components are:

$$K_{w,w',i} = g_{j-1}^{\left(r_w^{\mathsf{R},i} - r_w^{\mathsf{L}}\right)}$$

for all $i \in [\Delta]$, where Δ is the fanout of the gate w.

Suppose $f_w(x^*) = 0$. Then, by our imposed circuit structure, each wire fanning out of w is an input to an AND gate and hence $r_w^{\mathsf{R},i} = c_{n+1} \cdot \ldots \cdot c_{n+j} + \eta_w^{\mathsf{R},i}$ for all $i \in [\Delta]$, where Δ is the fanout of the gate w. Also, $r_w^{\mathsf{L}} = c_{n+1} \cdot \ldots \cdot c_{n+j} + \eta_w^{\mathsf{L}}$. Hence, the key components:

$$K_{w,w',i} = g_{j-1}^{\left(\eta_w^{\mathsf{R},i} - \eta_w^{\mathsf{L}}\right)} = g_{j-1}^{\left(r_w^{\mathsf{R},i} - r_w^{\mathsf{L}}\right)}$$

can be computed for all $i \in [\Delta]$, where Δ is the fanout of the gate w, since η_w^{L} and $\eta_w^{\mathsf{R},i}$ are known for all $i \in [\Delta]$.

We set, for the output wire $w = n + q$, $\eta_w = 0$, so that $r_w = u$ in $\mathcal{B}$'s internal view. It is easy to see that a_w and b_w have the same distribution in the real game and game executed by $\mathcal{B}$, since in the real game, they are chosen at random and in the game executed by $\mathcal{B}$, they are either chosen at random or are values offset by some random values ψ_w and ϕ_w, respectively. For $w \in [n + q - 1]$, r_w also has the same distribution in the real game and the game executed by $\mathcal{B}$, since in the real game, they are chosen so that randomness on the input wires of an OR gate are the same as the randomness on its output wire, and they are chosen at random for an AND gate, and in the game executed by $\mathcal{B}$, they are chosen in the exact same way, where

being "chosen at random" is either truly satisfied or fixed values are offset by random η_w values. Now, we look at r_{n+q}. In the real game, it is a fixed value u, and in the game executed by $\mathcal{B}$, by setting $\eta_{n+q} = 0$, $r_{n+q} = c_{n+1} \cdot c_{n+2} \cdot \ldots \cdot c_{n+\ell_{\mathsf{AND}}} = u$ internally. Hence, they too have the same distribution. Hence all the parameters in the real game and game executed by $\mathcal{B}$ have the identical distribution.

Evaluate:
Suppose a query is made for a secret key for an input $x \in \{0,1\}^n$. If $x = x^*$, then $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ identifies an arbitrary t such that $x_t \neq x_t^*$. Through ℓ_{AND} pairings of g^{c_m}, $n+1 \leq m \leq n+\ell_{\mathsf{AND}}$, it computes $H = g_{\ell_{\mathsf{AND}}}^{u} = g_{\ell_{\mathsf{AND}}}^{c_{n+1} \cdot \ldots \cdot c_{n+\ell_{\mathsf{AND}}}}$. Then, through pairing of $D_{m,x_m} \forall m \in [n] \backslash \{t\}$, it computes $g_{n-1}^{\prod_{m \in [n] \backslash \{t\}} d_{m,x_m}}$ and raises it to $d_{t,x_t} = z_t$ to get $H' = g_{n-1}^{\prod_{m \in [n]} d_{m,x_m}}$. Finally, it computes $H'' = e(H, H') = g_{n+\ell_{\mathsf{AND}}-1}^{u \prod_{m \in [n]} d_{m,x_m}} = F(k,x)$ and outputs it.

Eventually, $\mathcal{A}$ will issue a challenge input $\tilde{x}$. If $\tilde{x} = x^*$, $\mathcal{B}$ will return the value T and output the same bit as $\mathcal{A}$ does as its guess. If $\tilde{x} \neq x^*$, $\mathcal{B}$ outputs a random bit as its guess.

This completes the description of the adversary $\mathcal{B}$. We first note that in the case where T is part of a MDDH tuple, the real game and game executed by $\mathcal{B}$ have the identical distribution. Secondly, in both cases (i.e., whether or not T is part of the MDDH tuple), as long as $\mathcal{B}$ does not abort, once again, the real game and game executed by $\mathcal{B}$ have the identical distribution, except for the output of $\mathcal{B}$ on the challenge query x^*. We now analyze the probability that $\mathcal{B}$'s guess was correct. Let δ' denote $\mathcal{B}$'s output and let δ denote whether T is an MDDH tuple or not, $\delta, \delta' \in \{0,1\}$. Now

$$\begin{aligned}
\Pr[\delta' = \delta] &= \Pr[\delta' = \delta | \mathsf{abort}] \Pr[\mathsf{abort}] + \Pr[\delta' = \delta | \overline{\mathsf{abort}}] \Pr[\overline{\mathsf{abort}}] \\
&= \frac{1}{2}(1 - 2^{-n}) + \Pr[\delta' = \delta | \overline{\mathsf{abort}}] \cdot (2^{-n}) \\
&= \frac{1}{2}(1 - 2^{-n}) + \left(\frac{1}{2} + \epsilon\right) \cdot (2^{-n}) \\
&= \frac{1}{2} + \epsilon \cdot (2^{-n})
\end{aligned}$$

The set of equations shows that the advantage of $\mathcal{B}$ is $\epsilon(\lambda)/2^n$. The second equation is true since the probability of $\mathcal{B}$ not aborting is 2^{-n}. The third equation comes from the fact that the probability of the adversary winning conditioned on not aborting is the same as the original probability of winning.

This completes the proof of the theorem, which establishes the pseudorandomness property of the construction. Hence, the constrained PRF construction for the circuit-predicate case is secure under the κ-MDDH assumption.

Remarks. We illustrate that this technique has use inspite of the restrictions on the structure of the circuit. There are several circuits for which the technique provides a reduction in the number of levels of multilinear maps used, even if it means the circuit is blown up because of our requirements. Consider the example circuit given ahead.

The circuit in Figure 3 would require 12 levels of multilinear maps without the use of the Free-OR technique. For applying the technique, we require that the inputs of OR gates do not fanout. To this end, the circuit has to be re-drawn as shown in Figure 4. After applying the Free-OR technique on the circuit in Figure 4, only 8 levels of multilinear maps are required, i.e., both layers of OR gates become "free". The consequence is that since the number of AND gates in the circuit has increased, the number of keys to be given out has gone up at the cost of decreasing the number of levels required. However, giving out keys is cheaper than moving up levels in multilinear maps. Also note that having successive levels of AND gates is not an issue and we need not blow up the circuit. Also, inputs of AND can fanout and this requires no additional blow-up. Thus, even for a small circuit such as the one in Figure 3, we obtain a reduction in the

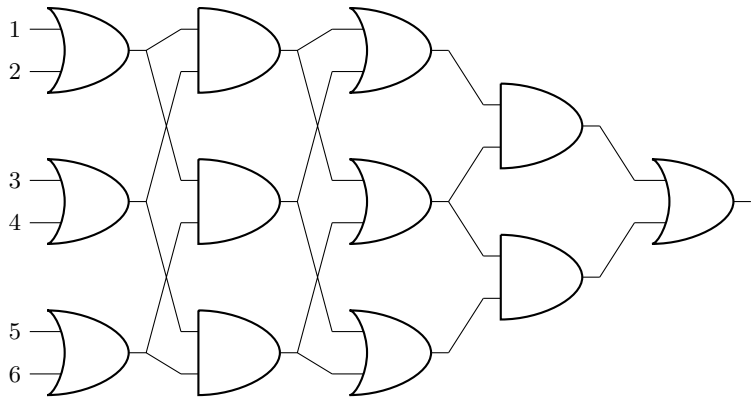

Fig. 3. Circuit illustrating use of the Free-OR technique

number of levels of multilinear maps (after the blow-up). Note that, even using our Free-AND technique would result in a scheme using 9 levels of multilinear maps.

C Backtracking attack in Constrained PRFs for Arbitrary Circuits

We first outline the intuition behind the Free-AND and Free-OR techniques and why they work in our setting. The constrained PRF construction for arbitrary circuits of Boneh and Waters [BW13] works by enabling the owner of the constrained key to learn some value associated with every wire in the circuit which evaluates to 1. On input x, when $f_w(x) = 1$ for some wire w, this value can be viewed as the PRF output on input x, with r_w as the PRF key. Proceeding in this manner, an evaluator can learn the PRF output on input x with u as the PRF key (which is the desired output), as $r_{n+q} = u$. In the [BW13] construction, the randomness on no two wires were correlated. To bridge that gap, one had to move up a level in the multilinear maps at every gate. Our techniques show that it is possible to have some of the randomness values on different wires correlated without compromising the security of the scheme. The exact correlation between the randomness follows from the structure of the gate under consideration.

AND gates. An AND gate evaluates to 1 if and only if both of its inputs evaluate to 1. In some sense, one must learn the value associated with the output wire of an AND gate if and only if one has already learnt the values associated with its two input wires. Informally, let $\mathtt{PRF}(k, x)$ denote the output of a PRF on input x and key k. Then, an evaluator who knows *both* $\mathtt{PRF}(r_{A(w)}, x)$ and $\mathtt{PRF}(r_{B(w)}, x)$, must be able to learn $\mathtt{PRF}(r_w, x)$. While this is possible in [BW13], by moving up a level in the multilinear maps, since the PRF under consideration is *key homomorphic*, we show that we can do so without moving up a level and simply by multiplying $\mathtt{PRF}(r_{A(w)}, x)$ and $\mathtt{PRF}(r_{B(w)}, x)$ to get $\mathtt{PRF}(r_w, x)$. This was possible, since we chose $r_w = r_{A(w)} + r_{B(w)}$. In general, it is easy to see that our technique can be extended to AND-gates having arbitrary[17] fan-in greater than 1, as we are simply secret sharing r_w among its input wires $r_{A(w)}$ and $r_{B(w)}$.

Now, note that this structure gives rise to the following property: if an evaluator knows the values associated with any two of the three wires connected to the AND gate (i.e. any two out of $\mathtt{PRF}(r_{A(w)}, x), \mathtt{PRF}(r_{B(w)}, x)$, and $\mathtt{PRF}(r_w, x)$), then the evaluator can compute the third value as well. If the (malicious) evaluator did indeed learn $\mathtt{PRF}(r_w, x)$, it must be the case that the AND gate evaluated to 1 and hence both its inputs must have been 1. In this case, no harm is done if the malicious evaluator indeed learned both $\mathtt{PRF}(r_{A(w)}, x)$ and $\mathtt{PRF}(r_{B(w)}, x)$.

[17] The fan-in of a gate is the number of inputs to it.

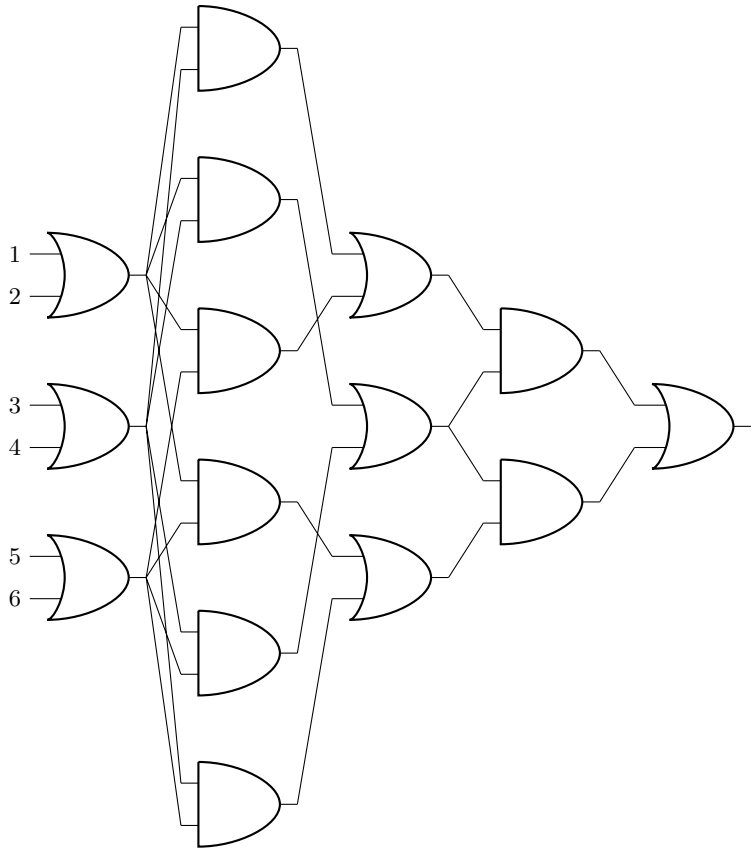

Fig. 4. Circuit in Figure 3 blown-up to apply the Free-OR technique

OR gates. Now, let us consider an OR gate. It evaluates to 1 if and only if any of its inputs evaluate to 1. Here, an evaluator must learn $\texttt{PRF}(r_w, x)$ iff he learns either $\texttt{PRF}(r_{A(w)}, x)$ or $\texttt{PRF}(r_{B(w)}, x)$. Note, that we achieved this by setting $r_w = r_{A(w)} = r_{B(w)}$. Again, it is easy to see that the technique can be extended to OR-gates having arbitrary fan-in greater than 1.

Now, note that this structure gives rise to the following property: if an evaluator knows the value associated with any one of the three wires connected to the AND gate (i.e. any one out of $\texttt{PRF}(r_{A(w)}, x), \texttt{PRF}(r_{B(w)}, x)$, and $\texttt{PRF}(r_w, x)$), then the evaluator can compute the other two values as well. Now, note that if the (malicious) evaluator did indeed learn $\texttt{PRF}(r_w, x)$, it must be the case that the OR gate evaluated to 1, and hence at least one of its inputs must have been 1. As was in the AND gate case, if $f_{A(w)}(x) = 1$, then no harm is done if the malicious evaluator indeed learns $\texttt{PRF}(r_{A(w)}, x)$. On the other hand, if $f_{B(w)}(x) = 0$ and yet $f_w(x) = 1$, the evaluator would learn $\texttt{PRF}(r_{B(w)}, x)$. This leads to an explicit attack when we consider circuits that have fan-out greater than 1 and hence we are unable to combine the Free-AND and Free-OR techniques for arbitrary circuits. This attack is similar in spirit to the "backtracking attack" described by Garg *et al.* [GGH+13c] in the context of attribute-based encryption. For a more detailed exposition of this attack in the context of constrained PRFs, we refer the reader to Appendix C.

Let us first describe the attack encountered when trying to combine Free-AND and Free-OR in more detail. The notion of backtracking, as discussed earlier, is that from the value associated with the output wire ($\texttt{PRF}(r_w, x)$), an adversarial evaluator can go backwards and learn the value associated with an input wire that actually evaluates to 0. This leads to an attack as illustrated ahead.

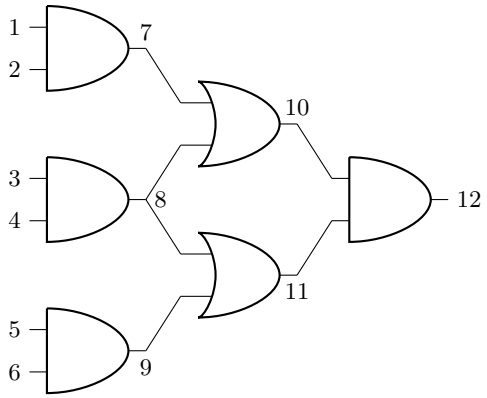

Fig. 5. Circuit with backtracking-attack

Consider the circuit shown in Figure 5. Assume that we try to use both Free-AND and Free-OR techniques, i.e., for AND gates, the randomness of the output wire is secret-shared between the randomness of the input wires, and for OR gates, all three wires have the same randomness. Suppose we have obtained the constrained key for this circuit. On input $x = 000011$, the circuit evaluates to 0 and hence we should not be able to evalute the PRF on x using the constrained key. However, we show that we indeed can.

The wires are numbered as in Figure 5. First, since we are employing both techniques, the depth of all gates as well as the final output wire is assumed to be the same. Let us refer to this common depth by γ. Secondly, we note that in the spirit of our Free-OR technique, a FANOUT-gate would have been placed at the output wire numbered 8, and we would have obtained keys for the FANOUT-gate, which would be of the form $g_\gamma^{\left(r_8^{\mathsf{R},1} - r_8^{\mathsf{L}}\right)}$ and $g_\gamma^{\left(r_8^{\mathsf{R},2} - r_8^{\mathsf{L}}\right)}$, where $r_8^{\mathsf{R},1}$ is the randomness on an input wire to the AND gate with output wire 10, and $r_8^{\mathsf{R},2}$ is the randomness on an input wire to the AND gate with output wire 11. On $x = 000011$, wire 9 evaluates to 1, while wires 7 and 8 evaluate to 0. Hence, we learn the value associated with wire 9. However, since wire 9 is an input wire to an OR gate, we also implicity learn the value associated with wire 8[18]. Since wire 8 is again an input to the OR gate with output wire 10, we implicity learn the value associated with wire 10, which should not have been the case since wire 10 evaluates to 0 on x. Since we have learnt the value associated with wire 9, we have also learnt the value associated with wire 11. Now that we have learnt the values associated with wires 10 and 11, we can learn the value associated with wire 12, which would be the PRF output on x. In other words, we have used the constrained key to obtain the PRF output on an x which does not satisfy the circuit, which compromises the security of the scheme.

We would like to draw attention to the following observations:

1. The attack discussed above was possible only because the value associated with 8 which was learnt only implicitly, and could not have been learnt otherwise, could be used in another part of the circuit, in other words, the attack was possible because the fanout of the AND gate with output wire 8 was more than 1. If this were not the case, the value learnt implicitly would have no further use and this would not be an attack. Since $\mathbf{NC}^1$ circuits have this property of fanout of every gate being restricted to 1, we can apply both techniques for $\mathbf{NC}^1$ circuits as shown ahead.
2. When only one of the techniques is used, such attacks do not occur since we, in some sense, re-set randomness at gates for which the technique is not used. For instance, when we use the Free-AND technique, we jump a level at OR gates and do a re-set on the randomness. In the Free-OR technique,

[18] Note that due to the presence of the FANOUT-gate, there are in fact three values associated with wire 8 due to the three randomness values r_8^{L}, $r_8^{\mathsf{R},1}$ and $r_8^{\mathsf{R},2}$. However, it is easy to see that all three of them can be learnt using the keys given out for the FANOUT-gate.

we explicitly avoid such attacks by requiring the circuit to have the property that AND gates have a fanout of 1. Hence, such attacks are prevented when only one of the techniques is applied on arbitrary circuits.

D Applications

In this section, we explain the application of our techniques to attribute-based encryption (ABE). Garg et al. [GGH$^+$13c] proposed an ABE scheme for all polynomial size circuits based on multilinear maps. This construction and its follow up works [BGG$^+$14] lend themselves to our optimisations. Hence, we get:

- an ABE scheme for arbitrary circuit predicates using an $(\ell_{\mathsf{OR}}+1)-$linear map, where ℓ_{OR} denotes the OR-depth of the predicate f when expressed as a boolean circuit. The security is provided under the $(\ell_{\mathsf{OR}}+1)-$MDDH assumption.
- an ABE scheme for circuit predicates using an $(\ell_{\mathsf{AND}}+1)-$linear map, where ℓ_{AND} denotes the AND-depth of the circuit predicate f. The security is provided under the $(\ell_{\mathsf{AND}}+1)-$MDDH assumption. In this construction, as in constrained PRFs, we require the circuit to be of a specific structure.
- an ABE scheme for all predicates $f \in \mathbf{NC}^1$ using an bilinear maps under the Decisional Bilinear Diffie-Hellman assumption. Our construction has some gains over the ABE construction for $\mathbf{NC}^1$ predicates of [GPSW06] that, during decryption, our construction involves only multiplications and additions over group elements and *no exponentiations* whereas [GPSW06] involves as many as l_x exponentiations (l_x represents the number of bits of x having the value 1)[19].
- The underlying hardness assumption is the $(\kappa, n)-$Multilinear Diffie-Hellman Exponent (MDHE) Assumption with n being the length of the attribute vector. This assumption states that when an adversary is given

$$\left(g^{c_1}, \ldots, g^{c_1^n}, g^{c_1^{n+2}}, \ldots, g^{c_1^{2n}}, g^{c_2}, \ldots, g^{c_\kappa}, \beta\right)$$

  distinguishing between $\beta = g_\kappa^{c_1^{n+1}\prod_{2\leq i\leq \kappa} c_i}$ and $\beta \xleftarrow{\$} \mathbb{G}_\kappa$ is hard. Here, $\kappa = \ell_{\mathsf{OR}}+1$ in the Free-AND case, $\kappa = \ell_{\mathsf{AND}}+1$ in the Free-OR case and $\kappa = 1$ for the $\mathbf{NC}^1$ construction. Note that $(1, n)-$MDHE is same as the 'standard' Bilinear Diffie-Hellman Exponent (BDHE) assumption.

D.1 Attribute-based encryption for circuits

Here, we present an ABE scheme which is the modified form of the one in [GGH$^+$13c] applying our "Free-AND" optimisation in detail. [20] The advantages provided by the resulting ABE scheme can be viewed in two ways.

- When a multilinear maps of $\kappa = \ell_{\mathsf{OR}}+1$ levels is used, our scheme can support circuits of "OR depth" ℓ_{OR} (and arbitrary levels of AND gates), whereas the scheme in [GGH$^+$13c] can only support circuits of *total* depth atmost ℓ_{OR}.
- Also, if the circuit family that needs to be supported by the ABE scheme has circuits whose OR depth is lesser than their overall depth, our scheme only requires lesser levels in multilinear maps. Thus, the public parameters, the keys and the ciphertexts are shorter.

In addition, no secret key components are required for the AND gates in any case. Now we present our "Free-AND" ABE construction.

[19] However, [GPSW06] can support any k out of n threshold gate, for $k \in [n]$, whereas our scheme can support only n out of n and 1 out of n threshold gates.

[20] All the other versions of ABE provided above can be obtained in a similar manner.

Construction $\mathsf{ABE.Setup}(1^\lambda, n, \ell_{\mathsf{OR}})$**:**
The setup algorithm takes as input the security parameter λ, the bit length, n, of the attribute vector and ℓ_{OR}, the maximum OR-depth of the circuit. The algorithm runs $\mathcal{G}(1^\lambda, \kappa = \ell_{\mathsf{OR}} + 1)$ and outputs a sequence of groups $\mathbb{G} = (\mathbb{G}_1, \ldots, \mathbb{G}_\kappa)$ of prime order p with canonical generators $g_1, \ldots, g_\kappa$, where $g = g_1$. It chooses an exponent $\alpha \in \mathbb{Z}_p$ and group elements $h_1, \ldots, h_n$ at random. It then sets the keys as:

$$mpk = (\mathbb{G}, p, g_1, \ldots, g_\kappa, g_\kappa^\alpha, h_1, \ldots, h_n); msk = (g_{\kappa-1})^\alpha$$

$\mathsf{ABE.Enc}(mpk, x \in \{0,1\}^n, M \in \{0,1\})$**:**
The encryption algorithm takes as input the attribute vector x and a message $M \in \{0,1\}$. The encryption algorithm chooses a random $s \in \mathbb{Z}_p$ and sets $C_M = (g_\kappa^\alpha)^s$ if $M = 1$, otherwise $C_M = g_\kappa^r$ for a random $r \in \mathbb{Z}_p$. Let $S \subseteq [n]$ be the set of i such that $x_i = 1$. The ciphertext is

$$CT = (C_M, g^s, \forall i \in S\ C_i = h_i^s)$$

$\mathsf{ABE.KeyGen}(msk, f = (n, q, A, B, \mathtt{GateType}))$**:**
The key generation algorithm takes as input the master secret key msk and a circuit description f. The circuit has $n + q$ wires with n input wires, q gates and the wire $n + q$ designated as the output wire.

To generate the secret key k_f, the key generation algorithm chooses random $r_1, \ldots, r_n \in \mathbb{Z}_p$ and $z_1, \ldots, z_n \in \mathbb{Z}_p$, where we think of the random values r_w and z_w as being associated with the input wire w. For each $w \in [n + q] \backslash [n]$, if $\mathtt{GateType}(w) = \text{AND}$, it sets $r_w = r_{A(w)} + r_{B(w)}$, otherwise, it chooses $r_w \in \mathbb{Z}_p$ at random.

Next, the algorithm generates key components. The structure of the key components depends on whether w is an input wire or an output of an OR gate. For AND gates, we do not need to give out any keys. The key components in each case are described below.

– *Input wire*
By convention, if $w \in [n]$, then it corresponds to the w-th input. The key component is:

$$K_{w,1} = g^{r_w} h_w^{z_w}, K_{w,2} = g^{-z_w}$$

– *OR gate*
Suppose that $w \in \mathsf{Gates}$ and that $\mathtt{GateType}(w) = \text{OR}$. In addition, let $j = \mathtt{depth}(w)$ be the OR-depth of the wire w. The algorithm chooses random $a_w, b_w \in \mathbb{Z}_p$. Then, the algorithm creates key components:

$$K_{w,1} = g^{a_w}, K_{w,2} = g^{b_w}, K_{w,3} = g_j^{r_w - a_w \cdot r_{A(w)}}, K_{w,4} = g_j^{r_w - b_w \cdot r_{B(w)}}$$

In addition to these, the algorithm also creates a "header" component $K_H = (g_{\kappa-1})^{\alpha - r_{n+q}}$ The secret key k_f consists of all these key components.

$\mathsf{ABE.Decrypt}(k_f, CT)$**:**
The decryption algorithm takes as input a secret key k_f for the circuit $f = (n, q, A, B, \mathtt{GateType})$ and an input $x \in \{0,1\}^n$. The algorithm first checks that $f(x) = 1$, and if not, it aborts.

The goal of decryption is to compute $g_\kappa^{\alpha s}$ using which M can be obtained from C_M. Hence, first using the header component the algorithm computes

$$E' = e(K_H, g^s) = e(g_{\kappa-1}^{\alpha - r_{n+q}}, g^s) = g_\kappa^{\alpha s} g_\kappa^{-r_{n+q} \cdot s}$$

Now the goal is reduced to computing $g_\kappa^{-r_{n+q} \cdot s}$.

The algorithm evaluates the circuit from input level to the output level. Consider the wire w at OR-depth j. If $f_w(x) = 1$, then, the algorithm computes $E_w = g_{j+1}^{s r_w}$. If $f_w(x) = 0$, then nothing needs to be computed for that wire. The algorithm proceeds iteratively starting with computing E_1 and proceeds, in order, to compute E_{n+q}. Computing these values in order ensures that the computation on a wire with OR-depth $j - 1$ that evaluates to 1, will be defined before computing for a wire with OR-depth j.

We show how to compute E_w for all w where $f_w(x) = 1$, case-wise, according to whether the wire is an input, an OR gate or an AND gate.

– *Input wire*
By convention, if $w \in [n]$, then it corresponds to the w-th input. Suppose $f_w(x) = 1$. The algorithm computes:

$$E_w = e\left(K_{w,1}, g^s\right) \cdot e\left(K_{w,2}, C_w\right) = e\left(g^{r_w} h_w^{z_w}, g^s\right) \cdot e\left(g^{-z_w}, h_w^s\right) = g_2^{sr_w}$$

– *OR gate*
Consider a wire $w \in \mathsf{Gates}$ with $\mathtt{GateType}(w) = \text{OR}$. In addition, let $j = \mathtt{OR\text{-}depth}(w)$ be the OR-depth of the wire w. The computation is performed if $f_w(x) = 1$. Note that in this case, at least one of $f_{A(w)}(x)$ and $f_{B(w)}(x)$ must be 1. If $f_{A(w)}(x) = 1$, the algorithm computes:

$$\begin{aligned} E_w &= e(E_{A(w)}, K_{w,1}) \cdot e(K_{w,3}, g^s) \\ &= e\left(g_j^{sr_{A(w)}}, g^{a_w}\right) \cdot e\left(g_j^{r_w - a_w \cdot r_{A(w)}}, g^s\right) = g_{j+1}^{sr_w} \end{aligned}$$

Otherwise, $f_{B(w)}(x) = 1$ and the algorithm computes:

$$\begin{aligned} E_w &= e(E_{B(w)}, K_{w,2}) \cdot e(K_{w,4}, g^s) \\ &= e\left(g_j^{sr_{B(w)}}, g^{b_w}\right) \cdot e\left(g_j^{r_w - b_w \cdot r_{B(w)}}, g^s\right) = g_{j+1}^{sr_w} \end{aligned}$$

– *AND gate*
Consider a wire $w \in \mathsf{Gates}$ with $\mathtt{GateType}(w) = \text{AND}$. In addition, let $j = \mathtt{OR\text{-}depth}(w)$ be the depth of the wire w. The computation is performed if $f_w(x) = 1$. Note that in this case, $f_{A(w)}(x) = f_{B(w)}(x) = 1$. The algorithm computes:

$$E_w = E_{A(w)} \cdot E_{B(w)} = g_{j+1}^{sr_{A(w)}} \cdot g_{j+1}^{sr_{B(w)}} = g_{j+1}^{sr_w}$$

If the output wire $f(x) = f_{n+q}(x) = 1$, then the algorithm would have computed $g_\kappa^{r_{n+q} \cdot s}$. It finally computes $E' \cdot E_{n+q} = g_\kappa^{\alpha s}$. If $C_M = g_\kappa^{\alpha s}$ output $M = 1$, otherwise output $M = 0$.

Proof of ABE scheme

Theorem 5. *If there exists a PPT adversary $\mathcal{A}$ that breaks the security of our ABE construction for circuits*[21] *of OR-depth $\kappa - 1$ and input length n with advantage $\epsilon(\lambda)$, then there exists a PPT algorithm $\mathcal{B}$ that breaks the κ- Multilinear Decisional Diffie-Hellman assumption with advantage $\epsilon(\lambda)/2^n$.*

Proof. The proof of this theorem is exactly the same as in [GGH+13c], except for the KeyGen oracle which works in a similar manner to the Constrain oracle in our Free-AND scheme. The algorithm $\mathcal{B}$ first receives a $\kappa = (\ell_{\mathsf{OR}} + 1)-$MDDH challenge consisting of the group sequence description $\mathbb{G}$ and $g = g_1, g^{c_1}, \ldots, g^{c_{\kappa+1}}$ along with T, where T is either $g_\kappa^{\prod_{i \in [\kappa+1]} c_i}$ or a random group element in $\mathbb{G}_\kappa$.

Setup:
$\mathcal{B}$ chooses an $x^* \in \{0,1\}^n$ uniformly at random. Next, it chooses random $y_1, \ldots, y_n \in \mathbb{Z}_p$ and sets

$$h_i = \begin{cases} g^{y_i} & \text{if } x_i^* = 1 \\ g^{y_i + c_1} & \text{if } x_i^* = 0 \end{cases}$$

for $i \in [n]$ and $\beta \in \{0,1\}$.

It then sets $g_\kappa^\alpha = g_\kappa^{\zeta + \prod_{i \in [\kappa]} c_i}$ and $g^s = g^{c_{\kappa+1}}$, where $\zeta \in \mathbb{Z}_p$ is chosen randomly.

KeyGen phase:
Suppose a query is made for a secret key for a circuit $f = (n, q, A, B, \mathtt{GateType})$. If $f(x^*) = 1$, then $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ generates key components for every wire w, case-wise, according to whether w is an input wire or an OR gate as described below.

[21] The circuits should also satisfy the properties specified in our Free-AND Constrained PRF scheme, but note that those are not restrictions; they are just for making the exposition of our scheme simpler.

– *Input wire*
By convention, if $w \in [n]$, it corresponds to the w-th input.
If $x^*_w = 1$, then $\mathcal{B}$ chooses $\eta_w = r_w$ and $\nu_w = z_w$ at random. The key components are:

$$K_{w,1} = g^{r_w} h_w^{z_w}, K_{w,2} = g^{-z_w}$$

If $x^*_w = 0$, then $\mathcal{B}$ implicitly sets $r_w = c_1 c_2 + \eta_w, z_w = -c_2 + \nu_w$, where $\eta_w, \nu_w \in \mathbb{Z}_p$ are randomly chosen elements. The key components are:

$$(K_{w,1}, K_{w,2}) = \left(g^{c_1 c_2 + \eta_w} h_w^{-c_2 + \nu_w}, g^{c_2 - \nu_w}\right) = \left(g^{-c_2 y_w + \eta_w + (y_w + c_1)\nu_w}, g^{c_2 - \nu_w}\right)$$

Note that these components can be generated by $\mathcal{B}$ from the components known to it.

– *OR gate*
Suppose that $w \in \mathsf{Gates}$ and that $\mathtt{GateType}(w) = \text{OR}$. In addition, let $j = \mathtt{OR\text{-}depth}(w)$. In order to show that $\mathcal{B}$ can simulate all the key components, we shall additionally show the following property:

Property 2. For any gate $w \in \mathsf{Gates}$, $\mathcal{B}$ will be able to compute $g_{j+1}^{r_w}$, where $j = \mathtt{OR\text{-}depth}(w)$.

We will prove the above property through induction on the OR-depth j; doing this will enable us to prove that $\mathcal{B}$ can compute all the key components required to give out the secret key. The base case of the input wires ($j = 1$) follows as we know that for an input wire w, $\mathcal{B}$ can compute $g_2^{r_w}$, where r_w is of the form η_w or $c_1 c_2 + \eta_w$. We now proceed to show the computation of the key-components. In each case, we show that property 2 is satisfied.

CASE 1: If $f_w(x^*) = 1$, then $\mathcal{B}$ chooses $\psi_w = a_w$, $\phi_w = b_w$ and $\eta_w = r_w$ at random. Then, $\mathcal{B}$ creates key components:

$$K_{w,1} = g^{a_w}, K_{w,2} = g^{b_w}, K_{w,3} = g_j^{r_w - a_w \cdot r_{A(w)}}, K_{w,4} = g_j^{r_w - b_w \cdot r_{B(w)}}$$

By virtue of property 2, since $\mathtt{OR\text{-}depth}(A(w)) = \mathtt{OR\text{-}depth}(B(w)) = j - 1$, by the induction hypothesis, we know that $\mathcal{B}$ can compute $g_j^{r_{A(w)}}$ and $g_j^{r_{B(w)}}$. Hence, $\mathcal{B}$ can compute the above key-components, as the remaining exponents were all chosen at random by $\mathcal{B}$. Further, since r_w was chosen by $\mathcal{B}$, $g_{j+1}^{r_w}$ can be be computed for this wire, and hence property 2 holds for this wire as well (at OR-depth j).

CASE 2: If $f_w(x^*) = 0$, then $\mathcal{B}$ implicitly sets $r_w = c_1 \cdots c_{j+1} + \eta_w$, where $\eta_w \in \mathbb{Z}_p$ is a randomly chosen element. Since η_w was chosen at random, note that $g_{j+1}^{r_w}$ can be be computed for this wire (since $g_{j+1}^{c_1 \cdots c_{j+1}}$ can be computed using $j + 1$ pairings of g^{c_m}, $1 \le m \le j + 1$), and hence property 2 holds for this wire as well. For computing the key-components, the choices of a_w and b_w are done more carefully.

1. If the level before the current level consists of the inputs, then $\mathcal{B}$ would know the values of $\eta_{A(w)}$ and $\eta_{B(w)}$, since for input wires, these values are always chosen at random. In this case, $\mathcal{B}$ implicitly sets $a_w = c_{j+1} + \psi_w$ and $b_w = c_{j+1} + \phi_w$, where $\psi_w, \phi_w \in \mathbb{Z}_p$ are randomly chosen elements. Then, $\mathcal{B}$ creates key components:

$$K_{w,1} = g^{c_{j+1} + \psi_w} = g^{a_w}, K_{w,2} = g^{c_{j+1} + \phi_w} = g^{b_w},$$

$$K_{w,3} = g_j^{\eta_w - c_{j+1} \cdot \eta_{A(w)} - \psi_w (c_1 \cdots c_j + \eta_{A(w)})} = g_j^{r_w - a_w \cdot r_{A(w)}},$$

$$K_{w,4} = g_j^{\eta_w - c_{j+1} \cdot \eta_{B(w)} - \psi_w (c_1 \cdots c_j + \eta_{B(w)})} = g_j^{r_w - b_w \cdot r_{B(w)}}$$

$\mathcal{B}$ is able to create the last two key components due to a cancellation. Since $f_{A(w)}(x^*) = f_{B(w)}(x^*) = 0$, $\mathcal{B}$ would have set $r_{A(w)} = c_1 \cdots c_j + \eta_{A(w)}$ and $r_{B(w)} = c_1 \cdots c_j + \eta_{B(w)}$.

2. Suppose the level before the current level consists of AND gates. Since $f_{A(w)}(x^*) = 0$, we have two cases: either one of $f_{A(A(w))}(x^*)$ and $f_{B(A(w))}(x^*)$ is zero, or both of them are zero. $\mathcal{B}$ sets $a_w = c_{j+1} + \psi_w$ in the former case, and $a_w = \frac{1}{2}c_{j+1} + \psi_w$ in the latter case, where $\psi_w \in \mathbb{Z}_p$ is a randomly chosen element. Similarly, since $f_{B(w)}(x^*) = 0$, we have two cases: either one of $f_{A(B(w))}(x^*)$ and $f_{B(B(w))}(x^*)$ must be zero, or both of them must be zero. $\mathcal{B}$ sets $b_w = c_{j+1} + \phi_w$ in the former case, and $b_w = \frac{1}{2}c_{j+1} + \phi_w$ in the latter case, where $\phi_w \in \mathbb{Z}_p$ is a randomly chosen element. Then, $\mathcal{B}$ creates key components:

$$K_{w,1} = g^{a_w}, K_{w,2} = g^{b_w}, K_{w,3} = g_j^{r_w - a_w \cdot r_{A(w)}}, K_{w,4} = g_j^{r_w - b_w \cdot r_{B(w)}}$$

We now show that these components can indeed be computed in every case. Note that the first two components can be computed in every case. Consider $K_{w,3}$ (a similar argument holds for $K_{w,4}$).

(a) Consider the first case, where one of $f_{A(A(w))}(x^*)$ and $f_{B(A(w))}(x^*)$ is zero. In particular, without loss of generality, assume that $f_{A(A(w))}(x^*) = 0$ and $f_{B(A(w))}(x^*) = 1$. Hence, $\mathcal{B}$ must have set $r_{A(A(w))} = c_1 \cdots c_j + \eta_{A(A(w))}$ and $r_{B(A(w))} = \eta_{B(A(w))}$. Since $A(w)$ is an AND gate, we would have $r_{A(w)} = r_{A(A(w))} + r_{B(A(w))} = c_1 \cdots c_j + \eta_{A(A(w))} + \eta_{B(A(w))}$. Hence, we have:

$$K_{w,3} = g_j^{\eta_w - c_{j+1}(\eta_{A(A(w))} + \eta_{B(A(w))}) - \psi_w(c_1 \cdots c_j + \eta_{A(A(w))} + \eta_{B(A(w))})} = g_j^{r_w - a_w \cdot r_{A(w)}}$$

which can be computed as follows. Since $A(w)$ is an AND gate, $A(A(w))$ and $B(A(w))$ must be OR gates, in which case, we would know the values of $\eta_{A(A(w))}$ and $\eta_{B(A(w))}$. Further, $g_j^{c_1 \cdots c_j}$ can be computed using j pairings of g^{c_m}, $1 \leq m \leq j$. Hence the key component can be computed.

(b) Consider the second case, where $f_{A(A(w))}(x^*) = f_{B(A(w))}(x^*) = 0$. Hence, $\mathcal{B}$ must have set $r_{A(A(w))} = c_1 \cdots c_j + \eta_{A(A(w))}$ and $r_{B(A(w))} = c_1 \cdots c_j + \eta_{B(A(w))}$. Since $A(w)$ is an AND gate, we would have $r_{A(w)} = r_{A(A(w))} + r_{B(A(w))} = 2c_1 \cdots c_j + \eta_{A(A(w))} + \eta_{B(A(w))}$. Hence, we have:

$$K_{w,3} = g_j^{\eta_w - \frac{1}{2}c_{j+1}(\eta_{A(A(w))} + \eta_{B(A(w))}) - \psi_w(c_1 \cdots c_j + \eta_{A(A(w))} + \eta_{B(A(w))})} = g_j^{r_w - a_w \cdot r_{A(w)}}$$

which can be computed as outlined in the former case.

Thus, the four key components can be given out in every case.

– *AND gate*

Suppose that $w \in \mathsf{Gates}$ and that $\mathtt{GateType}(w) = \text{AND}$. Let $j = \mathtt{OR\text{-}depth}(w)$. $\mathcal{B}$ sets $r_w = r_{A(w)} + r_{B(w)}$. Clearly, property 2 holds here as well, i.e., $g_{j+1}^{r_w} = g_{j+1}^{r_{A(w)}} + g_{j+1}^{r_{B(w)}}$ can be be computed for this wire, since both $g_{j+1}^{r_{A(w)}}, g_{j+1}^{r_{B(w)}}$ are known due to property 2.

Finally, for the output wire $w = n + q$, we will have $r_{n+q} = c_1 \cdots c_\kappa + \eta_{n+q}$. Now the header component K_H can be computed as $g_{\kappa-1}^{\alpha - r_{n+q}} = g_{\kappa-1}^{\zeta - \eta_{n+q}}$.

Challenge ciphertext:

$\mathcal{A}$ chooses an attribute vector $\tilde{x}$ at some point and gives it to $\mathcal{B}$. $\mathcal{B}$ first chooses $M_b \in \{0, 1\}$. Let $S \subseteq \{0, 1\}$ be the set of indices i such that $\tilde{x}_i = 1$. $\mathcal{B}$ now sets the challenge ciphertext as

$$CT = \left(M_b \cdot T \cdot g_\kappa^{s\zeta}, g^s, \forall i \in S\ C_i = (g^s)^{y_j}\right)$$

When $T = g_\kappa^{\prod_{i \in [\kappa+1]} c_i}$, CT is an encrpytion of 1, otherwise it is an encryption of 0.

Guess:

If $\tilde{x} = x^*$, $\mathcal{B}$ will output the same bit as $\mathcal{A}$ does as its guess. If $\tilde{x} \neq x^*$, $\mathcal{B}$ outputs a random bit as its guess.

This completes the description of the adversary $\mathcal{B}$. We first note that in the case where T is part of a MDDH tuple, the real game and game executed by $\mathcal{B}$ are statistically indistinguishable. Secondly, in both cases (i.e., whether or not T is part of the MDDH tuple), as long as $\mathcal{B}$ does not abort, once again, the real game and game executed by $\mathcal{B}$ are statistically indistinguishable, except for the output of $\mathcal{B}$ on the challenge

query x^*. We now analyze the probability that $\mathcal{B}$'s guess was correct. Let δ' denote $\mathcal{B}$'s output and let δ denote whether T is an MDDH tuple or not, $\delta, \delta' \in \{0,1\}$. Now

$$\begin{aligned}
\Pr[\delta' = \delta] &= \Pr[\delta' = \delta | \mathsf{abort}] \Pr[\mathsf{abort}] + \Pr[\delta' = \delta | \overline{\mathsf{abort}}] \Pr[\overline{\mathsf{abort}}] \\
&= \frac{1}{2}(1 - 2^{-n}) + \Pr[\delta' = \delta | \overline{\mathsf{abort}}] \cdot (2^{-n}) \\
&= \frac{1}{2}(1 - 2^{-n}) + \left(\frac{1}{2} + \epsilon\right) \cdot (2^{-n}) \\
&= \frac{1}{2} + \epsilon \cdot (2^{-n})
\end{aligned}$$

The set of equations shows that the advantage of $\mathcal{B}$ is $\epsilon(\lambda)/2^n$. The second equation is true since the probability of $\mathcal{B}$ not aborting is 2^{-n}. The third equation comes from the fact that the probability of the adversary winning conditioned on not aborting is the same as the original probability of winning.