Fully Secure Functional Encryption without Obfuscation

Sanjam Garg* Craig Gentry† Shai Halevi‡ Mark Zhandry§

Abstract

Previously known functional encryption (FE) schemes for general circuits relied on indistinguishability obfuscation, which in turn either relies on an exponential number of assumptions (basically, one per circuit), or a polynomial set of assumptions, but with an exponential loss in the security reduction. Additionally these schemes are proved in an unrealistic selective security model, where the adversary is forced to specify its target before seeing the public parameters. For these constructions, full security can be obtained but at the cost of an exponential loss in the security reduction.

In this work, we overcome the above limitations and realize a fully secure functional encryption scheme without using indistinguishability obfuscation. Specifically the security of our scheme relies only on the polynomial hardness of simple assumptions on multilinear maps.

1 Introduction

In traditional encryption schemes, decryption control is all or nothing: the sender encrypts its message under a particular key, and anyone with the corresponding secret key can recover the message. In contrast, functional encryption (FE) schemes [BSW11, O'N10] allow the sender to embed sophisticated functions into secret keys. More specifically, an FE scheme includes an authority, which holds a master secret key and publishes public system parameters. The sender uses the public parameters to encrypt its message m to obtain a ciphertext ct. A user may obtain a secret key sk_f for the function f from the authority (if the authority deems that the user is entitled). This key sk_f can be used to decrypt ct to recover $f(m)$; and nothing more. In a recent result, Garg et al. constructed the first FE scheme for general circuits using indistinguishability obfuscation (IO) [GGH+13b].

While tremendous progress has been made on justifying the security of IO [BR14, BGK+14, PST14, GLW14, GLSW14], ultimately the security of the resulting constructions still either relies on an exponential number of assumptions [BR14, BGK+14, PST14] (basically, one per circuit), or a polynomial set of assumptions, but with an exponential loss in the security reduction [GLW14, GLSW14]. For example, the recent IO scheme based on the MSE assumption [GLSW14] crucially uses complexity leveraging in its proof – specifically, the number of hybrids in the proof is proportional to $2^{|x|}$ where x is the input, and each hybrid "examines" a particular input x and implicitly "verifies" that the circuits C_0, C_1 in question satisfy $C_0(x) = C_1(x)$. Garg et al. [GGSW13] provide an intuitive argument suggesting that either of these shortcoming might be inherent when realizing indistinguishability obfuscation.[1] This intuitive argument however is not applicable to FE schemes. In this work we ask the following fundamental question:

*University of California at Berkeley, sanjamg@berkeley.edu
†IBM Research, craigbgentry@gmail.com
‡IBM Research, shaih@alum.mit.edu
§Stanford University, mzhandry@stanford.edu
[1] Garg et al. [GGSW13] only provide the intuition for witness encryption but it extends to IO.

Can we construct a functional encryption scheme for general circuits assuming only polynomial hardness of simple computational assumptions?

Another limitation of the Garg et al. [GGH+13b] scheme is that it is only *selectively secure* – that is, they have been proved secure only in an unrealistic model in which the adversary is required to specify the message m for its challenge ciphertext before it sees the public parameters of the FE scheme. We would like FE for circuits that is *fully secure* – i.e., that allows the adversary to choose m^* adaptively after seeing the public parameters and even responses to some of its private key queries. In general, one can trivially reduce full security to selectively security via *complexity leveraging* – essentially the reduction tries to guess the adversary's chosen m, and succeeds with probability $2^{-|m|}$ – but complexity leveraging loses a $2^{|m|}$ factor in the reduction to the underlying hard problem that we would like to avoid.

Can we construct a fully secure functional encryption scheme for general circuits without an exponential loss in the security reduction?

Achieving full security without the lossiness of complexity leveraging is just as important for FE for circuits as it was for identity-based encryption (IBE) ten years ago [Wat05, Gen06, Wat09], for both efficiency and conceptual reasons.

1.1 Our Results

In this work, we give positive answers to both questions above. Specifically we construct the first fully secure FE scheme for circuits without using indistinguishability obfuscation or any exponential loss in security reductions. Our scheme uses composite order multilinear maps in the asymmetric settings [BS02, GGH13a, CLT13] and security is based on polynomial hardness of fixed, relatively simple assumptions.

We extend the existing graded encoding schemes [GGH13a, CLT13] with a new *extension* function that serves as a crucial ingredient in our construction. This extension function serves a role similar to that of the straddling set systems of [BGK+14], binding various encodings so that only certain subsets can be paired together. The important difference is that the extension function allows the binding to happen dynamically and publicly. This allows, for example, an encrypter to bind ciphertext encodings together so that encodings from different ciphertexts cannot be "mixed and matched." We suspect that this new technique will be useful in other contexts as well.

1.2 Concurrent and Independent Work

In concurrent and independent work, Waters [Wat14] constructs a fully secure functional encryption (FE) scheme using indistinguishability obfuscation (IO) [GGH+13b]. While Waters' result on FE is exciting, the focus of this work is to avoid indistinguishability obfuscation altogether and to build fully secure functional encryption using simpler tools (multilinear maps and simple assumptions involving them).

2 Preliminaries

In this section, we start by providing the definition of adaptively secure FE for general circuits. Next we recall the notions of branching programs and graded encoding schemes and develop notation that will be needed in our context.

2.1 Adaptively Secure FE

A functional encryption system consists of four algorithms: $Setup$, $KeyGen$, $Encrypt$, and $Decrypt$.

- **Setup**(λ): The setup algorithm takes in the security parameter λ as input and outputs the public parameters MPK and a master secret key MSK.

- **KeyGen**(MSK, y): The key generation algorithm takes in the master secret key MSK, and an attribute string y as input. It outputs a private key SK_y for y. y is included as part of the secret key.

- **Encrypt**(MPK, x): The encryption algorithm takes in the public parameters MPK, and a message x as input. It outputs a ciphertext C.

- **Decrypt**(SK_y, C): The decryption algorithm takes a private key SK_y for attribute string y and a ciphertext C (encrypting say the message x) as input and outputs the value $\mathsf{C}(x, y)$, where C is a fixed universal circuit.

Correctness of the scheme requires that for correctly generated private keys for y and correctly generated ciphertexts encrypting x, decryption yields $\mathsf{C}(x, y)$ except with negligible probability.

We will now give the security definition for *adaptive* FE. This is described by a security game between a challenger and an attacker that proceeds as follows.

- **Setup:** The challenger runs the Setup algorithm and gives the public parameters MPK to the attacker.

- **Query Phase I:** The attacker queries the challenger for private keys corresponding to attribute strings $y_1, \ldots, y_{q_1}$, which the challenger provides.

- **Challenge:** The attacker declares two messages x_0, x_1. We require that $\forall i \in [q_1]$ we have that $\mathsf{C}(x_1, y_i) = \mathsf{C}(x_0, y_i)$. The challenger flips a random coin $\beta \in \{0, 1\}$ and runs $C \leftarrow$ **Encrypt**(MPK, x_β). The challenger gives the ciphertext C to the adversary.

- **Query Phase II:** The attacker queries the challenger for private keys corresponding to the attribute strings $y_{q_1+1}, \ldots, y_q$, with the added restriction that $\forall i \in \{q_1, \ldots, q\}$ we have $\mathsf{C}(x_1, y_i) = \mathsf{C}(x_0, y_i)$.

- **Guess:** The attacker outputs a guess β' for β.

The advantage of an attacker in this game is defined to be $\Pr[\beta = \beta'] - \frac{1}{2}$.

2.2 Branching Programs

A branching program consists of a sequence of steps, where each step is defined by a pair of permutations. In each step the the program examines one input bit, and depending on its value the program chooses one of the permutations. The program outputs 1 if and only if the multiplications of the permutations chosen in all steps is the identity permutation. In our setting, just like in previous work it will be easier to work with matrix branching programs that we define next.

Definition 1 (Matrix Branching Program)**.** *A* branching program *of width w and length ℓ on n-bit inputs is given by two 0/1 permutation matrices $M_0, M_1 \in \{0,1\}^{w \times w}$, $M_0 \neq M_1$ and by a sequence:*

$$BP = \big(\mathsf{inp}(i), B_{i,0}, B_{i,1}\big)_{i=1}^{\ell} \ ,$$

where each $B_{i,b}$ is a permutation matrix in $\{0,1\}^{w \times w}$, and $\mathsf{inp}(i) \in [n]$ is the input bit position examined in step i. We require that, for all inputs $x \in \{0,1\}^n$,

$$\prod_{i=1}^{\ell} B_{i,x_{\mathsf{inp}(i)}} \in \{M_0, M_1\}$$

Let (α, β) be a position where $M_1[\alpha, \beta] = 1$ and $M_0[\alpha, \beta] = 0$. Call (α, β) a distinguishing coordinate. The output of the branching program on input $x \in \{0,1\}^n$ is as follows:

$$BP(x) = \left(\prod_{i=1}^{\ell} B_{i,x_{\mathsf{inp}(i)}} \right) [\alpha, \beta]$$

Theorem 1 ([Bar86]). *For any depth-d fan-in-2 boolean circuit C, there exists an oblivious branching program of width 5 and length at most 4^d that computes the same function as the circuit C.*

Remark 1. *In our functional encryption construction we do not require that the branching program is of constant width. In particular we can use any reductions that result in a polynomial size branching program.*

For simplicity of notation, it will be convenient to consider two-input branching programs.[2] Here, the input $x \in \{0,1\}^{2n}$ is split into two inputs $(x[0], x[1])$. We then split inp into two functions:

- $\mathsf{inp}' : [\ell] \to \{0,1\}$ where $\mathsf{inp}'(i) = \lceil \mathsf{inp}(i)/n \rceil - 1$. Basically, inp' chooses which of the inputs $x[0]$ and $x[1]$ inp points to.
- $\mathsf{bit} : [\ell] \to [n]$ where $\mathsf{bit}(i) = \mathsf{inp}(i) \mod n$. Basically, bit chooses which bit of $x[b]$ inp points to, where b is the bit chosen by inp'.

Then we can write the branching program evaluation as

$$BP(x) = \left(\prod_{i=1}^{\ell} B_{i,x[\mathsf{inp}'(i)]_{\mathsf{bit}(i)}} \right) [\alpha, \beta]$$

Remark 2. *It is also straightforward to consider two-input branching programs where $x[0]$ and $x[1]$ have different sizes. We treat them as the same size for convenience.*

Kilian Randomization of Branching Programs. Let BP be a branching program as above. Fix a ring $\mathfrak{R}$. Choose random invertible matrices $R_1, \ldots, R_{\ell-1}$, and define a new branching program BP' which is identical to BP, except that the matrices $B_{i,b}$ are replaced with $\tilde{B}_{i,b} = R_{i-1} \cdot B_{i,b} \cdot R_i^{-1}$, where we take $R_0 = R_\ell = I_w$. We observe that

$$\prod_{i=1}^{\ell} \tilde{B}_{i,x_{\mathsf{inp}(i)}} = \prod_{i=1}^{\ell} B_{i,x_{\mathsf{inp}(i)}}$$

so that for every x we have that $BP'(x) = BP(x)$.

Moreover, we have the following theorem of Kilian:

Theorem 2 ([Kil88]). *Fix any input $x \in \{0,1\}^\ell$, and let $b = BP(x) = BP'(x)$. Then the set of matrices multiplied together to evaluate $BP'(x)$, namely the set*

$$\left\{ \tilde{B}_{i,x_{\mathsf{inp}(i)}} \right\}_{i \in [\ell]}$$

are distributed as uniform random $w \times w$ invertible matrices over $\mathfrak{R}$, conditioned on their product being M_b.

[2] Not to be confused with *dual-input* branching programs from [BGK+14].

2.3 Graded Encoding Scheme

Now, we describe the graded encoding scheme abstraction that will be needed in our context, mostly following [GGH13a, CLT13, GLW14]. To instantiate the abstraction, we can use Gentry et al.'s variant [GLW14] of the Coron-Lepoint-Tibouchi (CLT) graded encodings [CLT13]. This variant is designed to emulate multilinear groups of composite order, and to allow assumptions regarding subgroups of the multilinear groups. One key difference in our abstraction is a new *extension* function that we add to the GGH graded encoding abstraction. This new functionality will be crucial in our scheme. In Appendix A we briefly recall the CLT graded encodings and show how they can be adapted to also support this extension functionality.[3]

Definition 2 ($\mathbb{U}$-Graded Encoding System)**.** *A $\mathbb{U}$-Graded Encoding System consists of a ring $\mathfrak{R}$ and a system of sets $\mathcal{S} = \{S_T^{(\alpha)} \subset \{0,1\}^* : \alpha \in \mathfrak{R},\ T \subseteq \mathbb{U}, \}$, with the following properties:*

1. *For every fixed set T, the sets $\{S_T^{(\alpha)} : \alpha \in \mathfrak{R}\}$ are disjoint (hence they form a partition of $S_T \stackrel{\text{def}}{=} \bigcup_\alpha S_T^{(\alpha)}$).*
2. *There is an associative binary operation '$+$' and a self-inverse unary operation '$-$' (on $\{0,1\}^*$) such that for every $\alpha_1, \alpha_2 \in \mathfrak{R}$, every set $T \subseteq \mathbb{U}$, and every $u_1 \in S_T^{(\alpha_1)}$ and $u_2 \in S_T^{(\alpha_2)}$, it holds that*
$$u_1 + u_2 \in S_T^{(\alpha_1+\alpha_2)} \quad \text{and} \quad -u_1 \in S_T^{(-\alpha_1)}$$
where $\alpha_1 + \alpha_2$ and $-\alpha_1$ are addition and negation in $\mathfrak{R}$.
3. *There is an associative binary operation '$\times$' (on $\{0,1\}^*$) such that for every $\alpha_1, \alpha_2 \in \mathfrak{R}$, every T_1, T_2 with $T_1 \cup T_2 \subseteq \mathbb{U}$, and every $u_1 \in S_{T_1}^{(\alpha_1)}$ and $u_2 \in S_{T_2}^{(\alpha_2)}$, it holds that $u_1 \times u_2 \in S_{T_1 \cup T_2}^{(\alpha_1 \cdot \alpha_2)}$. Here $\alpha_1 \cdot \alpha_2$ is multiplication in $\mathfrak{R}$, and $T_1 \cup T_2$ is set union.*

CLT (and GGH) encodings do not quite meet the definition of graded encoding systems above, since the homomorphisms required in the definition eventually fail when the "noise" in the encodings becomes too large, analogously to how the homomorphisms may eventually fail in lattice-based homomorphic encryption. However, these noise issues are relatively straightforward (though tedious) to deal with.

Now, we define some procedures for graded encoding schemes. We start with the procedures standard in the graded encoding literature [GGH13a, CLT13].

Instance Generation. The randomized $\mathsf{InstGen}(1^\lambda, \mathbb{U}, r)$ takes as inputs the parameters $\lambda, \mathbb{U}, r$, and outputs params, where params is a description of a $\mathbb{U}$-Graded Encoding System as above for a ring $\mathfrak{R} = \mathfrak{R}_1 \times \ldots \times \mathfrak{R}_r$. We assume $\mathfrak{R}$ is chosen such that the density of zero divisors in each $\mathfrak{R}_i$ is negligible.

Note that setting $r = 1$ corresponds to the prime order setting, while $r > 1$ corresponds to the composite-order setting.

Ring Sampler. The randomized $\mathsf{samp}(\mathsf{params})$ outputs a "level-zero encoding" $a \in S_\phi^{(\alpha)}$ for a nearly uniform element $\alpha \in_R \mathfrak{R}$. (Note that we require that the "plaintext" $\alpha \in \mathfrak{R}$ is nearly uniform, but not that the encoding a is uniform in $S_\phi^{(\alpha)}$.)

[3] We note that the GGH encodings can also be extended to deal with this functionality as well but here we provide this it only for the CLT encodings.

Encoding. The (possibly randomized) $\mathsf{enc}(\mathsf{params}, T, a)$ takes a "level-zero" encoding $a \in S_\phi^{(\alpha)}$ for some $\alpha \in \mathfrak{R}$ and index $T \subseteq \mathbb{U}$, and outputs the "level-T" encoding $u \in S_T^{(\alpha)}$ for the same α.

Re-Randomization. The randomized $\mathsf{reRand}(\mathsf{params}, T, u)$ re-randomizes encodings relative to the same index. Specifically, for an index $T \subseteq \mathbb{U}$ and encoding $u \in S_T^{(\alpha)}$, it outputs another encoding $u' \in S_T^{(\alpha)}$. Moreover for any two $u_1, u_2 \in S_T^{(\alpha)}$, the output distributions of $\mathsf{reRand}(\mathsf{params}, T, u_1)$ and $\mathsf{reRand}(\mathsf{params}, T, u_2)$ are statistically indistinguishable.

Addition and negation. Given params and two encodings relative to the same index, $u_1 \in S_T^{(\alpha_1)}$ and $u_2 \in S_T^{(\alpha_2)}$, we have an addition function $\mathsf{add}(\mathsf{params}, T, u_1, u_2) = u_1 + u_2 \in S_T^{(\alpha_1+\alpha_2)}$, and a negation function $\mathsf{neg}(\mathsf{params}, T, u_1) = -u_1 \in S_T^{(-\alpha_1)}$.

Multiplication. For $u_1 \in S_{T_1}^{(\alpha_1)}$, $u_2 \in S_{T_2}^{(\alpha_2)}$ such that $T_1 \cup T_2 \subseteq \mathbb{U}$, we have a multiplication function $\mathsf{mul}(\mathsf{params}, T_1, u_1, T_2, u_2) = u_1 \times u_2 \in S_{T_1 \cup T_2}^{(\alpha_1 \cdot \alpha_2)}$.

Zero-test. The procedure $\mathsf{isZero}(\mathsf{params}, u)$ outputs 1 if $u \in S_{\mathbb{U}}^{(0)}$ and 0 otherwise. Note that in conjunction with the subtraction procedure, this lets us test if $u_1, u_2 \in S_{\mathbb{U}}$ encode the same element $\alpha \in \mathfrak{R}$.

Next, we define two new procedures on graded encodings that we will use:

Extension. This procedure allows extending the graded encoding system by fresh asymmetric levels. Specifically, $\mathsf{extend}(\mathsf{params}, \mathbb{V}, \{e_i\}_i)$ takes as input a set $\mathbb{V} \subseteq \mathbb{U}$ and a sequence of encodings e_i each at level $v_i \subseteq \mathbb{V}$ and outputs a new set $\mathbb{V}'$ and encodings e'_i each at level $v'_i \subseteq \mathbb{V}'$ along with a public transformation function $f_{\mathbb{V}' \to \mathbb{V}}$ such that:-

- Addition and multiplication procedures from above can be applied to encodings at these new levels as well.
- Let $\mathbb{V} = \{1, \dots t\}$ then $\mathbb{V}' = \{1', \dots t'\}$ and for each i we have that if $v_i = \{j_1, \dots j_k\}$ then $v'_i = \{j'_1, \dots j'_k\}$ where $j_1, \dots j_k \in \{1, \dots, t\}$.
- $f_{\mathbb{V}' \to \mathbb{V}}(e', \mathbb{W}')$ takes as input $e' \in S_{\mathbb{W}'}^{(\alpha)}$ where $\mathbb{V}' \subseteq \mathbb{W}'$ and outputs an encoding $e \in S_{\mathbb{V} \cup (\mathbb{W}' \setminus \mathbb{V}')}^{(\alpha)}$.

Extension†. This function $\mathsf{extend}^\dagger$ is the same as the previous function $\mathsf{extend}(\mathsf{params}, \mathbb{V}, \{e_i\}_i)$ except that it also outputs additionally randomizers (encodings of 0) for each level it outputs an encoding at.

In Appendix A, we demonstrate how to obtain the above extension procedures from the GLSW varant of the CLT encodings. We stress that, except for the new extension procedures, all the procedures above are *exactly* the same as in [GLW14]. The extension functions are built *on top* of the underlying graded encoding without any modifications to the existing procedures — in particular, no extra terms are needed in the public parameters.

2.4 Other Cryptographic Primitives

Punctured PRFs. A *punctured pseudorandom function (PRF)* [BW13, BGI14, KPTZ13] is a pseudorandom function PRF where the secret key k for the function can be punctured at an arbitrary input x, arriving at a punctured key k^x. k^x allows the evaluation of PRF at all points other than x: that is, $PRF(k^x, x') = PRF(k, x')$ as long as $x' \neq x$. For security, we require that the pair $(k^x, PRF(k, x))$ is indistinguishable from the pair (k^x, r) where r is chosen at random independent of k.

The original pseudorandom function of Goldreich, Goldwasser, and Micali [GGM84] is puncturable. However, we will need puncturable PRFs that can be evaluated in NC^1, and the GGM construction does not satisfy this requirement. Instead, we will use the PRFs of Boneh, Lewi, Montgomery, and Raghunathan [BLMR13], which are both puncturable and can be evaluated in NC^1.

Randomized Encodings Given a circuit C, a randomized encoding is a pair of functions $\hat{\mathsf{C}}, Rec$. $\hat{\mathsf{C}}(x; r)$ is a randomized function taking the same inputs as C that "encodes" the evaluation of C on input x. Rec takes as input $e = \hat{\mathsf{C}}(x; r)$, and output $\mathsf{C}(x)$.

The goal of randomized encodings is to take a complex circuit C and "encode" the evaluation of C on input x, where the encoding operation is much simpler than evaluating C directly. In our case, C will be an arbitrary polynomial-sized circuit, and we require that $\hat{\mathsf{C}}$ be computable in NC^1.

The security notion we require from randomized encodings is weaker than typically required in the literature. We require that, for two inputs x, x' such that $\mathsf{C}(x) = \mathsf{C}(x')$, that $\hat{\mathsf{C}}(x)$ and $\hat{\mathsf{C}}(x')$ are computationally indistinguishable distributions.

3 Slotted Functional Encryption

In this section, we define the notion of *slotted functional encryption*. Later we will show how this scheme can be used to realize a functional encryption scheme for general circuits. A slotted functional encryption scheme, is roughly a functional encryption with multiple "slots," where each slot roughly serves as an independent copy of the functional encryption scheme. For any ciphertext or secret key, each slot is either active or inactive, and active slots will contain some bit string that potentially varies from slot to slot. Decryption is well-defined only if all slots that are active in both the ciphertext and the secret key agree on the output, in which case the result of decryption is the agreed-upon output. Otherwise, the output is undefined. Slot 0 is a special slot and where the public parameters rest. This is the slot that anyone can encrypt a message to; all the other slots require secret parameters.

- **Setup**(λ, d, C): The setup algorithm takes in the security parameter λ, a number d of slots, and a fixed universal circuit description C as inputs and outputs the public parameters MPK and a master secret key MSK.

- **KeyGen**$(MSK, \mathbf{y})$: The key generation algorithm takes in the master secret key MSK, and a vector of attribute strings $\mathbf{y} \in \{\{0,1\}^n \cup \perp\}^d$ as input. It outputs a private key SK for $\mathbf{y}$.

- **KeyGen**(MSK, y): Alternatively, an unslotted version of the key generation algorithm takes the master secret key MSK, and a single string $y \in \{0,1\}^n$ as input. It runs **KeyGen**$(MSK, \mathbf{y})$ where $\mathbf{y} = (y, \perp, \dots)$.

- **Encrypt**$(MSK, \mathbf{x})$: A private slotted encryption algorithm takes in the secret parameters MSK, and a vector of messages $\mathbf{x} \in \{\{0,1\}^n \cup \perp\}^d$ as input. It outputs a ciphertext C.

- **Encrypt**(MPK, x): a public unslotted encryption algorithm takes in the public parameters MPK, and a single message $x \in \{0,1\}^n$ as input. It outputs an encryption of the message vector $(x, \perp, \perp, ...)$

- **Decrypt**(SK, C): The decryption algorithm takes a private key SK for attribute string $\mathbf{y}$ and a ciphertext C (encrypting say the messages $\mathbf{x}$). Let $S \subseteq [d]$ be the set of *active* indices, namely those $i \in [d]$ where $x[j] \neq \perp$ and $y[j] \neq \perp$. If $\mathsf{C}(x[j], y[j]) = b$ for all active indices $i \in S$, it outputs b. Otherwise, the output is undefined.

We note that, semantically, a slotted functional encryption scheme gives a functional encryption using only the unslotted versions of the KeyGen and Encrypt procedures. Our goal will be to prove security of the derived (unslotted) functional encryption scheme, using various security properties of the full slotted scheme.

For security of slotted FE, consider the following general security game, parameterized by a predicate P:

- **Setup:** The challenger runs the Setup algorithm and gives the public parameters MPK to the attacker. The challenger also flips a random coin $\beta \in \{0, 1\}$, which it keeps secret.
- **Query Phase I:** The attacker adaptively queries the challenger for private keys corresponding to attribute vectors pairs $\mathbf{y}_i^{(0)}, \mathbf{y}_i^{(1)} \in \{\{0,1\}^n \cup \perp\}^d$ for $i = 1, ..., q_1$. The challenger responds with the secret keys for $\mathbf{y}_i^{(\beta)}$.
- **Challenge:** The attacker declares two message s vector $\mathbf{x}^{(0)}, \mathbf{x}^{(1)} \in \{\{0,1\}^n \cup \perp\}^d$. The challenger responds with the ciphertext $C \leftarrow \mathbf{Encrypt}(MSK, \mathbf{x}^{(\beta)})$.
- **Query Phase II:** The attacker continues to adaptively queries the challenger for private keys corresponding to attribute vectors pairs $\mathbf{y}_i^{(0)}, \mathbf{y}_i^{(1)} \in \{\{0,1\}^n \cup \perp\}^d$ for $i = q_1 + 1, ..., q$. The challenger responds with the secret keys for $\mathbf{y}_i^{(\beta)}$.
- **Guess:** The attacker outputs a guess β' for β.
- **Check:** The challenger runs a predicate P on the secret key queries and challenge querie: $c = P(\{\mathbf{y}_i^{(b)}\}_{i \in [q], b \in \{0,1\}}, \mathbf{x}^{(0)}, \mathbf{x}^{(1)})$. If $c = 1$, the challenger outputs $\beta'' = \beta'$. Otherwise if $c = 0$, the challenger outputs a random bit β''.

The advantage of an attacker in this game is defined to be $\Pr[\beta = \beta''] - \frac{1}{2}$.

The security game varies depending on the predicate P. At a minimum P should check that the adversary cannot trivially distinguish the left and right sides by applying the decryption procedure on the secret keys and ciphertext received. P would also need to verify that P cannot distinguish the left and right sides by generating his own ciphertext. Ideally, security should hold for this general P.

However, this security definition is not efficiently checkable: P would have to test all possible vectors $\mathbf{x} = (x, 0, 0, \dots)$ that the adversary could produce himself with the public parameters to make sure the secret keys cannot distinguish.

Moreover, this security notion is too strong. If we just look at the case where $d = 1$ so there is a single slot, the P above would allow changing a secret key y to y' if $\mathsf{C}(x, y) = \mathsf{C}(x, y')$ for all $x \in \{0,1\}^n$. In other words, the scheme would hide the secret key function, thereby implying indistinguishability obfuscation. Specifically, to obfuscate a function f, let $\mathsf{C}(x, f)$ be the universal circuit which evaluates $f(x)$, construct a slotted functional encryption scheme, and then publish the secret key SK_f for attribute f. Anyone can evaluate $f(x)$ for an x of their choice by encrypting x under the scheme, and then using SK_f to evaluate $f(x)$. Finally, if P is the predicate as described above, any two functionally equivalent f and f' would be indistinguishable.

Our goal, then, is to describe simple checks P that are both efficient and do not imply full function hiding. This in principle has similarities with the Gentry et al. constructions of witness encryption [GLW14] and indistinguishability obfuscation [GLSW14] from instance independent assumptions. However, unlike the Genrty et al. construction, we will not require complexity leveraging to turn these simple requirements into full-fledged functional encryption.

3.1 Core Predicates

First, we describe some simple core predicates that we would like the construction of our slotted FE scheme to satisfy. These properties will enable the realization of the adaptively secure FE scheme.

0 **Slot Symmetry**. P checks that there is two slots $\alpha, \beta \in [d] \setminus \{0\}$, $\alpha \neq \beta$, such the queries have the following form:

$b = 0$

	$x[j]$	$y_i[j]$
$j = \alpha$	$x^{(0*)}$	$y_i^{(0*)}$
$j = \beta$	$x^{(1*)}$	$y_i^{(1*)}$
$j \neq \alpha, \beta$	$x[j]$	$y_i[j]$

$b = 1$

	$x[j]$	$y_i[j]$
$j = \alpha$	$x^{(1*)}$	$y_i^{(1*)}$
$j = \beta$	$x^{(0*)}$	$y_i^{(0*)}$
$j \neq \alpha, \beta$	$x[j]$	$y_i[j]$

Intuitively, this allows us to permute the contents of different slots without the adversary's notice.

1 **Single-Use Message and Function Hiding**. P checks that there is a slot $\alpha \in [d], \alpha \neq 0$ and a secret key query $\gamma \in [q]$ such that the queries have the following form:

$b = 0$

	$x[j]$	$y_i[j]$	
		$i = \gamma$	$i \neq \gamma$
$j = \alpha$	$x^{(0*)}$	$y^{(0*)}$	$y_i[\alpha]$
$j \neq \alpha$	$x[j]$	$y_i[j]$	

$b = 1$

	$x[j]$	$y_i[j]$	
		$i = \gamma$	$i \neq \gamma$
$j = \alpha$	$x^{(1*)}$	$y^{(1*)}$	$y_i[\alpha]$
$j \neq \alpha$	$x[j]$	$y_i[j]$	

Requirements:
$\mathsf{C}(x^{(0*)}, y^{(0*)}) = \mathsf{C}(x^{(1*)}, y^{(1*)})$ or
$x^{(0*)} = x^{(1*)} = \perp$ or
$y^{(0*)} = y^{(1*)} = \perp$

This allows us to argue both message and function hiding in any slot which is uniquely used by a ciphertext and a secret key. For example in the above tables slot α is used only in the challenge ciphertext and the γth secret key.

2 **Slot Duplication**. P checks that there is a slot α (possibly 0) and another slot $\beta \neq \alpha, 0$ such that the queries have the following form:

$b = 0$

	$x[j]$	$y_i[j]$
$j = \alpha$	x^*	y_i^*
$j = \beta$	$\perp$	$\perp$
$j \neq \alpha, \beta$	$x[j]$	$y_i[j]$

$b = 1$

	$x[j]$	$y_i[j]$
$j = \alpha$	x^*	y_i^*
$j = \beta$	x^* or $\perp$	y_i^* or $\perp$
$j \neq \alpha, \beta$	$x[j]$	$y_i[j]$

We stress that slot duplication can duplicate the slots of the ciphertext and secret keys *simultaneously*. We can choose to duplicate the slots of all keys and the ciphertext, or any subset of them.

3 **Ciphertext Moving**. P checks that there are slots $\alpha, \beta \in [d], \alpha \neq \beta$ such that the queries have the form:

$b = 0$

	$x[j]$	$y_i[j]$
$j = \alpha$	x^*	y_i^*
$j = \beta$	$\perp$	y_i^*
$j \neq \alpha, \beta$	$x[j]$	$y_i[j]$

$b = 1$

	$x[j]$	$y_i[j]$
$j = \alpha$	$\perp$	y_i^*
$j = \beta$	x^*	y_i^*
$j \neq \alpha, \beta$	$x[j]$	$y_i[j]$

This lets us move ciphertexts between slots provided the secret keys are identical on those slots.

4 **Weak key moving**. P checks that there are slots $\alpha, \beta \in [d], \alpha \neq 0$ and $\beta \neq \alpha, 0$ and secret-key query γ such that the queries have the following form:

$b = 0$

	$x[j]$	$y_i[j]$	
		$i = \gamma$	$i \neq \gamma$
$j = \alpha$	x^*	y^*	$y_i[j]$
$j = \beta$	x^*	$\perp$	
$j \neq \alpha$	$x[j]$	$y_\gamma[j]$	

$b = 1$

	$x[j]$	$y_i[j]$	
		$i = \gamma$	$i \neq \gamma$
$j = \alpha$	x^*	$\perp$	$y_i[j]$
$j = \beta$	x^*	y^*	
$j \neq \alpha$	$x[j]$	$y_\gamma[j]$	

This allows us to to moving the secret key from slot α to slot β as long as the messages encrypted in the two slots are the same.

We observe that the above properties, even in combination, will never allow the changing of a secret key in slot 0. Thus, we will not be able to obtain any form of function hiding for the derived unslotted functional encryption scheme just from the properties above. This serves as a sanity check that the above properties are not too strong, and might be obtainable from simple assumptions, and indeed we give a construction meeting these in Section 4.

3.2 Additional Derivable Predicates

Now we describe several additional properties that can be derived from the core properties above, potentially "using up" several additional slots.

5 **New Slot**. P checks that there are slots $\alpha, \beta \in [d]$ with $\alpha \neq 0$ and $\beta \neq \alpha$ such that the queries have the following form:

$b = 0$

	$x[j]$	$y_i[j]$
$j = \alpha$	$\perp$	$\perp$
$j = \beta$	$x[\beta] \neq \perp$	$y_i[j]$
$j \neq \alpha, \beta$	$x[j]$	

$b = 0$

	$x[j]$	$y_i[j]$
$j = \alpha$	x^*	$\perp$
$j = \beta$	$x[\beta] \neq \perp$	$y_i[j]$
$j \neq \alpha, \beta$	$x[j]$	

This allows us to create new slots with arbitrary messages in them, provided the secret keys are not active in the slot α.

6 **Strong key moving**. P checks that there are slots $\alpha, \beta \in [d], \alpha \neq 0$, $\beta \neq \alpha, 0$, and secret key query $\gamma \in [q]$ such that:

$b = 0$

	$x[j]$	$y_i[j]$	
		$i = \gamma$	$i \neq \gamma$
$j = \alpha$	x_0^*	y^*	$y_i[j]$
$j = \beta$	x_1^*	$\perp$	
$j \neq \alpha$	$x[j]$	$y_\gamma[j]$	

$b = 1$

	$x[j]$	$y_i[j]$	
		$i = \gamma$	$i \neq \gamma$
$j = \alpha$	x_0^*	$\perp$	$y_i[j]$
$j = \beta$	x_1^*	y^*	
$j \neq \alpha$	$x[j]$	$y_\gamma[j]$	

Requirements:
$\mathsf{C}(x_0^*, y^*) = \mathsf{C}(x_1^*, y^*)$

This allows us to actually move secret key component from slot α to slot β even when the messages encrypted in the two slots are not the same.

7 **Weak ciphertext indistinguishability**. P checks that there is a slot $\alpha \in [d], \alpha \neq 0$ such that the queries have the following form:

$b = 0$

	$x[j]$	$y_i[j]$
$j = \alpha$	x_0^*	y_i^*
$j \neq \alpha$	$x[j]$	$y_i[j]$

$b = 0$

	$x[j]$	$y_i[j]$
$j = \alpha$	x_1^*	y_i^*
$j \neq \alpha$	$x[j]$	$y_i[j]$

Requirements:
$\mathsf{C}(x_0^*, y_i^*) = \mathsf{C}(x_1^*, y_i^*) \forall i$

This almost gives us functional encryption, except for the requirement that $\alpha \neq 0$.

8 **Strong ciphertext indist**. Same as above, except α can be 0.

3.3 Reductions

Now we describe several reductions showing that core properties described above are sufficient for obtaining the additional derivable properties also described above, at the cost of "using up" several additional slots. We note that in all of the reductions below, any existing property, whether core or derived, is preserved in the reduction.

Lemma 1. *(1) Single-use hiding and (2) slot duplication imply (5) new slot.*

Proof. Use slot duplication to duplicate contents of the β slot into the originally empty α slot of the ciphertext (don't duplicate the secret keys), and then use single-use message and function hiding to change the message to x^*, which is possible since there are no secret keys components in the α slot. □

Lemma 2. *(1) Single-use hiding, (2) slot duplication, (3) and weak key moving for $d+1$ slots implies (6) strong key moving for d slots (all existing properties being preserved).*

Proof. We prove for $\alpha = 1, \beta = 2$, the other cases being identical. We will move secret key $\gamma \in [q]$. Let slot $d+1$ be a "scratch" slot, that is unused by the normal scheme. We will use slot $d+1$ in the security proof. Below is the table of hybrids. For secret keys $i \in [q], i \neq \gamma$ not included in the table, slot $d+1$ is inactive, and the rest of the slots remain the same throughout all hybrids. Similarly, slots $j \neq 1, 2, d+1$ remain the same for the ciphertext and the γth secret key.

Hybrid	$x[j]$			$y_\gamma[j]$			comments
	$j=1$	$j=2$	$j=d+1$	$j=1$	$j=2$	$j=d+1$	
H_0	x_0^*	x_1^*	$\perp$	y^*	$\perp$	$\perp$	
H_1	x_0^*	x_1^*	x_0^*	y^*	$\perp$	$\perp$	Slot duplication
H_2	x_0^*	x_1^*	x_0^*	$\perp$	$\perp$	y^*	Weak secret key moving
H_3	x_0^*	x_1^*	x_1^*	$\perp$	$\perp$	y^*	Single-use message hiding
H_4	x_0^*	x_1^*	x_1^*	$\perp$	y^*	$\perp$	Weak secret key moving
H_5	x_0^*	x_1^*	$\perp$	$\perp$	y^*	$\perp$	Slot duplication

□

Lemma 3. *(0) Slot symmetry, (5) new slot, and (6) strong key moving for $d+1$ slots implies weak (7) weak ciphertext indistinguishability for d slots (all existing properties being preserved).*

Proof. We prove for $\alpha = 1$, the other cases being identical. The slot $d+1$ will be the "scratch" slot, that is unused by the normal scheme but used in the security proof. In the hybrids below we will use the strong key moving property. Note that the strong key moving only allows for changing one key at a time but in the hybrids below we will need to change all the keys and this can be done by a sequence of hybrids changing one key at a time.

Hybrid	$x[j]$		$\forall \gamma \in [q], y_\gamma[j]$		comments
	$j=1$	$j=d+1$	$j=1$	$j=d+1$	
H_0	x_0^*	$\perp$	y^*	$\perp$	
H_1	x_0^*	x_1^*	y^*	$\perp$	New slot
H_2	x_0^*	x_1^*	$\perp$	y^*	Strong key moving ($\times q$)
H_3	$\perp$	x_1^*	$\perp$	y^*	New slot
H_4	x_1^*	$\perp$	y^*	$\perp$	Slot Symmetry

$\square$

Lemma 4. *(2) Slot duplication, (3) weak ciphertext moving, and (7) weak ciphertext indistinguishability for $d+1$ slots implies (8) strong ciphertext indistinguishability for d slots (all existing properties preserved).*

Proof. Only need to add the case for slot 0. Just as before, the slot $d+1$ will be the "scratch" slot, that is unused by the normal scheme but used in the security proof.

Hybrid	$x[j]$		$y_i[j]$		Comments
	$j=0$	$j=d+1$	$j=0$	$j=d+1$	
H_0	x_0^*	$\perp$	y_i^*	$\perp$	
H_1	x_0^*	$\perp$	y_i^*	y_i^*	Slot duplication
H_2	$\perp$	x_0^*	y_i^*	y_i^*	Weak ciphertext moving
H_3	$\perp$	x_1^*	y_i^*	y_i^*	Weak ciphertext indistinguishability
H_4	x_1^*	$\perp$	y_i^*	y_i^*	Weak ciphertext moving
H_5	x_1^*	$\perp$	y_i^*	$\perp$	Slot duplication

$\square$

4 Slotted Functional Encryption for NC^1

We now give our slotted FE scheme for NC^1. We will describe our scheme in terms of matrix branching programs, and rely on Barrington's Theorem (Theorem 1) to realize slotted FE for NC^1 circuits. We describe our scheme for single bit outputs — it can easily be extended to multi-bit outputs by running multiple instances of the scheme in parallel.

Setup(λ, BP, d)**:** Given a universal 2-input matrix branching program

$$BP = \left(\mathsf{bit}, \mathsf{inp}, (B_{i,b})_{i\in[\ell], b\in\{0,1\}}\right)$$

run $\mathsf{params} \leftarrow \mathsf{InstGen}(1^\lambda, \{1, \dots, \ell\}, d)$. Then, choose random matrices $R_i \in \mathfrak{R}$ for $i \in [\ell-1]$, as well as random $\alpha_{i,b}$ for $i \in [\ell], b \in \{0,1\}$. Let $\tilde{B}_{i,b} = \alpha_{i,b} \cdot R_{i-1} \cdot B_{i,b} \cdot R_i^{-1}$ for $i \in [2, \ell-1]$, and $\tilde{B}_{1,b} = \alpha_{1,b} \cdot B_{1,b} \cdot R_1^{-1}$ and $\tilde{B}_{\ell,b} = \alpha_{\ell,b} \cdot R_{\ell-1} \cdot B_{\ell,b}$[4]. Compute $A_{i,b}^j = [\tilde{B}_{i,b}]_{\{i\}}^j$ for $j \in [d]$. (Here R_0

[4]Using current graded encodings, it is not possible to *publicly* compute matrix inverses since users do not have direct access to the underlying ring. However, the setup procedure would know a trapdoor for the graded encodings that *does* allow computing the matrix inverse. Alternatively, we can replace R_i^{-1} with the *adjugate* matrix R_i^{adj}, encodings of which *can* be computed publicly. The adjugate and matrix inverse only differ by a scalar multiple (namely, the determinant), and since we multiply everything by a random scalar anyway, the distributions of encodings obtained are identical in both approaches.

and R_ℓ are set to identity.)

Let $\mathbb{V}$ be the subset of $[\ell]$ that corresponds to the secret key: $\mathbb{V} = \{i \in [\ell] : \mathsf{inp}(i) = 0\}$, and $\mathbb{W}$ be the subset of $[\ell]$ that corresponds to the ciphertext: $\mathbb{W} = \{i \in [\ell] : \mathsf{inp}(i) = 1\}$. Then the universe $\mathbb{U} = \mathbb{V} \cup \mathbb{W}$.

The master public key is

$$MPK = (\mathsf{params}, (A^0_{i,b})_{i \in \mathbb{W}, b \in \{0,1\}})$$

The master secret key consists of the $A^j_{i,b}$ for $i \in \mathbb{V} \cup \mathbb{W}$.

KeyGen$(MSK, \mathbf{y})$**:** Given an attribute $y \in \{\{0,1\}^n \cup \perp\}^d$, choose random $\beta_i \in \mathfrak{R}$ for $i \in \mathbb{V}, b \in \{0,1\}$, and output the secret key

$$SK_y = \mathsf{extend}\left(\mathsf{params}, \mathbb{V}, \left(\beta_i \cdot \left(\sum_{j : y[j] \neq \perp} A^j_{i, y[j]_{\mathsf{bit}(i)}}\right)\right)_{i \in \mathbb{V}}\right)$$

Encrypt$(MSK, \mathbf{x})$**:** Given an attribute $x \in \{\{0,1\}^n \cup \perp\}^d$, choose random $\beta_i \in \mathfrak{R}$ for $i \in \mathbb{W}, b \in \{0,1\}$, and output the ciphertext

$$C = \mathsf{extend}\left(\mathsf{params}, \mathbb{W}, \left(\beta_i \cdot \left(\sum_{j : x[j] \neq \perp} A^j_{i, x[j]_{\mathsf{bit}(i)}}\right)\right)_{i \in \mathbb{W}}\right)$$

Encrypt(MPK, m)**:** Given a message $m \in \{0,1\}^n$, choose random $\beta_i \in \mathfrak{R}$ for $i \in \mathbb{W}$, and output the ciphertext

$$C = \mathsf{extend}\left(\mathsf{params}, \mathbb{W}, \left(\beta_i \cdot A^0_{i, m_{\mathsf{bit}(i)}}\right)_{i \in \mathbb{W}}\right)$$

Remark 3. *Note that all the encodings given out in the ciphertext can be re-randomized (to noise σ') using the randomizer provided in the public parameters. We do not mention the re-randomization above explicitly, for the sake of simplicity of notation.*

Decrypt(MPK, SK, C)**:** Given a secret key $SK = f_{\mathbb{V}' \to \mathbb{V}}, (K_i)_{i \in \mathbb{V}'}$ and a ciphertext $C = f_{\mathbb{W}' \to \mathbb{W}}, (C_i)_{i \in \mathbb{W}'}$, let $D_i = \begin{cases} K_i & \text{if } i \in \mathbb{V}' \\ C_i & \text{if } i \in \mathbb{W}' \end{cases}$, and compute the product

$$D = f_{\mathbb{V}' \to \mathbb{V}}\left(f_{\mathbb{W}' \to \mathbb{W}}\left(\prod_{i \in \mathbb{U}} D_i\right)\right)$$

Then run the zero-test procedure on a distinguishing coordinate of D.

Correctness. Evaluation is carried out slot by slot. In slot j, if either K or C is inactive, then the corresponding ring will be empty. Therefore, the result of the computation is 0 in slot j.

In a slot j where K and C are both active, then write $K_i[j] = [\beta_i \alpha_{i, y[j]_{\mathsf{bit}(i)}} \tilde{B}_{i, y_{\mathsf{bit}(i)}}]^j_{\{i'\}}$ and $C_i[j] = [\beta_i \alpha_{i, m_{\mathsf{bit}(i)}} \tilde{B}_{i, m_{\mathsf{bit}(i)}}]^j_{\{i'\}}$ for some index elements i' to be the components of K, C in the ring $\mathfrak{R}_j$. Let $d[j] = (y[j], m[j]) \in \{0,1\}^{2n}$. Then we can write

$$D_i[j] = [\beta_i \alpha_{i, d[j]_{\mathsf{inp}(i), \mathsf{bit}(i)}} \tilde{B}_{i, d[j]_{\mathsf{inp}(i), \mathsf{bit}(i)}}]^j_{\{i\}}$$

Therefore, the product $\prod_{i\in\mathbb{U}} D_i[j]$ is equal to

$$\left[\prod_{i\in\mathbb{U}}\left(\beta_i\alpha_{i,d[j]_{\mathsf{inp}(i)},\mathsf{bit}(i)}\right)\prod_{i\in\mathbb{U}}\tilde{B}_{i,d[j]_{\mathsf{inp}(i)},\mathsf{bit}(i)}\right]^j_{\mathbb{U}'}=\left[\prod_{i\in\mathbb{U}}\left(\beta_i\alpha_{i,d[j]_{\mathsf{inp}(i)},\mathsf{bit}(i)}\right)\prod_{i\in\mathbb{U}}B_{i,d[j]_{\mathsf{inp}(i)},\mathsf{bit}(i)}\right]^j_{\mathbb{U}'}$$

Where $\mathbb{U}'=\mathbb{V}'\cup\mathbb{W}'$. Applying $f_{\mathbb{W}'\to\mathbb{W}}$ to this encoding gives an encoding of the same product, but relative to the set $\mathbb{V}'\cup\mathbb{W}$, and then applying $f_{\mathbb{V}'\to\mathbb{V}}$ gives the encoding relative to $\mathbb{U}$. Therefore,

$$D[j]=\left[\prod_{i\in\mathbb{U}}\left(\beta_i\alpha_{i,d[j]_{\mathsf{inp}(i)},\mathsf{bit}(i)}\right)\prod_{i\in\mathbb{U}}B_{i,d[j]_{\mathsf{inp}(i)},\mathsf{bit}(i)}\right]^j_{\mathbb{U}}=\left[\prod_{i\in\mathbb{U}}\left(\beta_i\alpha_{i,d[j]_{\mathsf{inp}(i)},\mathsf{bit}(i)}\right)M_{BP(d[j])}\right]^j_{\mathbb{U}}$$

We only care about ciphertexts and secret keys where the branching program evaluates the same in every slot, so $BP(d[j])$ is the same for all active slots j; call the result b. Define $\gamma[j]=\beta_i\alpha_{i,d[j]_{\mathsf{inp}(i)},\mathsf{bit}(i)}$ projected down to ring $\mathfrak{R}_j$, and $\gamma=\sum_{j\in S}\gamma[j]$ where S is the set of active slots. Note that we only care about secret keys and ciphertext where there is at least one active slot. Therefore with overwhelming probability $\gamma\neq 0$.

We can now write

$$D=[\gamma M_b]_{\mathbb{U}}$$

Then when we zero test a distinguishing coordinate of D, with overwhelming probability, the result will match b.

4.1 Hardness Assumptions

Fix a universe $\mathbb{U}$, a dimension d, and a partition of $\mathbb{U}$ into subsets $\mathbb{V},\mathbb{W}$. For the assumptions below we will assume that randomizers (encodings of zero) are provided for each index in $\mathbb{U}$.

Definition 3 (Assumption 1)**.** *The following distributions are indistinguishable:*

$$\left(\ \left([s_{i,j}]^j_{\{i\}}\right)_{i\in\mathbb{U},j>0},\left([t_i]^1_{\{i\}}\right)_{i\in\mathbb{U}}\ \right)\ \textit{and}\ \left(\ \left([s_{i,j}]^j_{\{i\}}\right)_{i\in\mathbb{U},j>0},\left([t_i]^{0,1}_{\{i\}}\right)_{i\in\mathbb{U}}\ \right)$$

Assumption 1 appears hard because, in order to distinguish the challenge elements, it is required to eliminate the component in $\mathfrak{R}_1$. However, the only way to accomplish this is to pair with one of the $[s_{i,j}]^j_{\{i\}}$ for $j\geq 2$, which will zero out both $\mathfrak{R}_1$ and $\mathfrak{R}_0$.

Definition 4 (Assumption 2)**.** *The following two distributions are indistinguishable:*

$$\Big(\ \left([s_{i,j}]^j_{\{i\}}\right)_{i\in\mathbb{V},j>1},\left([s_i]^j_{\{i\}}\right)_{i\in\mathbb{W},j\in[d]},\left([t_i]^{0,1}_{\{i\}}\right)_{i\in\mathbb{V}},$$
$$\mathsf{extend}^\dagger\left(\mathsf{params},\mathbb{W},\left\{\left([u_{i,j}]^j_{\{i\}}\right)_{i\in\mathbb{W},j>1},\left([v_i]^0_{\{i\}}\right)_{i\in\mathbb{W}}\right\}\ \right)\ \Big)\ \textit{and}$$
$$\Big(\ \left([s_{i,j}]^j_{\{i\}}\right)_{i\in\mathbb{V},j>1},\left([s_i]^j_{\{i\}}\right)_{i\in\mathbb{W},j\in[d]},\left([t_i]^{0,1}_{\{i\}}\right)_{i\in\mathbb{V}},$$
$$\mathsf{extend}^\dagger\left(\mathsf{params},\mathbb{W},\left\{\left([u_{i,j}]^j_{\{i\}}\right)_{i\in\mathbb{W},j>1},\left([v_i]^1_{\{i\}}\right)_{i\in\mathbb{W}}\right\}\ \right)\ \Big)$$

Assumption 2 appears hard because the challenge elements can only be paired with other extended elements, elements in $\mathbb{V}$, or other challenge elements, and the non-challenge extended elements and elements in $\mathbb{V}$ are all identical in $\mathfrak{R}_0$ and $\mathfrak{R}_1$.

4.2 Security Proof

Theorem 3. *Assuming Assumptions 1 and 2, the scheme described above satisfies the core properties of the slotted FE scheme.*

Slot Symmetry. Our scheme satisfies *perfect* slot symmetry, where the advantage of an even infinitely powerful adversary is 0. This follows from the fact that slots correspond to sub-rings in our scheme, and our subrings are generated in a totally symmetric manner.

Single-use Message and Function hiding. In our scheme, the matrices are just the matrices from Kilian-randomized branching programs, where the randomization in each sub-ring is independent. In the single slot j where changes are made, only the ciphertext and a single public key are active. Let $z = (x_0, y_0)$ be the ciphertext and secret key values active on the left side, and $z' = (x_1, y_1)$ be the values on the right side. Then on the left side, only the matrices $\tilde{B}_{i,z[\mathsf{inp}(i)]_{\mathsf{bit}(i)}}$ are handed out in ring $\mathfrak{R}_j$, and by Theorem 2, these matrices are uniform random matrices subject to their product being $M_{\mathsf{C}(x_0,y_0)}$. Similarly, on the left size, the matrices handed out are uniform random matrices subject their product being $M_{\mathsf{C}(x_1,y_1)}$. Since $\mathsf{C}(x_0, y_0) = \mathsf{C}(x_1, y_1)$, these distributions are identical, so our scheme satisfies *perfect* single use hiding.

Slot duplication. We will prove slot duplication from Assumption 1. Let $\alpha \in [d]$ and $\beta \neq \alpha, 0$. Obtain the challenge for assumption 1, and re-order the rings so that the challenge has the form $\left(S_{i,j} = [s_{i,j}]^j_{\{i\}}\right)_{i\in\mathbb{U}, j\neq\beta}, (T_i)_{i\in\mathbb{U}}$ where $T_i = [t_i]^\alpha_{\{i\}}$ or $T_i = [t_i]^{\alpha,\beta}_{\{i\}}$. We now simulate the view of the adversary as follows. Given a 0/1 matrix B and an encoding e, let $e \cdot B$ be the matrix of encodings, where $e \cdot B$ has e in any position where B has a 1, and an encoding of 0 in any position where B has a 0 (note that we will be multipling $e \cdot B$ by other matrices of encodings, so the encodings of 0 do not actually have to be computed, but merely serve as placeholders in the computation).

Choose random matrices $R_i \in \mathfrak{R}$ for $i \in [\ell - 1]$, as well as random $\alpha'_{i,b}$, and set $A^j_{i,b} = \alpha'_{i,b} \cdot R_{i-1} \cdot (S_{i,j} \cdot B_{i,b}) \cdot R_i^{-1}$ for $j \neq \beta$[5]. This formally sets $\alpha_{i,b} = \alpha'_{i,b} s_{i,j}$ in ring $\mathfrak{R}_j$, which leaves $\alpha_{i,b}$ in ring β undetermined. Define $D^j_{i,b} = \alpha'_{i,b} \cdot R_{i-1} \cdot (T_i \cdot B_{i,b}) \cdot R_i^{-1}$.

Using the $A^j_{i,b}$, we can simulate the public paramters as in the scheme. To answer the challenge ciphertext query, there are two cases. If slot β is empty, then we can answer the challenge ciphertext query as in the slotted FE scheme with the $A^j_{i,b}$ (since β is empty, we do not need $A^\beta_{i,b}$). If slot β is not a copy of slot α on either side of the challenge, then we answer the challenge query by choosing a random $\beta'_i \in \mathfrak{R}$ for $i \in \mathbb{W}, b \in \{0, 1\}$, and output the ciphertext

$$C = \mathsf{extend}\left(\mathsf{params}, \mathbb{W}, \left(\beta'_i \cdot \left(\sum_{j:x[j]\neq\perp, j\notin\{\alpha,\beta\}} A^j_{i,x[j]_{\mathsf{bit}(i)}} + D^j_{i,x[\alpha]_{\mathsf{bit}(i)}}\right)\right)_{i\in\mathbb{W}}\right)$$

If the T_i are only encodings in ring $\mathfrak{R}_\alpha$, then this correctly simulates the ciphertext when slot β empty, formally setting $\beta_i = \beta_i$ in rings other that $\mathfrak{R}_\alpha, \mathfrak{R}_\beta$, and setting $\beta_i = \beta'_i t_i$ in rings $\mathfrak{R}_\alpha, \mathfrak{R}_\beta$ (the value in $\mathfrak{R}_\beta$ is irrelevant in this case). If the T_i are encodings in $\mathfrak{R}_\alpha \times \mathfrak{R}_\beta$, then this correctly simulates the ciphertext when slot β is a copy of slot α, with the same formal settings of variables as before.

[5] We actually cannot compute the quantities R_i^{-1} since we do not have access to the trapdoor for the encodings. Therefore, we must actually compute R_i^{adj} instead of R_i^{-1}. However, since we multiply by a random scalar anyway, the distribution of encodings is exactly the same as if we had computed the matrix inverse.

We can perform a similar procedure to simulate the secret key queries. In the end, if T_i are only encodings in $\mathfrak{R}_\alpha$, then this correctly simulates the left side in slot duplication, where slot β is empty. If T_i are encodings in $\mathfrak{R}_\alpha \times \mathfrak{R}_\beta$, then this correctly simulates the right side of slot duplication, where slot β is sometimes a copy of slot α. Thus, if Assumption 1 holds, the two cases are indistinguishable.

Ciphertext moving We will prove ciphertext moving from Assumption 2. Let $\alpha \neq \beta$, where α is the slot the ciphertext is in, and β is the slot we wish to move the ciphertext to. Obtain the challenge for assumption 2, and re-order the rings so that the challenge has the form

$$\left(S_{i,j} = [s_{i,j}]^j_{\{i\}}\right)_{i\in\mathbb{V}, j\notin\{\alpha,\beta\}}, \left(S_{i,j} = [s_{i,j}]^j_{\{i\}}\right)_{i\in\mathbb{W}, j\in[d]}, \left(T_i = [t_i]^{\alpha,\beta}_{\{i\}}\right)_{i\in\mathbb{V}},$$
$$E = \mathsf{extend}^\dagger\left(\mathsf{params}, \mathbb{W}, \left\{\left(U_{i,j} = [u_{i,j}]^j_{\{i\}}\right)_{i\in\mathbb{W}, j>1}, \left(V_i = [v_i]^\gamma_{\{i\}}\right)_{i\in\mathbb{W}}\right\}\right)$$

where $\gamma = \alpha$ or $\gamma = \beta$.

We now simulate the view of the adversary as follows. Choose random matrices $R_i \in \mathfrak{R}$ for $i \in [\ell - 1]$, as well as random $\alpha'_{i,b}$, and set $A^j_{i,b} = \alpha'_{i,b} \cdot R_{i-1} \cdot (S_{i,j} \cdot B_{i,b}) \cdot R_i^{-1}$ for $i \in \mathbb{V}$, $j \notin \{\alpha, \beta\}$, and all $i \in \mathbb{W}$, $j \in [d]$. This formally sets $\alpha_{i,b} = \alpha'_{i,b} s_{i,j}$ in ring $\mathfrak{R}_j$, which leaves $\alpha_{i,b}$ in rings α and β undetermined for $i \in \mathbb{V}$. Define $A^\alpha_{i,b} + A^\beta_{i,b} = \alpha'_{i,b} \cdot R_{i-1} \cdot (T_i \cdot B_{i,b}) \cdot R_i^{-1}$ for $i \in \mathbb{V}$, which formally sets $\alpha_{i,b} = \alpha'_{i,b} T_i$ in rings $\mathfrak{R}_\alpha$ and $\mathfrak{R}_\beta$.

Now using the $A^j_{i,b}$ values, we can simulate the public parameters (since we have all the values for $i \in \mathbb{W}, j = 0$), as well as all the secret key queries (since all the secret key queries are identical in slots α and β, meaning we will always have $A^\alpha_{i,b} + A^\beta_{i,b}$ together, neither being used separately). To generate the challenge ciphertext, we use the result E of extension. Let $U'_{i,j}$ be the components in E corresponding to the $U_{i,j}$, and V'_i the components corresponding to the V_i. Then the challenge ciphertext is set as

$$C = f_{\mathbb{W}' \to \mathbb{W}}, \left(\beta_i \cdot R_{i-1} \cdot \left((V'_i \cdot B_{i,x^*_{\mathsf{bit}(i)}}) + \sum_{j: x[j] \neq \perp, j \notin \{\alpha,\beta\}} (U'_{i,j} \cdot B_{i,x[j]_{\mathsf{bit}(i)}})\right) \cdot R_i^{-1}\right)_{i\in\mathbb{W}}$$

Note that the randomization terms given in E must be used to randomize the components above.

Where x^* is the ciphertext term that is either in slot α or slot β. It is straightforward to show that if the V_i are encodings in $\mathfrak{R}_\alpha$, then this simulates the challenge ciphertext with x^* in slot α, and similarly if V_i are encodings in $\mathfrak{R}_\beta$, the challenge ciphertext has x^* in slot β. Therefore, since the two cases are indistinguishable, ciphertext moving follows.

Weak key moving. This is basically the same as ciphertext moving, except that we swap the roles of $\mathbb{W}$ and $\mathbb{V}$. The main difference is that, because now the public parameters lie in $\mathbb{V}$, and we are not given terms in $\mathbb{V}$ containing α separate from β, we must have $\alpha, \beta \neq 0$ so that we can still generate the public parameters in $\mathfrak{R}_0$.

4.3 Adaptively Secure FE for NC^1

Our slotted FE scheme easily gives adaptively secure FE for NC^1:

Theorem 4. *If assumptions 1 and 2 above hold, then adaptively secure FE for NC^1 exists.*

Proof. Set $d = 4$ in our slotted FE scheme. Then Lemma 1, 2, 3, and 4 gives a slotted scheme with $d = 1$ that satisfies strong ciphertext indistinguishability, which implies adaptive FE security. $\square$

5 Randomized Adaptive Functional Encryption for all Circuits

We now use our slotted FE scheme for NC^1 to build functional encryption for all circuits. Our construction proceeds in two steps:

- First, we build a randomized functional encryption scheme for NC^1. In a randomized FE scheme, the result of decryption is no longer a fixed value $\mathsf{C}(x, y)$, but a (pseudorandom) sample from a distribution determined by x and y: $f(x, y; r)$. Now we allow the secret keys to decrypt the challenge ciphertext differently, but require that the resulting distributions are computationally indistinguishable. This will require puncturable PRFs that can be evaluated in NC^1.

- Second, we will bootstrap the scheme above and obtain a randomized functional encryption scheme for all circuits. This will require a randomized encoding scheme that can be computed in NC^1.

5.1 Slotted FE for NC^1 to Randomized FE for NC^1

We present the definition of a randomized FE scheme, first defined by Goyal et al. [GJKS13]. The semantics of a randomized FE scheme are similar to standard FE, except that the ciphertext x and secret key attribute y no longer define a fixed value $\mathsf{C}(x, y)$, but now define a distribution $\mathsf{C}(x, y; r)$. Correctness is relaxed to requiring that the output of decryption is equal to $\mathsf{C}(x, y; r)$ for some r.

Security is defined by the following experiment:

- **Setup:** The challenger runs the Setup algorithm and gives the public parameters MPK to the attacker.

- **Query Phase I:** The attacker queries the challenger for private keys corresponding to attribute strings $y_1, \ldots, y_{q_1}$, which the challenger provides.

- **Challenge:** The attacker declares two messages x_0, x_1. We require that $\forall i \in [q_1]$ we have that the distributions $\mathsf{C}(x_1, y_i; r)$ and $\mathsf{C}(x_0, y_i; r)$ are computationally indistinguishable. The challenger flips a random coin $\beta \in \{0, 1\}$ and runs $C \leftarrow \mathbf{Encrypt}(MPK, x_\beta)$. The challenger gives the ciphertext C to the adversary.

- **Query Phase II:** The attacker queries the challenger for private keys corresponding to the attribute strings $y_{q_1+1}, \ldots, y_q$, with the added restriction that $\forall i \in \{q_1, \ldots, q\}$ we have that the distributions $\mathsf{C}(x_0, y_i; r)$ and $\mathsf{C}(x_1, y_i; r)$ are computationally indistinguishable.

- **Guess:** The attacker outputs a guess β' for β.

The advantage of an attacker in this game is defined to be $\Pr[\beta = \beta'] - \frac{1}{2}$.

We note that the above security notion is not falsifiable in general; indeed, the condition that $\mathsf{C}(x_1, y_i; r)$ and $\mathsf{C}(x_0, y_i; r)$ be indistinguishable is not even computable. However, in our application, the distributions will be guaranteed to be indistinguishable.

Our Construction. Let ($\mathbf{Setup}', \mathbf{KeyGen}', \mathbf{Encrypt}', \mathbf{Decrypt}'$) be a slotted FE scheme for NC^1 circuits. Let $\mathbf{PRF}, \mathbf{Punct}$ be a puncturable PRF that can be evaluated in NC^1. Let $f(x, y; r)$ be some randomized two-input function that can be evaluated in NC^1. We now give our randomized FE scheme:

Setup(λ, f)**:** Run $\mathbf{Setup}'(\lambda, \mathsf{C}, d)$ for constant d to be chosen later, and where C is defined as:

$$\mathsf{C}(\ (x, k, e_0, b)\ ,\ (y, s, e_1)\ ) = \begin{cases} f(x, y; PRF(k, s)) & \text{if } k \text{ is not punctured at } s \\ e_b & \text{if } k \text{ is punctured at } s \end{cases}$$

KeyGen(MSK, y)**:** Choose a random $s \in \{0,1\}^\lambda$, and define $\mathbf{y} = ((y, s, \epsilon), \perp, \perp, \dots)$, where ϵ is the empty string. Then run $\mathbf{KeyGen}'(MSK, \mathbf{y})$

Encrypt(MPK, x)**:** Choose a random $k \in \{0,1\}^\lambda$, and define $x' = (x, k, \epsilon, 0)$. Then run $\mathbf{Encrypt}'(MPK, x')$.

Decrypt(MPK, SK, C)**:** Run $\mathbf{Decrypt}'(MPK, SK, C)$.

Theorem 5. *If a slotted FE scheme satisfying properties 1 through 7 for $d = 4$ exists, and puncturable PRFs exist that can be evaluated in NC^1, then randomized FE for NC^1 exists.*

Before proving this, we get the following corollary:

Corollary 1. *If assumptions 1 and 2 hold, and puncturable PRFs exist that can be evaluated in NC^1, then randomized functional encryption for NC^1 exists*

Proof. Set $d = 6$. Then applying Lemmas 1, 2, and 3 gives a slotted encryption scheme with $d = 4$ satisfying properties 1 through 7. Together with the puncutrable PRF evaluatable in NC^1 and Theorem 5, the corollary follows. □

We now return to the proof of Theorem 5.

Proof. Our proof follows a sequence of hybrids, given below. We start with the challenge ciphertext encrypting x_0. Then, we "detach" the ciphertext form the public parameters as in the proof of Lemma 4 by copying the secret keys into a new slot (say slot 1), and then moving the challenge ciphertext to this slot. Then, similar to the proof of Lemma 3, we create an additional new slot (say slot 2) in the ciphertext containing x_1, and gradually shift all the secret keys from being in slots 0 and 1 to being in slots 0 and 2. We then eliminate slot 1 (which contains x_0), and finally, we rely on slot symmetry to swap the roles of slots 1 and 2. At the end, the ciphertext encrypts x_1 and all the secret keys are returned to normal.

However, moving the secret keys turns out to be a much more involved task than in the proof of Lemma 3, namely because the result of decrypting the challenge ciphertext with a secret key actually changes when we move the secret key to slot 2, meaning we cannot rely on strong secret key moving. Nonetheless, by carefully combining secret key moving with PRF puncturing, we show that we can, in fact, move the secret keys to slot 2.

Now we present the hybrids:

Hybrid 0. We start with the case where the challenge ciphertext encrypts x_0. Then the ciphertext contains $x_0, k, \epsilon, 0$ in 0, secret key i encrypts (y_i, s_i, ϵ) in 0. Slots $j \geq 1$ are inactive for the ciphertext and all keys.

	$C[j]$	$SK_i[j]$
$j = 0$	$(x_0, k, \epsilon, 0)$	(y_i, s_i, ϵ)
$j = 1, 2, 3$	$\perp$	$\perp$

Hybrid 1. This is identical to Hybrid 0, except that now all the secret keys are active in slots 0 and 1. We move from Hybrid 0 to Hybrid 1 using slot duplication.

	$C[j]$	$SK_i[j]$
$j=0$	$(x_0, k, \epsilon, 0)$	(y_i, s_i, ϵ)
$j=1$	$\perp$	(y_i, s_i, ϵ)
$j=2,3$	$\perp$	$\perp$

Hybrid 2. This is identical to Hybrid 1, except that we "detach" the challenge ciphertext from the public parameters by moving it from slot 0 to slot 1. This is done using ciphertext moving.

	$C[j]$	$SK_i[j]$
$j=0$	$\perp$	(y_i, s_i, ϵ)
$j=1$	$(x_0, k, \epsilon, 0)$	(y_i, s_i, ϵ)
$j=2,3$	$\perp$	$\perp$

Hybrid 3. This is identical to Hybrid 2, except that slot 2 is now active and contains $x_1, k, \epsilon, 0$. This change follows from new slot.

	$C[j]$	$SK_i[j]$
$j=0$	$\perp$	(y_i, s_i, ϵ)
$j=1$	$(x_0, k, \epsilon, 0)$	(y_i, s_i, ϵ)
$j=2$	$(x_1, k, \epsilon, 0)$	$\perp$
$j=3$	$\perp$	$\perp$

Hybrid 4.ℓ Hybrid 4.ℓ is the same has Hybrid 3, except that the first ℓ secret keys are active in slots 0 and 2, whereas the remaining $q-\ell$ secret keys are still active in slots 0 and 1.

	$C[j]$	$SK_i[j] : i \leq \ell$	$SK_i[j] : i > \ell$
$j=0$	$\perp$	(y_i, s_i, ϵ)	(y_i, s_i, ϵ)
$j=1$	$(x_0, k, \epsilon, 0)$	$\perp$	(y_i, s_i, ϵ)
$j=2$	$(x_1, k, \epsilon, 0)$	(y_i, s_i, ϵ)	$\perp$
$j=3$	$\perp$	$\perp$	$\perp$

The ciphertexts are different in these slots, and the result of C may be different (though indistinguishable), so we cannot perform these hybrid steps directly using strong key moving and instead need additional hybrids.

For $\ell \leq q_1$ (i.e., the secret key queries before the challenge ciphertext is provided), this is relatively easy:

Hybrid 4.ℓ.1$^{\ell \leq q_1}$ This is identical to Hybrid 4.$(\ell-1)$, except that the PRF key k in the ciphertext is punctured at the ℓth secret key tag, namely s_ℓ. Moreover, the value $f_\ell = f(x_0, y_\ell, PRF(k, s_\ell)) = \mathsf{C}(\ (x_0, k, \epsilon, 0)\ ,\ (y_\ell, s_\ell, \epsilon)\ )$ is hard-coded into the e_0 component of the challenge ciphertext (since the challenge ciphertext comes after the secret key here, we will know the value of f_ℓ when generating the challenge ciphertext). Lastly, the indicator bit b is set to 0, telling C it should use the value hard-coded in e_0 as the output when needed.

Since $s_i \neq s_\ell$ for all $i \neq \ell$, puncturing at s_ℓ does not affect the evaluation of C for secret keys other than ℓ. Moreover, f_ℓ is set to the value that C outputted on the encryption of x_0 *before* puncturing, so this puncturing does not affect the evaluation of secret key ℓ in slot 1. Lastly, secret key ℓ is not active in slot 2. Therefore, wee move from Hybrid 4.$(\ell-1)$ to 4.ℓ.1$^{\ell \leq q_1}$ using two invocations of weak ciphertext indistinguishability, once for slot 1 and once for slot 2.

	$C[j]$	$SK_i[j] : i < \ell$	$SK_\ell[j]$	$SK_i[j] : i > \ell$
$j = 0$	$\perp$	(y_i, s_i, ϵ)	$(y_\ell, s_\ell, \epsilon)$	(y_i, s_i, ϵ)
$j = 1$	$(x_0, k^{s_\ell}, f(x_0, y_\ell, PRF(k, s_\ell)), 0)$	$\perp$	$(y_\ell, s_\ell, \epsilon)$	(y_i, s_i, ϵ)
$j = 2$	$(x_1, k^{s_\ell}, f(x_0, y_\ell, PRF(k, s_\ell)), 0)$	(y_i, s_i, ϵ)	$\perp$	$\perp$
$j = 3$	$\perp$	$\perp$	$\perp$	$\perp$

Hybrid 4.ℓ.2$^{\ell \leq q_1}$ This is the same as Hybrid 4.ℓ.1$^{\ell \leq q_1}$, except that we replace $PRF(k, s_\ell)$ with a random r. The punctured PRF security of PRF shows that this change is indistinguishable. Now f_ℓ is a fresh sample from the distribution $f(x_0, y_\ell)$.

	$C[j]$	$SK_i[j] : i < \ell$	$SK_\ell[j]$	$SK_i[j] : i > \ell$
$j = 0$	$\perp$	(y_i, s_i, ϵ)	$(y_\ell, s_\ell, \epsilon)$	(y_i, s_i, ϵ)
$j = 1$	$(x_0, k^{s_\ell}, f(x_0, y_\ell; r), 0)$	$\perp$	$(y_\ell, s_\ell, \epsilon)$	(y_i, s_i, ϵ)
$j = 2$	$(x_1, k^{s_\ell}, f(x_0, y_\ell; r), 0)$	(y_i, s_i, ϵ)	$\perp$	$\perp$
$j = 3$	$\perp$	$\perp$	$\perp$	$\perp$

Hybrid 4.ℓ.3$^{\ell \leq q_1}$ This is the same as Hybrid 4.ℓ.2$^{\ell \leq q_1}$, except that we replace f_ℓ with a random sample from $f(x_1, y_\ell)$, relying on the indistinguishability of samples.

	$C[j]$	$SK_i[j] : i < \ell$	$SK_\ell[j]$	$SK_i[j] : i > \ell$
$j = 0$	$\perp$	(y_i, s_i, ϵ)	$(y_\ell, s_\ell, \epsilon)$	(y_i, s_i, ϵ)
$j = 1$	$(x_0, k^{s_\ell}, f(x_1, y_\ell; r), 0)$	$\perp$	$(y_\ell, s_\ell, \epsilon)$	(y_i, s_i, ϵ)
$j = 2$	$(x_1, k^{s_\ell}, f(x_1, y_\ell; r), 0)$	(y_i, s_i, ϵ)	$\perp$	$\perp$
$j = 3$	$\perp$	$\perp$	$\perp$	$\perp$

Hybrid 4.ℓ.4$^{\ell \leq q_1}$ This is the same as Hybrid 4.ℓ.3$^{\ell \leq q_1}$, except that we move the ℓth secret key from slots 0 and 1 to slots 0 and 2. Since the ciphertext is punctured at s_ℓ in slots 1 and 2, when decrypting with the ℓth secret key, the hard-coded value f_ℓ will be outputted in both slots. Therefore, we can rely on strong secret key moving to make this change.

	$C[j]$	$SK_i[j] : i < \ell$	$SK_\ell[j]$	$SK_i[j] : i > \ell$
$j = 0$	$\perp$	(y_i, s_i, ϵ)	$(y_\ell, s_\ell, \epsilon)$	(y_i, s_i, ϵ)
$j = 1$	$(x_0, k^{s_\ell}, f(x_1, y_\ell; r), 0)$	$\perp$	$\perp$	(y_i, s_i, ϵ)
$j = 2$	$(x_1, k^{s_\ell}, f(x_1, y_\ell; r), 0)$	(y_i, s_i, ϵ)	$(y_\ell, s_\ell, \epsilon)$	$\perp$
$j = 3$	$\perp$	$\perp$	$\perp$	$\perp$

Hybrid 4.ℓ.5$^{\ell \leq q_1}$ This is the same as Hybrid 4.ℓ.3$^{\ell \leq q_1}$, except that we replace r with $PRF(k, s_\ell)$, relying on punctured PRF security.

	$C[j]$	$SK_i[j] : i < \ell$	$SK_\ell[j]$	$SK_i[j] : i > \ell$
$j = 0$	$\perp$	(y_i, s_i, ϵ)	$(y_\ell, s_\ell, \epsilon)$	(y_i, s_i, ϵ)
$j = 1$	$(x_0, k^{s_\ell}, f(x_1, y_\ell; PRF(k, s_\ell)), 0)$	$\perp$	$\perp$	(y_i, s_i, ϵ)
$j = 2$	$(x_1, k^{s_\ell}, f(x_1, y_\ell; PRF(k, s_\ell)), 0)$	(y_i, s_i, ϵ)	$(y_\ell, s_\ell, \epsilon)$	$\perp$
$j = 3$	$\perp$	$\perp$	$\perp$	$\perp$

Hybrid 4.ℓ for $\ell \leq q_1$ We obtain Hybrid 4.ℓ for $\ell \leq q_1$ from Hybrid 4.ℓ.5$^{\ell \leq q_1}$ by unpuncturing the PRF key in slots 1 and 2 of the ciphertext. This is obtained in a similar manner to the transition from Hybrid 4.$(\ell - 1)$ to Hybrid 4.ℓ.1$^{\ell \leq q_1}$: we apply weak message indistinguishability twice, once in each slot. Since the puncturing only affects the evaluation using the ℓth secret key, and slot 1 is inactive for key ℓ, we can unpuncture in slot 1. Key ℓ is active in slot 2, but the correct value

is hard-coded in the challenge ciphertext, so unpuncturing does not affect the final outcome of the evaluation.

	$C[j]$	$SK_i[j] : i < \ell$	$SK_\ell[j]$	$SK_i[j] : i > \ell$
$j = 0$	$\perp$	(y_i, s_i, ϵ)	$(y_\ell, s_\ell, \epsilon)$	(y_i, s_i, ϵ)
$j = 1$	$(x_0, k, \epsilon, 0)$	$\perp$	$\perp$	(y_i, s_i, ϵ)
$j = 2$	$(x_1, k, \epsilon, 0)$	(y_i, s_i, ϵ)	$(y_\ell, s_\ell, \epsilon)$	$\perp$
$j = 3$	$\perp$	$\perp$	$\perp$	$\perp$

For $\ell > q_1$, i.e. secret key queries after the challenge, things are harder, since we can no longer embed the result in the ciphertext, and must instead use the secret key. However, we do not have any form of secret key indistinguishability (as this would imply iO), so the argument is a bit more involved.

Hybrid 4.ℓ.1$^{\ell > q_1}$ This is identical to Hybrid 4.$(\ell - 1)$, except that we copy slot one of the ciphertext into a new slot, slot 3. This is obtained from Hybrid 4.$(\ell - 1)$ using new slot in slot 3, or slot duplication.

	$C[j]$	$SK_i[j] : i < \ell$	$SK_\ell[j]$	$SK_i[j] : i > \ell$
$j = 0$	$\perp$	(y_i, s_i, ϵ)	$(y_\ell, s_\ell, \epsilon)$	(y_i, s_i, ϵ)
$j = 1$	$(x_0, k, \epsilon, 0)$	$\perp$	$(y_\ell, s_\ell, \epsilon)$	(y_i, s_i, ϵ)
$j = 2$	$(x_1, k, \epsilon, 0)$	(y_i, s_i, ϵ)	$\perp$	$\perp$
$j = 3$	$(x_0, k, \epsilon, 0)$	$\perp$	$\perp$	$\perp$

Hybrid 4.ℓ.2$^{\ell > q_1}$ This is identical to Hybrid 4.ℓ.1$^{\ell > q_1}$, except that we move the secret key from slots 0 and 1 to slots 0 and 3. Since the ciphertext is identical in slots 1 and 3, we accomplish this using weak secret key moving.

	$C[j]$	$SK_i[j] : i < \ell$	$SK_\ell[j]$	$SK_i[j] : i > \ell$
$j = 0$	$\perp$	(y_i, s_i, ϵ)	$(y_\ell, s_\ell, \epsilon)$	(y_i, s_i, ϵ)
$j = 1$	$(x_0, k, \epsilon, 0)$	$\perp$	$\perp$	(y_i, s_i, ϵ)
$j = 2$	$(x_1, k, \epsilon, 0)$	(y_i, s_i, ϵ)	$\perp$	$\perp$
$j = 3$	$(x_0, k, \epsilon, 0)$	$\perp$	$(y_\ell, s_\ell, \epsilon)$	$\perp$

Hybrid 4.ℓ.3$^{\ell > q_1}$ This is identical to 4.ℓ.2$^{\ell > q_1}$, except that the PRF key k in slots 1 and 2 of the ciphertext is punctured at the ℓth secret key tag, namely s_ℓ. Since secret key ℓ is non-existent in slots 1 and 2, this follows from two applications of weak ciphertext indistinguishability.

	$C[j]$	$SK_i[j] : i < \ell$	$SK_\ell[j]$	$SK_i[j] : i > \ell$
$j = 0$	$\perp$	(y_i, s_i, ϵ)	$(y_\ell, s_\ell, \epsilon)$	(y_i, s_i, ϵ)
$j = 1$	$(x_1, k^{s_\ell}, \epsilon, 0)$	$\perp$	$\perp$	(y_i, s_i, ϵ)
$j = 2$	$(x_2, k^{s_\ell}, \epsilon, 0)$	(y_i, s_i, ϵ)	$\perp$	$\perp$
$j = 3$	$(x_1, k, \epsilon, 0)$	$\perp$	$(y_\ell, s_\ell, \epsilon)$	$\perp$

Hybrid 4.ℓ.4$^{\ell > q_1}$ This is identical to 4.ℓ.3$^{\ell > q_1}$, except that the PRF key k in slot 3 of the chiphertext is punctured at s_ℓ. Moreover, the value $f_\ell = f(x_0, y_\ell, PRF(k, s_\ell)) = \mathsf{C}(\ (x_0, k, \epsilon, 0)\ ,\ (y_\ell, s_\ell, \epsilon)\ )$ is hard-coded into the e_1 component of slot 3 of the ℓth secret key (since the challenge ciphertext comes before the secret key here, we will know the value of f_ℓ when generating the secret key). Lastly, the indicator bit b in slot 3 is set to 1, telling C it should use the value hard-coded in e_1 as the output when needed. These changes only affect slot 3, which is only present in the ciphertext

and ℓth secret key. Moveover, because the correct value is hard-coded in the secret key, the output of C does not change. Therefore, we can rely on single-use hiding to make this transition.

	$C[j]$	$SK_i[j] : i < \ell$	$SK_\ell[j]$	$SK_i[j] : i > \ell$
$j = 0$	$\perp$	(y_i, s_i, ϵ)	$(y_\ell, s_\ell, \epsilon)$	(y_i, s_i, ϵ)
$j = 1$	$(x_0, k^{s_\ell}, \epsilon, 0)$	$\perp$	$\perp$	(y_i, s_i, ϵ)
$j = 2$	$(x_1, k^{s_\ell}, \epsilon, 0)$	(y_i, s_i, ϵ)	$\perp$	$\perp$
$j = 3$	$(x_0, k^{s_\ell}, \epsilon, 1)$	$\perp$	$(y_\ell, s_\ell, f(x_0, y_\ell; PRF(k, s_\ell))$	$\perp$

Hybird 4.ℓ.5$^{\ell > q_1}$ This is identical to Hybrid 4.ℓ.4$^{\ell > q_1}$, except that we replace $PRF(k, s_\ell)$ with a random r. Indistinguishability follows from the punctured PRF security of PRF. This amounts to replacing f_ℓ with a fresh random sample from $f(x_0, s_\ell)$.

	$C[j]$	$SK_i[j] : i < \ell$	$SK_\ell[j]$	$SK_i[j] : i > \ell$
$j = 0$	$\perp$	(y_i, s_i, ϵ)	$(y_\ell, s_\ell, \epsilon)$	(y_i, s_i, ϵ)
$j = 1$	$(x_0, k^{s_\ell}, \epsilon, 0)$	$\perp$	$\perp$	(y_i, s_i, ϵ)
$j = 2$	$(x_1, k^{s_\ell}, \epsilon, 0)$	(y_i, s_i, ϵ)	$\perp$	$\perp$
$j = 3$	$(x_2, k^{s_\ell}, \epsilon, 1)$	$\perp$	$(y_\ell, s_\ell, f(x_0, y_\ell; r))$	$\perp$

Hybrid 4.ℓ.6$^{\ell > q_1}$ This is identical to Hybrid 4.ℓ.5$^{\ell > q_1}$, except that we eplace f_ℓ with a sample from $f(x_1, s_\ell)$. Indistinguishability follows from the indistinguishability of the samples.

	$C[j]$	$SK_i[j] : i < \ell$	$SK_\ell[j]$	$SK_i[j] : i > \ell$
$j = 0$	$\perp$	(y_i, s_i, ϵ)	$(y_\ell, s_\ell, \epsilon)$	(y_i, s_i, ϵ)
$j = 1$	$(x_0, k^{s_\ell}, \epsilon, 0)$	$\perp$	$\perp$	(y_i, s_i, ϵ)
$j = 2$	$(x_1, k^{s_\ell}, \epsilon, 0)$	(y_i, s_i, ϵ)	$\perp$	$\perp$
$j = 3$	$(x_0, k^{s_\ell}, \epsilon, 1)$	$\perp$	$(y_\ell, s_\ell, f(x_1, y_\ell; r))$	$\perp$

Hybrid 4.ℓ.7$^{\ell > q_1}$ This is identical to Hybrid 4.ℓ.6$^{\ell > q_1}$, except that we replace r with $PRF(k, s_i))$; indistinguishability follows from the punctured PRF security of PRF.

	$C[j]$	$SK_i[j] : i < \ell$	$SK_\ell[j]$	$SK_i[j] : i > \ell$
$j = 0$	$\perp$	(y_i, s_i, ϵ)	$(y_\ell, s_\ell, \epsilon)$	(y_i, s_i, ϵ)
$j = 1$	$(x_0, k^{s_\ell}, \epsilon, 0)$	$\perp$	$\perp$	(y_i, s_i, ϵ)
$j = 2$	$(x_1, k^{s_\ell}, \epsilon, 0)$	(y_i, s_i, ϵ)	$\perp$	$\perp$
$j = 3$	$(x_0, k^{s_\ell}, \epsilon, 1)$	$\perp$	$(y_\ell, s_\ell, f(x_1, y_\ell; PRF(k, s_\ell))$	$\perp$

Hybrid 4.ℓ.8$^{\ell > q_1}$ This is identical to Hybrid 4.ℓ.7$^{\ell > q_1}$, except for the following modification in slot 3: unpuncture k in the ciphertext, replace x_0 with x_1, and the remove hard-coding in secret key. That is, ciphertext now encrypts $(x_2, k, \epsilon, 0)$ in both slots 2 and 3, and secret key i has (y_i, s_i, ϵ) in slots 1 and 3. When the secret key ℓ decrypts the challenge ciphertext, the output is still $f(x_1, y_\ell; PRF(k, s_\ell))$, so the output remains unchanged. Thus this modification is made using single-use hiding.

	$C[j]$	$SK_i[j] : i < \ell$	$SK_\ell[j]$	$SK_i[j] : i > \ell$
$j = 0$	$\perp$	(y_i, s_i, ϵ)	$(y_\ell, s_\ell, \epsilon)$	(y_i, s_i, ϵ)
$j = 1$	$(x_0, k^{s_\ell}, \epsilon, 0)$	$\perp$	$\perp$	(y_i, s_i, ϵ)
$j = 2$	$(x_1, k^{s_\ell}, \epsilon, 0)$	(y_i, s_i, ϵ)	$\perp$	$\perp$
$j = 3$	$(x_1, k, \epsilon, 0)$	$\perp$	$(y_\ell, s_\ell, \epsilon)$	$\perp$

Hybrid 4.ℓ.9$^{\ell>q_1}$ This is identical to Hybrid 4.ℓ.8$^{\ell>q_1}$, except that we unpuncture the PRF key k in the ciphertext in slots 1 and 2, using two applications of weak ciphertext indistinguishability.

	$C[j]$	$SK_i[j] : i < \ell$	$SK_\ell[j]$	$SK_i[j] : i > \ell$
$j = 0$	$\perp$	(y_i, s_i, ϵ)	$(y_\ell, s_\ell, \epsilon)$	(y_i, s_i, ϵ)
$j = 1$	$(x_0, k, \epsilon, 0)$	$\perp$	$\perp$	(y_i, s_i, ϵ)
$j = 2$	$(x_1, k, \epsilon, 0)$	(y_i, s_i, ϵ)	$\perp$	$\perp$
$j = 3$	$(x_1, k, \epsilon, 0)$	$\perp$	$(y_\ell, s_\ell, \epsilon)$	$\perp$

Hybrid 4.ℓ.10$^{\ell>q_1}$ This is identical to Hybrid 4.ℓ.9$^{\ell>q_1}$, except that we move secret key ℓ to from slots 0 and 3 to slots 0 and 2 using weak key moving.

	$C[j]$	$SK_i[j] : i < \ell$	$SK_\ell[j]$	$SK_i[j] : i > \ell$
$j = 0$	$\perp$	(y_i, s_i, ϵ)	$(y_\ell, s_\ell, \epsilon)$	(y_i, s_i, ϵ)
$j = 1$	$(x_0, k, \epsilon, 0)$	$\perp$	$\perp$	(y_i, s_i, ϵ)
$j = 2$	$(x_1, k, \epsilon, 0)$	(y_i, s_i, ϵ)	$(y_\ell, s_\ell, \epsilon)$	$\perp$
$j = 3$	$(x_1, k, \epsilon, 0)$	$\perp$	$\perp$	$\perp$

Hybrid 4.ℓ for $\ell > q_1$ We arrive at Hybrid 4.ℓ for $\ell > q_1$ from 4.ℓ.10$^{\ell>q_1}$ by deactivating slot 3 in the ciphertext. This is done using new slot or slot duplication.

	$C[j]$	$SK_i[j] : i < \ell$	$SK_\ell[j]$	$SK_i[j] : i > \ell$
$j = 0$	$\perp$	(y_i, s_i, ϵ)	$(y_\ell, s_\ell, \epsilon)$	(y_i, s_i, ϵ)
$j = 1$	$(x_1, k, \epsilon, 0)$	$\perp$	$\perp$	(y_i, s_i, ϵ)
$j = 2$	$(x_2, k, \epsilon, 0)$	(y_i, s_i, ϵ)	$(y_\ell, s_\ell, \epsilon)$	$\perp$
$j = 3$	$\perp$	$\perp$	$\perp$	$\perp$

Hybrid 4.q Setting $\ell = q$, we now have that all the secret keys are in slots 0 and 2. We finish off the proof by making a few more hybrid steps.

	$C[j]$	$SK_i[j]$
$j = 0$	$\perp$	(y_i, s_i, ϵ)
$j = 1$	$(x_0, k, \epsilon, 0)$	$\perp$
$j = 2$	$(x_1, k, \epsilon, 0)$	(y_i, s_i, ϵ)
$j = 3$	$\perp$	$\perp$

Hybrid 5 This is identical to Hybrid 4.q, except that we deactive slot 1 of the ciphertext. This is accomplished using new slot.

	$C[j]$	$SK_i[j]$
$j = 0$	$\perp$	(y_i, s_i, ϵ)
$j = 1$	$\perp$	$\perp$
$j = 2$	$(x_1, k, \epsilon, 0)$	(y_i, s_i, ϵ)
$j = 3$	$\perp$	$\perp$

Hybrid 6 This is identical to Hybrid 5, except that we move the ciphertext to slot 0 using ciphertext moving.

	$C[j]$	$SK_i[j]$
$j = 0$	$(x_1, k, \epsilon, 0)$	(y_i, s_i, ϵ)
$j = 1$	$\perp$	$\perp$
$j = 2$	$\perp$	(y_i, s_i, ϵ)
$j = 3$	$\perp$	$\perp$

Hybrid 7 Finally, this hybrid is identical to Hybrid 6, except that we deactivate slot 2 of the secret keys using slot duplication. At this point, we have an encryption of x_1.

	$C[j]$	$SK_i[j]$
$j=0$	$(x_1, k, \epsilon, 0)$	(y_i, s_i, ϵ)
$j=1$	$\perp$	$\perp$
$j=2$	$\perp$	$\perp$
$j=3$	$\perp$	$\perp$

Through this sequence of hybrids, we have shown that Hybrid 0, which encrypts x_0, is indistinguishable from Hybrid 7, which encrypts x_1. This completes the proof.

□

5.2 Randomized adaptive FE for NC^1 to FE for all circuits

Let $(\mathbf{Setup}', \mathbf{KeyGen}', \mathbf{Encrypt}', \mathbf{Decrypt}')$ be an adaptive FE scheme for randomized NC^1 circuits. For an arbitrary polynomial-sized circuit C, let $\hat{\mathsf{C}}(x, y; s)$ be a randomized encoding for the evaluation of C on inputs x, y, and Rec the corresponding reconstruction function such that $Rec(\hat{\mathsf{C}}(x, y; s)) = \mathsf{C}(x, y)$. We require that $\hat{\mathsf{C}}$ can be evaluated in NC^1.

We now give our construction of functional encryption for all circuits.

Setup(λ, C)**:** Run $\mathbf{Setup}'(\lambda, \hat{\mathsf{C}})$.

KeyGen(MSK, y)**:** Run $\mathbf{KeyGen}'(MSK, \mathbf{y})$

Encrypt(MPK, x)**:** $\mathbf{Encrypt}'(MPK, x)$.

Decrypt(MPK, SK, C)**:** Run $e \leftarrow \mathbf{Decrypt}'(MPK, SK, C)$, and then output $Rec(e)$

Correctness follows from the correctness of the underlying randomized FE scheme and the correctness of the randomized encodings.

Theorem 6. *If* $(\mathbf{Setup}', \mathbf{KeyGen}', \mathbf{Encrypt}', \mathbf{Decrypt}')$ *is a randomized adaptive FE for* NC^1 *circuits,* $\hat{\mathsf{C}}$ *is a randomized encoding for* C*, then the construction above is an adaptive FE for all circuits*

Proof. Given an adversary $\mathcal{A}$ for the adaptive FE scheme above, we will construct an adversary $\mathcal{B}$ for the underlying randomized adaptive FE scheme that simulates $\mathcal{A}$, playing the role of FE challenger. When $\mathcal{B}$ receives the public parameters, it forwards them to $\mathcal{A}$. When $\mathcal{A}$ makes a secret key query on attribute y, $\mathcal{B}$ makes a secret key query on the same attribute y, and gives the resulting key to $\mathcal{A}$. When $\mathcal{A}$ makes a challenge on messages (x_0, x_1), $\mathcal{B}$ makes the same challenge, and forwards the resulting challenge ciphertext to $\mathcal{A}$. When $\mathcal{A}$ makes a guess b', $\mathcal{B}$ outputs the guess.

It is straightforward to see that $\mathcal{B}$ perfectly simulates the view of $\mathcal{A}$, and also that $\mathcal{B}$ has the same advantage in breaking the randomized FE security as $\mathcal{A}$ does in breaking FE security. It remains, then, to show that $\mathcal{B}$ makes legal queries. Indeed, $\mathcal{A}$ is restricted to queries such that $\mathsf{C}(x_0, y_i) = \mathsf{C}(x_1, y_i)$ for all secret key queries i. Therefore, by the security of the randomized encodings, $\hat{\mathsf{C}}(x_0, y_i; r)$ is indistinguishable form $\hat{\mathsf{C}}(x_1, y_i; r)$, and so $\mathcal{B}$ makes valid queries. Therefore, $\mathcal{B}$ breaks the security of the underlying randomized adaptive FE scheme, a contradiction. □

References

[Bar86] D A Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc_1. In *STOC*, 1986.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014: 17th International Workshop on Theory and Practice in Public Key Cryptography*, volume 8383 of *Lecture Notes in Computer Science*, pages 501–519, Buenos Aires, Argentina, March 26–28, 2014. Springer, Berlin, Germany.

[BGK+14] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 221–238, Copenhagen, Denmark, May 11–15, 2014. Springer, Berlin, Germany.

[BLMR13] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic PRFs and their applications. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 410–428, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Berlin, Germany.

[BR14] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In Yehuda Lindell, editor, *TCC 2014: 11th Theory of Cryptography Conference*, volume 8349 of *Lecture Notes in Computer Science*, pages 1–25, San Diego, CA, USA, February 24–26, 2014. Springer, Berlin, Germany.

[BS02] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. Cryptology ePrint Archive, Report 2002/080, 2002. `http://eprint.iacr.org/2002/080`.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *TCC 2011: 8th Theory of Cryptography Conference*, volume 6597 of *Lecture Notes in Computer Science*, pages 253–273, Providence, RI, USA, March 28–30, 2011. Springer, Berlin, Germany.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology – ASIACRYPT 2013, Part II*, volume 8270 of *Lecture Notes in Computer Science*, pages 280–300, Bengalore, India, December 1–5, 2013. Springer, Berlin, Germany.

[CLT13] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 476–493, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Berlin, Germany.

[Gen06] Craig Gentry. Practical identity-based encryption without random oracles. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 445–464, St. Petersburg, Russia, May 28 – June 1, 2006. Springer, Berlin, Germany.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 1–17, Athens, Greece, May 26–30, 2013. Springer, Berlin, Germany.

[GGH+13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual Symposium on Foundations of Computer Science*, pages 40–49, Berkeley, CA, USA, October 26–29, 2013. IEEE Computer Society Press.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. On the cryptographic applications of random functions. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology – CRYPTO'84*, volume 196 of *Lecture Notes in Computer Science*, pages 276–288, Santa Barbara, CA, USA, August 19–23, 1984. Springer, Berlin, Germany.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 467–476, Palo Alto, CA, USA, June 1–4, 2013. ACM Press.

[GJKS13] Vipul Goyal, Abhishek Jain, Venkata Koppula, and Amit Sahai. Functional encryption for randomized functionalities. Cryptology ePrint Archive, Report 2013/729, 2013. `http://eprint.iacr.org/2013/729`.

[GLSW14] Craig Gentry, Allison Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. Cryptology ePrint Archive, Report 2014/309, 2014. `http://eprint.iacr.org/2014/309`.

[GLW14] Craig Gentry, Allison B. Lewko, and Brent Waters. Witness encryption from instance independent assumptions. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 426–443, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Berlin, Germany.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *20th Annual ACM Symposium on Theory of Computing*, pages 20–31, Chicago, Illinois, USA, May 2–4, 1988. ACM Press.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13: 20th Conference on Computer and Communications Security*, pages 669–684, Berlin, Germany, November 4–8, 2013. ACM Press.

[O'N10] Adam O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010. `http://eprint.iacr.org/2010/556`.

[PST14] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 500–517, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Berlin, Germany.

[Wat05] Brent R. Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 114–127, Aarhus, Denmark, May 22–26, 2005. Springer, Berlin, Germany.

[Wat09] Brent Waters. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 619–636, Santa Barbara, CA, USA, August 16–20, 2009. Springer, Berlin, Germany.

[Wat14] Brent Waters. A punctured programming approach to adaptively secure functional encryption. Cryptology ePrint Archive, Report 2014/588, 2014. `http://eprint.iacr.org/`.

A Instantiation of Graded Encoding Scheme

In this section we briefly recall the CLT encodings using description taken essentially verbatim from [GLW14]. We adapt the construction to include the new *extension* functionality that our scheme crucially relies on.

A.1 Overview of CLT Encodings

CLT encodings have a couple of properties that make them more attractive in our setting than the original multilinear maps of Garg, Gentry and Halevi (GGH) [GGH13a]. First, as Garg et al. noted in their paper, GGH encodings are subject to a weak discrete log attack. This attack can be avoided by working with multilinear jigsaw puzzle pieces [GGH$^+$13b] consisting of matrices of encodings (rather than individual encodings). However, we find it simpler to work with CLT encodings, which (as far as we know) do not seem to be vulnerable to this attack in the first place. Second, GGH encodings are built for a prime-order encoding space. While it is probably relatively straightforward to modify GGH encodings to support a composite-order encoding space, we prefer to work with CLT encodings, which inherently support a composite integer encoding space already. Unfortunately, the translation from composite order groups to CLTs composite order encoding space is not quite as direct as one would like - the most "direct" translation is subject to attacks, as discuss in [CLT13, Section B.6] - but it is still relatively straightforward.

A κ-linear symmetric CLT encoding system uses a "small" inner modulus $N = p_1 \dots p_s$ that is the product of $s = s(\lambda, \kappa)$ "small" primes, and a "large" outer modulus $Q = P_1 \dots P_s$ that is the product of s "large" primes. It uses a random $z \leftarrow \mathbb{Z}_Q^*$. An encoding $c \in S_1^{(m)}$ is an element of $\mathbb{Z}_Q$ such that

$$c \equiv \frac{[m]_{p_i} + c_i \cdot p_i}{z} \mod P_i \text{ for } i \in [s], \tag{1}$$

where $[m]_{p_i}$ is m reduced modulo p_i into a small range such as $(-p_i/2, p_i/2)$, and the x_i's are random small integers. An encoding in S_κ has a similar form, but with z^κ in the denominator.

For random small integers $h_1, \dots, h_s$, the system includes a zero-testing parameter p_{zt} for level κ of the form:

$$p_{zt} = \sum_{i=1}^{s} h_i \cdot (z^\kappa \cdot p_i^{-1}) \cdot \prod_{j \neq i} P_j \mod Q.$$

If c is a level-κ encoding of $0 \in \mathbb{Z}_N$ - i.e., each $[m]_{p_i} = 0$ - we have:

$$\begin{aligned} c \cdot p_{zt} &= \sum_{i=1}^{s} (x_i \cdot p_i / z^{\kappa}) \cdot h_i \cdot (z^{\kappa} \cdot p_i^{-1}) \cdot \prod_{j \neq i} P_j \mod Q \\ &= \sum_{i=1}^{s} x_i \cdot h_i \cdot \prod_{j \neq i} P_j \mod Q \end{aligned}$$

which is a number substantially smaller than Q assuming the x_i's and h_i's satisfy certain smallness constraints - in particular, that each $x_i \cdot h_i \ll P_i$. On the other hand, if c encodes something other than 0, $c \cot p_{zt}$ likely will not be a small number, due to uncanceled p_i^1's in the expression above. Thus, p_{zt} enables zero-testing. (Actually, CLT uses a polynomial number of such zero-testing parameters, and they prove that c encodes 0 if it passes the tests with respect to all of them, and does not encode 0 otherwise.)

By CRT, we can add and multiply CLT encodings while preserving their form (per Equation 1) as long as the numerators in Equation 1 do not grow too large - i.e., they do not "wrap" modulo P_i for any i. The P_i's must be chosen large enough to ensure that such wrapping never occurs for the functions we will compute over the encodings. These additions and multiplications induce additions and multiplications on the underlying "messages" that are encoded, much like homomorphic encryption.

Asymmetric settings. Like GGH, CLT generalizes easily to allow asymmetric graded encodings. The simplest way to build asymmetric multilinear CLT encodings is simply to generate a random $z_i \leftarrow \mathbb{Z}_Q^*$ for each asymmetric group, rather than a single z. For $i \in [\kappa]$, an encoding in $S_i^{(m)}$ now has the form

$$c \equiv \frac{[m]_{p_i} + c_i \cdot p_i}{z_i} \mod P_i \text{ for } i \in [s], \tag{2}$$

The form of the zero-test parameter changes to:

$$p_{zt} = \sum_{i=1}^{s} h_i \cdot ((\prod_{i \in [\kappa]} z_i) \cdot p_i^{-1}) \cdot \prod_{j \neq i} P_j \mod Q.$$

Similar to the symmetric case, multiplying p_{zt} with an encoding in $S_T^{(0)}$ (which has $\prod_{i \in [\kappa]} z_i$ in the denominator) results in a mod-Q number that is small relative to Q.

Intuitively, the asymmetric form of the encodings limits how a user can meaningfully multiply together encodings, so that each monomial it computes corresponds to multiplying together exactly one encoding from each source group, so that it obtains an encoding with $\prod_{i \in [\kappa]} z_i$ in the denominator. For example, the multilinear map cannot be used directly to solve decision Diffie-Hellman over elements in S_1, since this would involve multiplying together encodings from S_1, which would induce an uncancellable z_1^2 in the denominator.

In the asymmetric setting the construction can naturally be translated to a setting where the levels are described as sets rather than just a number as described in Definition 2.

Composite order setting. Finally we want to be able to encode subrings of $\mathbb{Z}_N$ with CLT encodings. Unfortunately as described in [GLW14, Section B.6] it is not safe, to give an encoding of some m that is in the index-p_i subring of $\mathbb{Z}_N$. However, GLW present a simple way to fix the problem. They avoid letting any p_i be "isolated" by giving it many - i.e., $poly(\lambda)$ - "buddies": any encoding that an attacker sees is 0 modulo p_i and all of its prime buddies $\{p_j\}$, or is (whp) nonzero for all of them. As we discuss in [GLW14, Section B.6], this approach seems resilient to attacks. We will not provide further details on specific parameters needed for the implementation of this scheme and refer the reader to [GLW14, Section B.4] for more details.

A.2 Implementing the Extension Functionality

Now we are ready to describe how the CLT graded encoding scheme can be extended to support the extension functionality that we need. Recall that, we need to realize the function $\mathsf{extend}(\mathsf{params}, \mathbb{V}, \{e_i\}_i)$ that takes as input a set $\mathbb{V} \subseteq \mathbb{U}$ and a sequence of encodings e_i each at level $v_i \subseteq \mathbb{V}$ and outputs a new set $\mathbb{V}'$ and encodings e'_i at appropriate levels $v'_i \subseteq \mathbb{V}'$ such that if $\mathbb{V} = \{1, \ldots t\}$ then $\mathbb{V}' = \{1', \ldots t'\}$ and for each i we have that if $v_i = \{j_1, \ldots j_k\}$ then $v'_i = \{j'_1, \ldots j'_k\}$ where $j_1, \ldots j_k \in \{1, \ldots, t\}$.

For each $i \in \mathbb{V}$ sample a fresh $z'_i \leftarrow \mathbb{Z}_Q^*$ subject to the constraint that $\prod_{i \in V} z'_i = 1$ and translate each encoding e_i at level v_i to $e'_i = \frac{e_i}{\prod_{j \in v_i} z'_j}$.

Note that we also need to generate the description of the function $f_{\mathbb{V}' \to \mathbb{V}}(e', \mathbb{W}')$ that takes as input $e' \in S^{(\alpha)}_{\mathbb{W}'}$ where $\mathbb{V}' \subseteq \mathbb{W}'$ and outputs an encoding $e \in S^{(\alpha)}_{\mathbb{V} \cup (\mathbb{W}' \setminus \mathbb{V}')}$. Since $\prod_{i \in V} z'_i = 1$ therefore we note that just the identity function serves the purpose of $f_{\mathbb{V}' \to \mathbb{V}}$.

Finally note that the $\mathsf{extend}^\dagger$ function also outputs additionally randomizers (encodings of 0) for each level it outputs an encoding at. This can be achieved by generating encodings of 0 at levels v'_i and then taking random linear combinations.