Multi-Input Functional Encryption

S. Dov Gordon* Jonathan Katz† Feng-Hao Liu‡ Elaine Shi§ Hong-Sheng Zhou¶

Abstract

Functional encryption (FE) is a powerful primitive enabling fine-grained access to encrypted data. In an FE scheme, secret keys ("tokens") correspond to functions; a user in possession of a ciphertext $\mathsf{ct} = \mathsf{Enc}(x)$ and a token TK_f for the function f can compute $f(x)$ but learn nothing else about x. An active area of research over the past few years has focused on the development of ever more expressive FE schemes.

In this work we introduce the notion of *multi-input* functional encryption. Here, informally, a user in possession of a token TK_f for an n-ary function f and *multiple* ciphertexts $\mathsf{ct}_1 = \mathsf{Enc}(x_1)$, ..., $\mathsf{ct}_n = \mathsf{Enc}(x_n)$ can compute $f(x_1, \ldots, x_n)$ but nothing else about the $\{x_i\}$. Besides introducing the notion, we explore the feasibility of multi-input FE in the public-key and symmetric-key settings, with respect to both indistinguishability-based and simulation-based definitions of security.

*ACS. Email: sgordon@appcomsci.com.

†Department of Computer Science, University of Maryland. Email: jkatz@cs.umd.edu. This research was sponsored by NSF award #1223623 and by the US Army Research Laboratory and the UK Ministry of Defence under Agreement Number W911NF-06-3-0001.

‡Department of Computer Science, University of Maryland. Email: fenghao@cs.umd.edu.

§Department of Computer Science, University of Maryland. Email: elaine@cs.umd.edu. This work is partially supported by NSF award CNS-1314857 and a Google Research Award.

¶Department of Computer Science, Virginia Commonwealth University. Email: hszhou@vcu.edu. This work is partially supported by an NSF CI postdoctoral fellowship, and was mostly done while at the University of Maryland.

Contents

1 Introduction

Traditional encryption schemes provide rather coarse-grained access to encrypted data: given a ciphertext $\mathsf{ct} = \mathsf{Enc}(x)$, a user in possession of the appropriate secret key is authorized to learn x in its entirety, whereas a user who does not hold the secret key learns nothing about the underlying data x. *Functional encryption* [10, 21, 17], which extends the earlier notion of predicate encryption (aka, attribute-based encryption) [25, 11, 6, 27], offers much more fine-grained control. In a functional encryption (FE) scheme, secret keys—which we will refer to as "tokens"—are associated with *functions*; roughly speaking, a user who holds a token TK_f associated with the function f and a ciphertext $\mathsf{ct} = \mathsf{Enc}(x)$ can learn $f(x)$ but nothing else about x. (Predicate encryption corresponds to the special case in which a user learns all of x if and only if $f(x) = 1$.) The past few years have seen steady progress toward constructing predicate encryption schemes [24, 19, 18, 20] and FE schemes [31, 23, 21, 20] with greater expressiveness and collusion resistance, culminating in the recent work of Garg et al. [17] showing an FE scheme for arbitrary functions with security against an attacker who obtains an unbounded number of tokens.

Functional encryption applies to *unary* functions taking a single input. Here, we introduce the notion of *multi-input functional encryption*, a generalization to the case of n-ary functions. Informally, a token TK_f for the n-ary function f allows a user who obtains ciphertexts $\mathsf{ct}_1 = \mathsf{Enc}(x_1), \ldots, \mathsf{ct}_n = \mathsf{Enc}(x_n)$ to compute $f(x_1, \ldots, x_n)$ while learning nothing else about the underlying data $x_1, \ldots, x_n$[1]. Beyond serving as a natural extension of FE, multi-input FE also has several important applications that we discuss below.

The usual setting for functional encryption is the public-key one, in which anyone (including the adversary) can generate ciphertexts. While multi-input FE can be defined in such a setting, it is not hard to see that the security that can be obtained in this case is relatively weak. Specifically (simplifying to the case of a two-input function f), an attacker who holds TK_f and $\mathsf{Enc}(x_1)$ for some unknown x_1 can always generate, on its own, ciphertexts $\mathsf{ct}_2^{(1)} = \mathsf{Enc}(x_2^{(1)}), \ldots, \mathsf{ct}_2^{(\ell)} = \mathsf{Enc}(x_2^{(\ell)})$ for known $x_2^{(1)}, \ldots, x_2^{(\ell)}$, and then use TK_f to learn $f(x_1, x_2^{(1)}), \ldots, f(x_1, x_2^{(\ell)})$. In general, this can reveal a significant amount of information about x_1.

On the other hand, multi-key FE can be very useful in the *symmetric-key* setting where a user holding a token may not be able to encrypt new messages and so the above attack no longer applies. In this setting, we can hope to achieve a notion of security which is the natural counterpart of what can be achieved in the unary case [32]: possession of tokens $\mathsf{TK}_{f_1}, \ldots, \mathsf{TK}_{f_\ell}$ corresponding to the functions $f_1, \ldots, f_\ell$ allows a user who obtains ciphertexts $\mathsf{ct}_1 = \mathsf{Enc}(x_1), \ldots, \mathsf{ct}_n = \mathsf{Enc}(x_n)$ to learn $f_1(x_1, \ldots, x_n), \ldots, f_\ell(x_1, \ldots, x_n)$ but nothing else about the $\{x_i\}$ (but see footnote 1). In the symmetric-key setting we actually consider two variant models: a "basic" setting in which there is a single encryption key, and a "multi-client" setting in which there are n secret keys $\mathsf{usk}_1, \ldots, \mathsf{usk}_n$ and the token TK_f can only be used to compute $f(x_1, \ldots, x_n)$ when given n ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_n$ such that $\mathsf{ct}_i = \mathsf{Enc}_{\mathsf{usk}_i}(x_i)$. The second notion is natural in a setting in which there are multiple senders; by enforcing an ordering among the inputs $\{x_i\}$, it also addresses the issue raised in footnote 1.

1.1 Our Results

Besides introducing the notion of multi-input FE, we initiate a comprehensive study of its feasibility. We consider both the public-key and private-key settings. Although, as discussed above, the public-key setting appears rather limited, we include a consideration of this setting both for completeness and because it provides a useful stepping stone to our results in the symmetric-key case.

As in the case of standard FE [10], we consider both indistinguishability-based and simulation-based

[1] For the purposes of the present discussion, we assume f is symmetric so the order of its inputs does not matter. In the general case we allow the user to learn $f(x_{i_1}, \ldots, x_{i_n})$ for every permutation of the inputs.

notions of security. With respect to the indistinguishability-based definition, we show constructions of multi-input FE for arbitrary functions (in both the public-key and symmetric-key settings) based on any *indistinguishability obfuscator* $i\mathcal{O}$ as defined in [3, 22] and constructed in [17]. With regard to simulation-based security in the public-key setting, we are already faced with impossibility results [10, 1, 5] for realizing general functions even in the unary case. We extend these results to show that if a function family supports simulation-based security for multi-input FE in the public-key setting, then the functions are "learnable" in a sense we define. We also demonstrate that this is tight, providing a construction for "learnable" function families achieving simulation-based security in the public key setting. This construction can also be extended to the symmetric key setting, in this case supporting arbitrary functions. Our constructions are based on *virtual black-box (VBB) obfuscation*. Although VBB obfuscation for all functions is impossible in the standard model [3], it was recently shown to be possible in a strengthened version of the generic-group model [13, 4]. Our results are subject to the same caveats as theirs, and we view our constructions as initial feasibility results that can be viewed as secure against "generic" attacks. On a more positive note, we also show that we can use the compiler of Caro et al. [14] to avoid the assumption of VBB obfuscation, though at the expense of only achieving bounded-message/bounded-key security.

In the indistinguishability-based setting, we note that our construction from $i\mathcal{O}$ has succinct ciphertexts, but the runtime of the encryption and decryption protocols grow with the number of challenge plaintexts. This is because the crs in our public parameters is forced to grow with the number of simulated proofs. However, if the underlying obfuscation protocol has differing-input security [3, 2, 12], we can achieve runtime that is fully independent of the number of ciphertexts. Interestingly, even with this stronger assumption, we only know how to achieve selective security. We introduce a new notion of indistinguishability-obfuscation that we call *strong differing-inputs obfuscation*, and prove that this assumption suffices for achieving the desired efficiency together with adaptive security. See Remark 2.5 and Appendix D for more details.

1.2 Applications of Multi-Input FE

We briefly mention some applications of multi-input FE, focusing on the symmetric-key setting. We describe these applications informally as motivation for the notion of multi-input FE, and omit any formal consideration of these applications.

Order-preserving and property-preserving encryption. Multi-input symmetric-key FE can be used for searching over encrypted data, where it can function in the same role as *order-preserving encryption* (OPE) [8, 9] or, more generally, property-preserving encryption [28]. Specifically, consider a setting in which a client uploads several encrypted data items $\mathsf{ct}_1 = \mathsf{Enc}(x_1), \ldots, \mathsf{ct}_n = \mathsf{Enc}(x_n)$ to a server. If, at some later point in time, the client wants to retrieve all data items less than some value t, the client can send $\mathsf{ct}^* = \mathsf{Enc}(t)$ along with a token TK_f for the (binary) comparison function. This allows the server to identify exactly which data items are less than the desired threshold t (and send the corresponding ciphertexts back to the client), without learning anything beyond the relative ordering of the data items. The search query itself remains hidden as well. A direct application of our construction yields the *first* OPE scheme to satisfy the indistinguishability notion of security proposed by Boldyreva et al. [8], resolving the primary open question in that line of research.

In fact, we can hide even more information than OPE: if the client tags every data item with a '0' (i.e., uploads $\mathsf{ct}_i = \mathsf{Enc}(0\|x_i)$) and tags the search term with a '1' (i.e., sends $\mathsf{ct}^* = \mathsf{Enc}(1\|t)$), then the client can send TK_f for the function

$$f(b\|x, b'\|t) = \begin{cases} x < t & b = 0,\ b' = 1 \\ 0 & \text{otherwise} \end{cases}$$

Thus, TK_f allows comparisons only between the data items and the threshold, but not between the data items themselves. More generally, the same approach can be used to enable arbitrary searches over encrypted data while revealing only a minimal amount of information. In this sense, our results generalize property-preserving encryption [28] to arbitrary functions.

Streaming verifiable computation. We can also apply multi-input symmetric-key FE to verifiable computation, using the ideas of [29]. Let f be a predicate that a client wishes to outsource, and define a function f^* such that $f^*(r_1\|\mathsf{ind}_1\|x_1, \ldots, r_n\|\mathsf{ind}_n\|x_n)$ outputs r_1 if (1) $r_1 = \cdots = r_n$, (2) $\mathsf{ind}_i = i$ for all i, and (3) $f(x_1, \ldots, x_n) = 1$ (and $\perp$ otherwise). The client can compute and send TK_{f^*} to the server in a pre-processing phase. Then, to evaluate $f(x_1, \ldots, x_n)$ the client chooses a random nonce r and sends $\mathsf{Enc}(r\|i\|x_i)$ for $i = 1, \ldots, n$; the server returns r iff $f(x_1, \ldots, x_n) = 1$. (As in [29], the scheme can be run a second time using the negation of f so the client can verify when the result is 0.) Although, as described, this could be achieved with unary FE (by simply having the client send $\mathsf{Enc}(x_1\| \cdots \|x_n)$), an advantage of multi-input FE is that it applies naturally to a *streaming* setting where the client may never hold all the $\{x_i\}$ at any one time. We remark also that by using a multi-client scheme we can achieve multi-client verifiable computation [15] with security against malicious clients.

Multi-client data aggregation. Consider a setting in which n senders (e.g., sensors or network monitors) collect data and report it to a central server. The data from the various senders is encrypted, yet the server should be able to compute some function of the reported results. An additional challenge here is that a subset of the senders may be compromised, possibly colluding with the server, yet the corrupted parties should still learn nothing about the data reported by any honest senders (other than the allowed function computed over all the results). Our notion of multi-client, symmetric-key FE (MC-FE) can be used in exactly this scenario. (A similar notion was proposed by Shi et al. [33], and several potential applications are discussed there.)

1.3 Informal Overview of Our Constructions

Intuition. Our construction is inspired by the technique used by Garg et al. [17] for constructing (unary) FE from $i\mathcal{O}$. We focus on achieving binary FE for concreteness. In our construction, plaintexts are encrypted using a standard public-key encryption scheme. The token for a function f consists of an indistinguishability obfuscation of a function that decrypts two ciphertexts using the master secret key, evaluates f on the resulting plaintexts, and outputs the result. Unfortunately, as described, this scheme is insecure since $i\mathcal{O}$ does not guarantee the secrecy of the decryption key that is hard-coded in the $i\mathcal{O}$.

We modify the above by encrypting each plaintext *twice*, and adding a statistically simulation-sound NIZK (SSS-NIZK) proof that the two ciphertexts encrypt the same plaintext x. While real-world ciphertexts encrypt consistent copies of the same plaintext (x, x), in the proof of security we use a hybrid experiment in which ciphertexts instead consist of encryptions of two different messages (x, y) and a simulated NIZK proof of consistency. The main difficulty is to argue that, even given these "simulated" ciphertexts, an $i\mathcal{O}$ that uses the 1st decryption key to decrypt is indistinguishable from an $i\mathcal{O}$ that uses the 2nd decryption key to decrypt. To make this argument, we need to show that these two $i\mathcal{O}$s are functionally equivalent, i.e., they always agree on the output for any input. As we will see, in the public-key setting this argument is quite straightforward. Counter-intuitively, it is the private key setting (both the basic setting, and the multi-client setting) that is more challenging, requiring several changes to the construction just described. We will summarize these below, after providing intuition for the proof in the simpler public-key setting.

The public-key setting. Fix a (two-input) function f and challenge plaintext pairs x_1, x_2 and y_1, y_2 with $f(x_1, \cdot) = f(y_1, \cdot)$, and $f(\cdot, x_2) = f(\cdot, y_2)$. (This restriction is inherent for any meaningful definition of security in the public-key setting.) The adversary is given TK_f and either $\mathsf{ct}_1 = (\mathsf{Enc}(x_1), \mathsf{Enc}(x_1), \pi_1)$

and $\mathsf{ct}_2 = (\mathsf{Enc}(x_2), \mathsf{Enc}(x_2), \pi_2)$, or $\mathsf{ct}_1 = (\mathsf{Enc}(y_1), \mathsf{Enc}(y_1), \pi_1)$ and $\mathsf{ct}_2 = (\mathsf{Enc}(y_2), \mathsf{Enc}(y_2), \pi_2)$. The token TK_f is an $i\mathcal{O}$ of a circuit that takes two FE ciphertexts (each consisting of a pair of ciphertexts plus an NIZK proof), verifies the NIZK proofs, decrypts the first ciphertext in each FE ciphertext, and computes f on the resulting plaintexts. In a hybrid experiment, we modify TK_f to a TK'_f that decrypts and uses the *second* ciphertext in each of the FE ciphertexts. As part of arguing that this hybrid world is indistinguishable from the real experiment, we have to argue that TK_f and TK'_f have identical input/output behavior. If we restrict ourselves to well-formed inputs, the claim is trivial. To handle adversarially generated ciphertexts, we rely on the properties of SSS-NIZK to ensure that the only existing "ill-formed" ciphertexts with valid proofs are precisely the challenge ciphertexts themselves. Since $f(x_1, \cdot) = f(y_1, \cdot)$, and $f(\cdot, x_2) = f(\cdot, y_2)$, it holds that the outputs of TK_f and TK'_f are equivalent for all inputs.

The symmetric-key setting. Counter-intuitively, the public-key setting turns out to be easier precisely because of the limitation that $f(x_1, \cdot) = f(y_1, \cdot)$ and $f(\cdot, x_2) = f(\cdot, y_2)$ for the challenge plaintext pairs, which is not present in the symmetric-key setting. In the symmetric key setting, in contrast, we only require that $f(x_1, x_2) = f(y_1, y_2)$. However, now when we consider the ill-formed ciphertext from the hybrid world, say $\mathsf{ct}_1 = (\mathsf{Enc}(x_1), \mathsf{Enc}(y_1), \pi')$, we can no longer claim that TK_f and TK'_f output the same value on all inputs. In fact, there likely exists some msg such that $f(x_1, \mathsf{msg}) \neq f(y_1, \mathsf{msg})$. We therefore modify the construction as follows. We place a statistically binding commitment to some random value α in the public parameters. We modify the encryption algorithm to include α, along with a NIZK proof that this is the value in the commitment. Then, during decryption, the $i\mathcal{O}$ circuit in TK_f checks the proof, and checks that the value α is consistent across both ciphertexts. As before, if the two inputs are well-formed, it is trivial to see that TK_f and TK'_f have the same input/output behavior. When one (or both) ciphertexts are ill-formed, we again turn to the SSS property of the NIZK, and the constraint described above. Using the SSS property, we ensure that the only ill-formed ciphertexts with accepting NIZKs are precisely those that we need in our hybrid proof, all of which will use some fixed $\alpha' \neq \alpha$, along with a simulated proof of consistency with the committed value. Now, consider the outputs of TK_f and TK'_f when (at least) one input is an ill-formed ciphertext from the hybrid game, say $\mathsf{ct}_1 = (\mathsf{Enc}(x_1), \mathsf{Enc}(y_1), \pi'_1, \alpha')$. If the second ciphertext is well-formed, both circuits will output $\perp$ because of the failed check of whether $\alpha' = \alpha$. If the second ciphertext is also from the hybrid world, say of the form $\mathsf{ct}_2 = (\mathsf{Enc}(x_2), \mathsf{Enc}(y_2), \pi'_2, \alpha')$, then both ciphertexts use α' and both TK_f and TK'_f will proceed to compute f. TK_f will output $f(x_1, x_2)$ while TK'_f will output $f(y_1, y_2)$, but now we can rely on the constraint that these values are equal for all legitimate challenge plaintexts.

The multi-client symmetric-key setting. In this setting, the adversary is allowed to corrupt some set $\bar{G}$. Our restriction on the adversary is that for challenge vectors $\vec{x}_G$ and $\vec{y}_G$, $f(\vec{x}_G, \cdots) = f(\vec{y}_G, \cdots)$, where $\vec{x}_G$ and $\vec{y}_G$ correspond to the plaintexts by the uncorrupted parties, and $\cdots$ denotes the plaintexts corresponding to the corrupted parties.

Recall that in the aforementioned single-client, symmetric-key setting, the sender must have a secret value α to encrypt. However, here we cannot give a single α to each party since the adversary can corrupt a subset of the parties. Instead, we would like to give each party their own α_i.

As before, there is a hybrid world in which the challenger must encrypt as $(\mathsf{Enc}(\vec{x}_G), \mathsf{Enc}(\vec{y}_G))$ in the two parallel encryptions. Later, in order for us to switch the decryption key in the $i\mathcal{O}$ from sk to sk', the two $i\mathcal{O}$'s (using sk and sk' respectively) must be functionally equivalent. To achieve this functional equivalence, we must prevent mix-and-match of simulated and honest ciphertexts. In the earlier single-client, symmetric-key setting, this is achieved by using a fake α value in the simulation, and verifying that all ciphertexts input into the $i\mathcal{O}$ must have the same α value. In the multi-client setting, a simple equality check no longer suffices, so we need another way to prevent mix-and-match of hybrid ciphertexts with well-formed ciphertexts. We do this by choosing a random vector $\vec{\beta}_G$ such that $\langle \vec{\beta}_G, \vec{\alpha}_G \rangle = 0$. We hard-code $\vec{\beta}_G$ in the $i\mathcal{O}$, and if the $\vec{\alpha}_G$ values in the ciphertexts are not orthogonal to $\vec{\beta}_G$, the $i\mathcal{O}$ will simply output $\perp$.

In the hybrid world, instead of using the honest vector $\vec{\alpha}_G$, the simulator uses another random $\vec{\alpha}'_G$ orthogonal to $\vec{\beta}_G$, and simulates the NIZKs. In this way, a mixture of honest and simulated ciphertexts for the set G will cause the $i\mathcal{O}$ to simply output $\perp$, since mixing the coordinates of $\vec{\alpha}_G$ and $\vec{\alpha}'_G$ will result in a vector *not* orthogonal to $\vec{\beta}_G$ (except with negligible probability over the choice of these vectors). In this way, except with negligible probability over the choice of these vectors, using either sk or sk' to decrypt in the $i\mathcal{O}$ will result in exactly the same input and output behavior.

Finally, in order for us to obtain faster encryption and decryption time, instead of encoding $\vec{\alpha}_G$ directly in the ciphertexts, we use a generator for a group that supports the Diffie-Hellman assumption and encode $g^{\vec{\alpha}_G}$ instead. As we will show later, this enables the simulator to simulate fewer NIZKs. In fact, with this trick, the simulator only needs to simulate NIZKs for the challenge time step alone. Therefore, the CRS and the time to compute ciphertexts will be independent of the number of time steps. The details appear in Section 2.3.

2 Indistinguishability-Based Security

In this section we present indistinguishability-based (IND-security) definitions and constructions for the public key, symmetric key, and multi-client settings. For simplicity, we focus on the binary-input setting in both the public key and symmetric key settings. However, we remark that our construction extend naturally to the multi-input setting. The only modification is to make the $i\mathcal{O}$ circuit accept more ciphertexts as inputs, and compute the function f over all decrypted values. The proof follows in a straightforward manner.

2.1 The Public-Key Setting

2.1.1 Definitions

Let $\mathcal{F} = \{\mathcal{F}_\kappa\}_{\kappa>0}$ be a collection of function families, where every $f \in \mathcal{F}_\kappa$ is a polynomial time function $f : \mathcal{M}_\kappa \times \mathcal{M}_\kappa \to \mathcal{M}'_\kappa$, where $\mathcal{M}_\kappa = \{0,1\}^{\mu(\kappa)}$, $\mathcal{M}'_\kappa = \{0,1\}^{\mu'(\kappa)}$, and μ, μ' are polynomial. A public-key binary FE scheme supporting $\mathcal{F}$ is a collection of algorithms: $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval})$. The first three algorithms are probabilistic, and Eval is deterministic. They have the following semantics:

Setup: $(\mathsf{msk}, \mathsf{param}) \leftarrow \mathsf{Setup}(1^\kappa)$

KeyGen: for any $f \in \mathcal{F}_\kappa$, $\mathsf{TK}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$

Enc: $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{param}, \mathsf{msg})$

Eval: $\mathsf{ans} \leftarrow \mathsf{Eval}(\mathsf{param}, \mathsf{TK}_f, \mathsf{ct}_1, \mathsf{ct}_2)$

Correctness. The correctness property states that with overwhelming probability over the randomness used in Setup, KeyGen, and Enc, for all $f \in \mathcal{F}$ and all messages $x, y \in \mathcal{M}$,

$$\mathsf{Eval}(\mathsf{param}, \mathsf{KeyGen}(\mathsf{msk}, f), \mathsf{Enc}(\mathsf{param}, x), \mathsf{Enc}(\mathsf{param}, y)) = f(x, y).$$

Adaptive security. Here we define the full IND-security notion, referred to as adaptive security, for the public key setting. We say that a binary FE scheme is adaptively, multi-message (resp. single-message), IND-secure if for all PPT, *non-trivial, stateful* adversaries $\mathcal{A}$, for a uniformly chosen $b \leftarrow \{0,1\}$, $\Pr[b' = b] \leq \frac{1}{2} + \mathsf{negl}(\kappa)$ for some negligible function $\mathsf{negl}(\cdot)$ in the following multi-message (resp. single-message) experiment:

adaptive IND-security:

1. $(\mathsf{msk}, \mathsf{param}) \leftarrow \mathsf{Setup}(1^\kappa)$
2. $b \leftarrow \{0,1\}$
3. $(x_1, y_1) \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\cdot)}(\mathsf{param}, 1^\kappa)$.
4. For $i = 1$ to m:
   - if $b = 0$: $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{param}, x_i)$, else: $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{param}, y_i)$.
   - $(x_{i+1}, y_{i+1}) \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\cdot)}(\mathsf{ct}_i)$.
5. $b' \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\cdot)}$

adaptive single-message IND-security:

1. $(\mathsf{msk}, \mathsf{param}) \leftarrow \mathsf{Setup}(1^\kappa)$
2. $b \leftarrow \{0,1\}$
3. $(x, y) \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\cdot)}(\mathsf{param}, 1^\kappa)$.
4. if $b = 0$: $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{param}, x)$, else: $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{param}, y)$.
5. $b' \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\cdot)}(\mathsf{ct})$

An adversary is considered *non-trivial* if for every query f made to the $\mathsf{KeyGen}(\cdot)$ oracle, for any polynomial m and for all $i \in [m]$, and for all $\mathsf{msg} \in \mathcal{M}$, it holds that $f(x_i, \mathsf{msg}) = f(y_i, \mathsf{msg})$, and $f(\mathsf{msg}, x_i) = f(\mathsf{msg}, y_i)$. For simplicity we often use the following notation to denote this requirement:

$$f(x_i, \cdot) = f(y_i, \cdot), \quad f(\cdot, x_i) = f(\cdot, y_i)$$

Lemma 2.1 *The above adaptive, multi-message indistuiguishability security is equivalent to adaptive, single-message indistinguishability security.*

Proof: (sketch.) We show that if a public-key binary FE scheme is single-message secure, then it is multi-message secure. This can be achieved through a sequence of hybrid games. For $i \in \{1, 2, \ldots, m\}$, define Hybrid i where the simulator encrypts $(y_1, y_2, \ldots, y_i, x_{i+1}, \ldots, x_m)$. What we would like to show is that Hybrid 0 (encrypting all x_i values) is indistinguishable from Hybrid m (encrypting all y_i values). To show this, we just need to show that each adjacent pairs of hybrids are indistinguishable — and this can be easily reduced to the single-message security. ■

We stress that while in the public-key setting, multi-message and single-message security are equivalent, this is not true in the symmetric-key setting mentioned later.

Selective security. Our $i\mathcal{O}$-based construction below only achieves selective security. Next we define selective secure indistinguishability-based security for binary FE. but we note that we can achieve the stronger definitions through standard complexity-leveraging techniques.

Note that similar to the adaptive setting, in the selective model, single-message and multi-message security are equivalent. We therefore only define single-message security for simplicity.

We say that a binary FE scheme is selectively, single-message, IND-secure if for all PPT, *non-trivial, stateful* adversaries $\mathcal{A}$, for a uniformly chosen $b \leftarrow \{0,1\}$, $\Pr[b' = b] \leq \frac{1}{2} + \mathsf{negl}(\kappa)$ in the following experiment:

selective single-message IND-security:

1. $(x, y) \leftarrow \mathcal{A}(1^\kappa)$
2. $(\mathsf{msk}, \mathsf{param}) \leftarrow \mathsf{Setup}(1^\kappa)$
3. $b \leftarrow \{0,1\}$
4. If $b = 0$: $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{param}, x)$, else: $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{param}, y)$
5. $b' \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\cdot)}(\mathsf{param}, \mathsf{ct})$

Here $\mathcal{A}$ is *non-trivial* if $f(x, \cdot) = f(y, \cdot)$ and $f(\cdot, x) = f(\cdot, y)$ for every query f made to KeyGen oracle.

Internal (hardcoded) state: $\mathsf{param} = (\mathsf{crs}, \mathsf{pk}, \mathsf{pk}')$, sk, f

On input: $\mathsf{ct}_0, \mathsf{ct}_1$

- Unpack $(c_0, c_0', \pi_0) \leftarrow \mathsf{ct}_0$ and $(c_1, c_1', \pi_1) \leftarrow \mathsf{ct}_1$. Let $\mathsf{stmt}_0 := (c_0, c_0')$, and $\mathsf{stmt}_1 := (c_1, c_1')$ be statements for the NP-language $L_{\mathsf{pk},\mathsf{pk}'}$.
- If $\mathsf{NIZK.Verify}(\mathsf{crs}, \pi_0, \mathsf{stmt}_0) = \mathsf{NIZK.Verify}(\mathsf{crs}, \pi_1, \mathsf{stmt}_1) = 1$ then, compute $x = \mathcal{E}.\mathsf{Dec}(\mathsf{sk}, c_0)$ and $y = \mathcal{E}.\mathsf{Dec}(\mathsf{sk}, c_1)$, and output $f(x, y)$.
- Else, output $\perp$.

Figure 1: Public-key IND-secure binary FE: Program P

2.1.2 Construction based on Indistinguishability Obfuscation

Scheme description. Our construction uses a SSS-NIZK scheme $\mathsf{NIZK} := (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ and a best possible obfuscation scheme $i\mathcal{O}$, both of which are defined in Appendix A. We also use a semantically secure public-key encryption scheme $\mathcal{E} := (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. We assume the encryption scheme has perfect correctness. The detailed construction is described below.

$\underline{\mathsf{Setup}(1^\kappa)}$:

1. $\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\kappa)$
2. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$, $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$
3. Output $\mathsf{param} := (\mathsf{crs}, \mathsf{pk}, \mathsf{pk}')$, $\mathsf{msk} := (\mathsf{param}, \mathsf{sk})$

$\underline{\mathsf{KeyGen}(\mathsf{msk}, f)}$:

1. Parse msk into $(\mathsf{param}, \mathsf{sk})$ where $\mathsf{param} = (\mathsf{crs}, \mathsf{pk}, \mathsf{pk}')$. Construct a circuit C_f for the program P described in Figure 2.
2. Define $\mathsf{TK}_f := i\mathcal{O}(C_f)$, and output TK_f.

$\underline{\mathsf{Enc}(\mathsf{param}, x)}$:

1. Unpack $(\mathsf{crs}, \mathsf{pk}, \mathsf{pk}') \leftarrow \mathsf{param}$.
2. Compute $c = \mathcal{E}.\mathsf{Enc}(\mathsf{pk}, x; \rho)$ and $c' = \mathcal{E}.\mathsf{Enc}(\mathsf{pk}', x; \rho')$ for randomly chosen ρ and ρ' needed for the randomized encryption algorithm.
3. Create a NIZK $\pi = \mathsf{NIZK.Prove}(\mathsf{crs}, (c, c'), (\rho, \rho', x))$ for the following language $L_{\mathsf{pk},\mathsf{pk}'}$: for any statement $\mathsf{stmt} := (c, c')$, $\mathsf{stmt} \in L_{\mathsf{pk},\mathsf{pk}'}$ if and only if
$$\exists(\rho, \rho', x) \text{ s.t. } (c = \mathcal{E}.\mathsf{Enc}(\mathsf{pk}, x; \rho)) \wedge \left(c' = \mathcal{E}.\mathsf{Enc}(\mathsf{pk}', x; \rho')\right).$$
4. Output $\mathsf{ct} := (c, c', \pi)$

$\underline{\mathsf{Eval}(\mathsf{param}, \mathsf{TK}_f, \mathsf{ct}_0, \mathsf{ct}_1)}$:

1. Interpret TK_f as an obfuscated program. Compute $\mathsf{TK}_f(\mathsf{ct}_0, \mathsf{ct}_1)$ and output the result.

In Appendix B.1 we prove the following theorem.

Theorem 2.2 (Public-key IND-secure binary-FE from $i\mathcal{O}$.) *If $i\mathcal{O}$ is an indistinguishability obfuscator,* NIZK *is statistically simulation sound NIZK, and the encryption scheme is semantically secure and perfectly correct, then the above construction is selectively, IND-secure, as defined in Section 2.1.1. Further, using standard complexity leveraging techniques, we can achieve adaptive IND-security.*

Instantiation and efficiency. Suppose we instantiate our scheme with the $i\mathcal{O}$ construction by Garg et al. [17], and the SSS-NIZK constructions described in Appendix A.2. With this instantiation, the public parameter size, encryption time, decryption time, ciphertext length are all $\mathrm{poly}(\kappa)$ (i.e., depend only on the security parameter). The size of the token and time to generate the token is $O(|f|) \cdot \mathrm{poly}(\kappa)$.

2.2 The Symmetric-Key Setting

2.2.1 Definitions

Let $\mathcal{F} = \{\mathcal{F}_\kappa\}_{\kappa>0}$ be a collection of function families, where every $f \in \mathcal{F}_\kappa$ is a polynomial time function $f : \mathcal{M}_\kappa \times \mathcal{M}_\kappa \to \mathcal{M}'_\kappa$. A symmetric-key binary FE scheme supporting $\mathcal{F}$ is a collection of algorithms: $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval})$. The first three algorithms are probabilistic, and Eval is deterministic. They have the following semantics, if we leave the randomness implicit:

Setup: $(\mathsf{msk}, \mathsf{param}) \leftarrow \mathsf{Setup}(1^\kappa)$

KeyGen: for any $f \in \mathcal{F}_\kappa$, $\mathsf{TK}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$

Enc: $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{msk}, \mathsf{msg})$

Eval: $\mathsf{ans} \leftarrow \mathsf{Eval}(\mathsf{param}, \mathsf{TK}_f, \mathsf{ct}_1, \mathsf{ct}_2)$

In comparison with the public-key setting, here the encryptor must have a secret key msk to perform encryption.

Correctness. As usual, we must define the desired correctness and security properties. The correctness property states that, given $(\mathsf{msk}, \mathsf{param}) \leftarrow \mathsf{Setup}(1^\kappa)$, with overwhelming probability over the randomness used in Setup, KeyGen and Enc, it holds that $\mathsf{Eval}(\mathsf{KeyGen}(\mathsf{msk}, f), \mathsf{param}, \mathsf{Enc}(\mathsf{msk}, x), \mathsf{Enc}(\mathsf{msk}, y)) = f(x, y)$.

Adaptive security. We now define security for IND-secure symmetric-key binary FE. Similarly as before, we first define adaptive security, then we define the selective security relaxation. Unlike in the public key setting, here single-message security does not imply multi-message security, so we cannot prove a parallel to Lemma 2.1. A similar observation was made by Pandey and Rouselakis [28]. Therefore we only define the stronger multi-message security notion.

An Ind-Secure scheme is said to be *adaptively* Ind-Secure if for all PPT, *non-trivial* adversary $\mathcal{A}$, its probability of winning the following game is $\Pr[b = b'] < \frac{1}{2} + \mathsf{negl}(\kappa)$.

Ind-Secure-adaptive:

1. $(\mathsf{msk}, \mathsf{param}) \leftarrow \mathsf{Setup}(1^\kappa)$
2. $b \xleftarrow{\$} \{0, 1\}$
3. $(x_1, y_1) \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\cdot)}(\mathsf{param}, 1^\kappa)$.
4. For $i = 1$ to m:
   - if $b = 0$: $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{msk}, x_i)$, else: $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{msk}, y_i)$.
   - $(x_{i+1}, y_{i+1}) \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\cdot)}(\mathsf{ct}_i)$.
5. $b' \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\cdot)}$

An adversary is considered *non-trivial* if for every query f made to the $\mathsf{KeyGen}(\cdot)$ oracle, and for any polynomial m, for all $i, j \in [m]$, it holds that $f(x_i, x_j) = f(y_i, y_j)$.

Internal (hardcoded) state: $\mathsf{param} = (\mathsf{crs}, \mathsf{pk}, \mathsf{pk}', \mathsf{com})$, sk, f

On input: $\mathsf{ct}_0, \mathsf{ct}_1$

- Parse ct_0 as $(c_0, c_0', \alpha_0, \pi_0)$ and ct_1 as $(c_1, c_1', \alpha_1, \pi_1)$. Let $\mathsf{stmt}_0 := (c_0, c_0', \alpha_0)$, and $\mathsf{stmt}_1 := (c_1, c_1', \alpha_1)$. Verify that $\alpha_0 = \alpha_1$ and $\mathsf{NIZK.Verify}(\mathsf{crs}, \mathsf{stmt}_0, \pi_0) = \mathsf{NIZK.Verify}(\mathsf{crs}, \mathsf{stmt}_1, \pi_1) = 1$. If fails, output $\perp$.
- Compute $x_0 = \mathcal{E}.\mathsf{Dec}(\mathsf{sk}, c_0)$ and $x_1 = \mathcal{E}.\mathsf{Dec}(\mathsf{sk}, c_1)$ output $f(x_0, x_1)$.

Figure 2: Symmetric-key IND-secure binary FE: Program P

Selective security. Our main construction (based on $i\mathcal{O}$) below only achieves selective security, but we note that we can achieve adaptive security through standard complexity-leveraging techniques.

An Ind-Secure scheme is said to be *selectively* IND-secure if for all PPT, *non-trivial* adversary $\mathcal{A}$, its probability of winning the following game is $\Pr[b = b'] < \frac{1}{2} + \mathsf{negl}(\kappa)$.

Ind-Secure-selective:

1. $\{(x_1, \ldots, x_m), (y_1, \ldots, y_m)\} \leftarrow \mathcal{A}(1^\kappa)$
2. $(\mathsf{msk}, \mathsf{param}) \leftarrow \mathsf{Setup}(1^\kappa)$
3. $b \leftarrow \{0, 1\}$
4. if $b = 0$: $\forall i \in [m] : \mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{msk}, x_i)$, else: $\forall i \in [m] : \mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{msk}, y_i)$.
5. $b' \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\cdot)}(\mathsf{param}, \mathsf{ct}_1, \ldots, \mathsf{ct}_m)$

An adversary is considered *non-trivial* if for every query f made to the $\mathsf{KeyGen}(\cdot)$ oracle, and for all $i, j \in [m]$ where $m = \mathrm{poly}(\kappa)$, it holds that $f(x_i, x_j) = f(y_i, y_j)$. We note that this is a much weaker restriction on the adversary than the one used in the public key setting, which makes symmetric key schemes more difficult to construct.

2.2.2 Construction based on Indistinguishability Obfuscation

Scheme description. Our construction uses a SSS-NIZK scheme $\mathsf{NIZK} := (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ that is statistically simulation sound for multiple simulated statements, an indistinguishable obfuscation scheme $i\mathcal{O}$, and a perfectly binding commitment scheme $(\mathsf{commit}, \mathsf{open})$, all of which are defined in Appendix A. We also use a CPA-secure public-key encryption scheme $\mathcal{E} := (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with perfect correctness. Our construction is as follows:

$\mathsf{Setup}(1^\kappa)$:

1. $\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\kappa, m)$ where m is an upper-bound on the number of ciphertexts.
2. $\alpha, r \leftarrow \{0, 1\}^\kappa$; $\mathsf{com} = \mathsf{commit}(\alpha; r)$
3. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$, $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$
4. Output $\mathsf{param} := (\mathsf{crs}, \mathsf{pk}, \mathsf{pk}', \mathsf{com})$, $\mathsf{msk} := (\mathsf{param}, \mathsf{sk}, \mathsf{sk}', \alpha, r)$

$\mathsf{KeyGen}(\mathsf{msk}, f)$

1. Using $\mathsf{msk} = (\mathsf{sk}, \mathsf{sk}', \alpha, r)$, construct a circuit C_f that computes program P as described in Figure 2.
2. Define $\mathsf{TK}_f := i\mathcal{O}(C_f)$, and output TK_f.

$\mathsf{Enc}(\mathsf{msk}, x)$:

1. Parse msk as $(\mathsf{param}, \mathsf{sk}, \mathsf{sk}', \alpha, r)$, where $\mathsf{param} := (\mathsf{crs}, \mathsf{pk}, \mathsf{pk}', \mathsf{com})$.
2. Compute $c = \mathcal{E}.\mathsf{Enc}(\mathsf{pk}, x; \rho)$ and $c' = \mathcal{E}.\mathsf{Enc}(\mathsf{pk}', x; \rho')$ for random strings ρ and ρ' consumed by the encryption algorithm.
3. Output $\mathsf{ct} := (c, c', \alpha, \pi)$ where $\pi := \mathsf{NIZK.Prove}(\mathsf{crs}, (c, c', \alpha), (r, \rho, \rho', x))$ is a NIZK for the language $L_{\mathsf{pk},\mathsf{pk}',\mathsf{com}}$: for any statement $\mathsf{stmt} := (c, c', \alpha)$, $\mathsf{stmt} \in L_{\mathsf{pk},\mathsf{pk}',\mathsf{com}}$ if and only if

$$\exists (r, \rho, \rho', x) \text{ s.t. } \left(c = \mathcal{E}.\mathsf{Enc}(\mathsf{pk}, x; \rho)\right) \wedge \left(c' = \mathcal{E}.\mathsf{Enc}(\mathsf{pk}', x; \rho')\right) \wedge \left(\mathsf{com} = \mathsf{commit}(\alpha; r)\right). \quad (1)$$

$\underline{\mathsf{Eval}(\mathsf{param}, \mathsf{TK}_f, \mathsf{ct}_0, \mathsf{ct}_1)}$:

1. Interpret TK_f as an obfuscated circuit. Compute $\mathsf{TK}_f(\mathsf{ct}_0, \mathsf{ct}_1)$ and output the result.

In Appendix B.2 we provide a proof of the following theorem.

Theorem 2.3 (Symmetric-key IND-secure binary-FE from $i\mathcal{O}$.) *If $i\mathcal{O}$ is an indistinguishability obfuscator,* NIZK *is a statistically simulation sound NIZK scheme, the commitment is perfectly binding and computationally hiding, and the encryption scheme is semantically secure and perfectly correct, then the above construction is selectively IND-secure, as defined in Section 2.2.1. Using standard complexity leveraging techniques, we can further achieve adaptive IND-security.*

Instantiation and efficiency. We can instantiate our scheme with the SSS-NIZK scheme outlined in Appendix A.2 and with the $i\mathcal{O}$ construction by Garg et al. [17]. The ciphertext is succinct, and is $\mathrm{poly}(\kappa)$ in size. For a scheme tolerant up to m ciphertext queries, the public parameter size, encryption time is $O(m)\mathrm{poly}(\kappa)$, decryption time is $O(m + |f|)\mathrm{poly}(\kappa)$. The reason for the dependence on m is due to the simulator's need to simultaneously simulate $O(m)$ SSS-NIZKs in the simulation, which increases the size of the crs to $O(m)\mathrm{poly}(\kappa)$. Removing the dependence on m for encryption and decyrption remains an important open problem (unless we assume non-falsifiable assumptions such as di$\mathcal{O}$, as we will see next).

2.2.3 Construction based on Differing-inputs Obfuscation

We can make the encryption and decryption time more succinct if we assume a stronger assumption, namely, differing input obfuscation. To do this, we can make the following modifications to our construction: we replace the indistinguishability obfuscation $i\mathcal{O}$ with a differing-inputs obfuscation di$\mathcal{O}$, and replace statistically simulation-sound NIZK (SSS-NIZK) scheme with an simulation sound NIZK.

Efficiency. Using di$\mathcal{O}$, the public parameter size and ciphertext size are $\mathrm{poly}(\kappa)$; encryption time is $\mathrm{poly}(\kappa)$; and decryption time is $O(|f|)\mathrm{poly}(\kappa)$. This assumes that a di$\mathcal{O}$ scheme exists whose obfuscated program is of size $O(|f|)\mathrm{poly}(\kappa)$ for an original program f of size $O(|f|+\kappa)$. The improvement stems from the fact that the crs no longer has to grow to accommodate every simulated (false) NIZK proof required in our hybrid games. This is because false proofs are hard for an adversary to find without the trapdoor, which is sufficient for leveraging the security properties of di$\mathcal{O}$ obfuscation. We do not prove the next theorem, as Theorem 2.11 is very similar and is proven in Section B.3.

Theorem 2.4 (Symmetric-key IND-secure binary FE from di$\mathcal{O}$.) *If di$\mathcal{O}$ is a differing-input obfuscater,* NIZK *is an adaptive simulation sound NIZK scheme, the commitment is perfectly binding and computationally hiding, and the encryption scheme is semantically secure and perfectly correct, then the above construction is selectively, IND-secure, as defined in Section 2.2.1.Further, we can obtain adaptive IND-security through standard complexity leveraging techniques.*

Remark 2.5 We note that even with the stronger di$\mathcal{O}$ assumption, we can still only achieve selective security (unless we use complexity leveraging techniques). Intuitively, in order to reduce to the security of the di$\mathcal{O}$ construction, we have to argue that it is hard to find inputs on which the two circuits to be obfuscated differ. This property stems from the fact that the NIZK trapdoor is kept from the reduction adversary. On the other hand, without this trapdoor, it is hard for the reduction adversary to create the false proof that is needed to simulate the challenge ciphertexts. In the selective-security model, the reduction adversary can simple get the challenge ciphertext from the auxiliary output of the sampling algorithm in the di$\mathcal{O}$ construction.

In Appendix D we define a new, interactive notion of di$\mathcal{O}$ that addresses this problem (Definition D.2). In the multi-client setting, we demonstrate that this assumption suffices for achieving adaptive security (Appendix D). The same is true with the construction just presented. We omit the proof as it is essentially the same as the one in Appendix D.

2.3 The Symmetric-Key, Multi-Client Setting

2.3.1 Definitions

Let $\mathcal{F} = \{\mathcal{F}_\kappa\}_{\kappa>0}$ be a collection of function families, where every $f \in \mathcal{F}_\kappa$ is a polynomial time function $f : \mathcal{M}_\kappa \times \cdots \times \mathcal{M}_\kappa \to \mathcal{M}'_\kappa$. A multi-client functional encryption scheme (MC-FE) supporting n users and function family $\mathcal{F}_\kappa$ is a collection of the following algorithms:

$\mathsf{Setup} : (\mathsf{msk}, \{\mathsf{usk}_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\kappa, n)$. Here usk_i is a user secret key for user $i \in [n]$.

$\mathsf{Enc} : \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{usk}_i, x, t)$. Here t represents the current time step $t \in \mathbb{N}$.

$\mathsf{KeyGen} : \mathsf{TK}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$.

$\mathsf{Dec} : \mathsf{ans} \leftarrow \mathsf{Dec}(\mathsf{TK}_f, \{\mathsf{ct}_1, \mathsf{ct}_2, \ldots, \mathsf{ct}_n\})$.

Correctness. We say that an MC-FE scheme is correct, if given $(\mathsf{msk}, \{\mathsf{usk}_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\kappa, n)$, given some $t \in \mathbb{N}$, except with negligible probability over randomness used in Setup, Enc, KeyGen, and Dec, it holds that $\mathsf{Dec}(\mathsf{KeyGen}(\mathsf{msk}, f), \mathsf{Enc}(\mathsf{usk}_1, x_1, t), \ldots, \mathsf{Enc}(\mathsf{usk}_n, x_n, t)) = f(x_1, x_2, \ldots, x_n)$.

2.3.2 Security Definitions

Our definitions assume a *static corruption* model where the corrupted parties are specified at the beginning of the security game. How to support adaptive corruption is an interesting direction for future work.

Notations. We often use a shorthand $\vec{x}$ to denote a vector $\vec{x} := (x_1, x_2, \ldots, x_n)$. Let disjoint sets $G, \overline{G}$ denote the set of uncorrupted and corrupted parties respectively. $G \cup \overline{G} = [n]$. We use the short-hand $\overrightarrow{\mathsf{var}}_G$ to denote the vector $\{\mathsf{var}_i\}_{i\in G}$ for a variable var. Similarly, we use the short-hand $\vec{\mathsf{ct}}_G \leftarrow \mathsf{Enc}(\vec{\mathsf{usk}}_G, \vec{x}_G, t)$ to denote the following: $\forall i \in G : \mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{usk}_i, x_i, t)$. We use the shorthand $f(\vec{x}_G, \cdot) : \mathcal{M}^{|\overline{G}|} \to \mathcal{M}'$ to denote a function restricted to a subset G on inputs denoted $\vec{x}_G \in \mathcal{M}^{|G|}$.

Define short-hand $K(\cdot) := \mathsf{KeyGen}(\mathsf{msk}, \cdot)$ to be an oracle to the KeyGen function. Define $E_G(\cdot)$ to be a *stateful* encryption oracle for the uncorrupted set G. Its initial state is the intial time step counter $t := 0$. Upon each invocation $E_G(\vec{x}_G)$, the oracle increments the current time step $t \leftarrow t + 1$, and returns $\mathsf{Enc}(\vec{\mathsf{usk}}_G, \vec{x}_G, t)$.

Adaptive security. Now, define the following adaptive, IND-security experiment for a stateful adversary $\mathcal{A}$. For simplicity, we will omit writing the adversary $\mathcal{A}$'s state explicitly.

1. $G, \overline{G} \leftarrow \mathcal{A}$

2. $b \xleftarrow{\$} \{0,1\}$
3. $(\mathsf{msk}, \{\mathsf{usk}_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\kappa, n)$
4. $(\vec{x}_G^0, \vec{y}_G^0) \leftarrow \mathcal{A}^{K(\cdot)}(\vec{\mathsf{usk}}_{\overline{G}})$

   For $j = 0, 1, \ldots, \mathrm{poly}(\kappa)$:

   If $b = 0$: $\vec{\mathsf{ct}}_G^j \leftarrow E_G(\vec{x}_G^j)$. Else: $\vec{\mathsf{ct}}_G^j \leftarrow E_G(\vec{y}_G^j)$

   $(\vec{x}_G^{j+1}, \vec{y}_G^{j+1}) \leftarrow \mathcal{A}^{K(\cdot)}(\vec{\mathsf{ct}}_G^j)$
5. $b' \leftarrow \mathcal{A}^{K(\cdot)}$.

We say that an adversary $\mathcal{A}$ is *non-trivial* if for all j in the above game, the $\vec{x}_G^j$ and $\vec{y}_G^j$ submitted by $\mathcal{A}$ satisfies the following property: for any function f queried to the $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ oracle,

$$f(\vec{x}_G^j, \cdot) = f(\vec{y}_G^j, \cdot)$$

Definition 2.6 (Adaptive, IND-security of MC-FE) *We say that an MC-FE scheme is indistinguishably secure, if for any polynomial-time, non-trivial adversary $\mathcal{A}$ in the above game,*

$$\left|\Pr[b' = b] - \frac{1}{2}\right| \leq \mathsf{negl}(\kappa).$$

Single-challenge security. We define a single-challenge version of the adaptive security game — it turns out that single-challenge, adaptive security is equivalent to full adaptive security by Lemma 2.8. Single-challenge security can typically be easier to work with in proofs. Intuitively, single-challenge security is when there is a single challenge time step, in the following game.

1. $G, \overline{G} \leftarrow \mathcal{A}$
2. $b \xleftarrow{\$} \{0,1\}$
3. $(\mathsf{msk}, \{\mathsf{usk}_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\kappa, n)$
4. $(\vec{x}_G^*, \vec{y}_G^*) \leftarrow \mathcal{A}^{K(\cdot), E_G(\cdot)}(\vec{\mathsf{usk}}_{\overline{G}})$
5. If $b = 0$: $\vec{\mathsf{ct}}_G^* \leftarrow E_G(\vec{x}_G^*)$. Else: $\vec{\mathsf{ct}}_G^* \leftarrow E_G(\vec{y}_G^*)$.
6. $b' \leftarrow \mathcal{A}^{K(\cdot), E_G(\cdot)}(\vec{\mathsf{ct}}_G^*)$.

We say that an adversary $\mathcal{A}$ is *non-trivial*, if the following condition holds: For any function f queried to the $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ oracle,

$$f(\vec{x}_G^*, \cdot) = f(\vec{y}_G^*, \cdot)$$

Definition 2.7 (Single-challenge, adaptive, IND-security of MC-FE) *We say that an MC-FE scheme is single-challenge, indistinguishably secure, if for any polynomial-time, non-trivial adversary $\mathcal{A}$ in the above game, $\left|\Pr[b' = b] - \frac{1}{2}\right| \leq \mathsf{negl}(\kappa)$.*

We show in the following lemma that single-challenge security and multi-challenge security are equivalent.

Lemma 2.8 *For MC-FE, single-challenge, adaptive IND-security (Definition 2.6) is equivalent to multi-challenge, adaptive IND-security (Definition 2.7).*

Proof: It is trivial to see that Definition 2.6 implies Definition 2.7.

The other direction, that single-challenge, adaptive, IND-based security implies adaptive, IND-based security, may be proven through a simple hybrid argument. Suppose we have an adversary $\mathcal{A}$ that can break the full IND-security of Definition 2.6, we show how to construct an adversary $\mathcal{B}$ that can break the single-challenge security of Definition 2.7. Define a sequence of hybrid games. Let Hybrid X be the game where for each time step, the challenger always encrypts $\vec{x}_G$. Let Hybrid Y be the game where for each time step, the challenger always encrypts $\vec{y}_G$. Suppose $\mathcal{A}$ queries ciphertexts for T time steps. Define Hybrid i, where $0 \leq i \leq T$ as below: In Hybrid i, for the first i time steps, the challenger encrypts $\vec{x}_G$, and for the remaining time steps, it encrypts $\vec{y}_G$. Obviously, Hybrid 0 is the same as Hybrid X, and Hybrid T is the same as Hybrid Y. Now, since $\mathcal{A}$ has non-negligible advantage of distinguishing Hybrid 0 and Hybrid T, due to the hybrid argument, there must exist $0 \leq i < T$, such that $\mathcal{A}$ can distinguish Hybrid i and $i+1$ — note that this means that $\mathcal{A}$ can break single-challenge security. ■

Selective security. Our construction below is based on $i\mathcal{O}$ which achieves a relaxed notion of security, i.e., selective security. But we note that we can lift the scheme to have adaptive security through standard complexity-leveraging techniques.

Define the following single-challenge, selective experiment for a stateful adversary $\mathcal{A}$. For simplicity, we will omit writing the adversary $\mathcal{A}$'s state explicitly. Define short-hand $K(\cdot) := \mathsf{KeyGen}(\mathsf{msk}, \cdot)$ to be an oracle to the KeyGen function. Define $E_G(\cdot)$ to be a *stateful* encryption oracle for the uncorrupted set G. Its initial state is the intial time step counter $t := 0$. Upon each invocation $E_G(\vec{x}_G)$, the oracle increments the current time step $t \leftarrow t + 1$, and returns $\mathsf{Enc}(\vec{\mathsf{usk}}_G, \vec{x}_G, t)$.

1. $G, \overline{G},\ (\vec{x}^*_G, \vec{y}^*_G) \leftarrow \mathcal{A}$.
2. $b \xleftarrow{\$} \{0,1\}, \quad (\mathsf{msk}, \{\mathsf{usk}_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\kappa, n)$
3. "challenge" $\leftarrow \mathcal{A}^{K(\cdot), E_G(\cdot)}(\vec{\mathsf{usk}}_{\overline{G}})$.
4. If $b = 0$: $\vec{\mathsf{ct}}^*_G \leftarrow E_G(\vec{x}^*_G)$. Else: $\vec{\mathsf{ct}}^*_G \leftarrow E_G(\vec{y}^*_G)$.
5. $b' \leftarrow \mathcal{A}^{K(\cdot), E_G(\cdot)}(\vec{\mathsf{ct}}^*_G)$.

We say $\mathcal{A}$ is *non-trivial*, if for any function f queried to the $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ oracle, $f(\vec{x}^*_G, \cdot) = f(\vec{y}^*_G, \cdot)$.

Definition 2.9 (Selective IND-security of MC-FE) *We say that an MC-FE scheme is selectively indistinguishably secure, if for any* PPT*, non-trivial adversary $\mathcal{A}$ in the above selective security game,* $\left|\Pr[b' = b] - \frac{1}{2}\right| \leq \mathsf{negl}(\kappa)$.

2.3.3 Construction based on Indistinguishability Obfuscation

As mentioned in the introduction, the challenge here is that a subset of the parties may be corrupted. Our idea is to give to each party a different α_i which they must rely on during encryption. In the simulation, the α_i's in the challenge-time step will be simulated, and we embed an orthogonal test in the $i\mathcal{O}$ to prevent the mix-and-match of simulated ciphertexts and honest ones for the honest set of parties G. We rely on a DDH assumption to separate out the challenge time step from the non-challenge time steps, such that the simulator only needs to use fake α values for the challenge time steps, and the NIZK proofs for the non-challenge time steps need not be simulated.

We first describe a version of our scheme with a random oracle. However, later, we point out that *the random oracle can easily be removed* at the cost of expanding the public parameters by an additive $O(T)\mathrm{poly}(\kappa)$ factor.

Let $\mathcal{G}$ denote a group of prime order $p > 2^n \cdot 2^\kappa$ in which Decisional Diffie-Hellman is hard. Let $H : \mathbb{N} \to \mathcal{G}$ denote a hash function modelled as a random oracle. Let $\mathcal{E} := (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ denote a public-key encryption scheme.

- $\mathsf{Setup}(1^\kappa, n)$: Compute $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$, and $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$. Run $\mathsf{crs} := \mathsf{NIZK.Setup}(1^\kappa, n)$, where n is the number of clients. Choose a random generator $g \xleftarrow{\$} \mathcal{G}$. Choose random $\alpha_1, \alpha_2, \ldots, \alpha_n \in \mathbb{Z}_p$. For $i \in [n]$, let $g_i := g^{\alpha_i}$. Set $\mathsf{param} := (\mathsf{crs},\ \mathsf{pk}, \mathsf{pk}',\ g, \{g_i\}_{i\in[n]})$. The secret keys for each user are: $\mathsf{usk}_i := (\alpha_i, \mathsf{param})$ The master secret key is: $\mathsf{msk} := (\{\alpha_i\}_{i\in[n]}, \mathsf{sk}, \mathsf{sk}')$.
- $\mathsf{Enc}(\mathsf{usk}_i, x, t)$: For user i to encrypt a message x for time step t, it computes the following. Let $h_t := H(t)$. Choose random ρ and ρ' as the random bits needed for the public-key encryption scheme. Let $c := \mathcal{E}.\mathsf{Enc}(\mathsf{pk}, x; \rho)$ and $c' := \mathcal{E}.\mathsf{Enc}(\mathsf{pk}', x; \rho')$. Let $d = h_t^{\alpha_i}$, Let statement $\mathsf{stmt} := (t, i, c, c', d)$; let witness $w := (\rho, \rho', x, \alpha_i)$. Let the NP language be defined as in Figure 3. Let $\pi := \mathsf{NIZK.Prove}(\mathsf{crs}, \mathsf{stmt}, w)$. Informally, this proves that 1) the two ciphertexts c and c' encrypt consistent plaintexts using pk and pk' respectively; and 2) (h_t, g_i, d) is a true Diffie-Hellman tuple.

  The ciphertext is defined as:$\mathsf{ct} := (t, i, c, c', d, \pi)$.

Our NP language $L_{\mathsf{pk},\mathsf{pk}',g,\{g_i\}_{i\in[n]}}$ is parameterized by $(\mathsf{pk}, \mathsf{pk}', g, \{g_i\}_{i\in[n]})$ output by the Setup algorithm as part of the public parameters. A statement of this language is is of the format $\mathsf{stmt} := (t, i, c, c', d)$, and a witness is of the format $w := (\rho, \rho', x, \omega)$. A statement $\mathsf{stmt} := (t, i, c, c', d) \in L_{\mathsf{pk},\mathsf{pk}',g,\{g_i\}_{i\in[n]}}$, iff

$$\exists\, x, (\rho, \rho'), \omega \quad \text{s.t. } \mathsf{DH}(h_t, g_i, d, \omega) \wedge (c = \mathcal{E}.\mathsf{Enc}(\mathsf{pk}, x; \rho)) \wedge \left(c' = \mathcal{E}.\mathsf{Enc}(\mathsf{pk}', x; \rho')\right)$$

where $h_t = H(t)$ for the t defined by ct; $g_i := g^{\alpha_i}$ is included in the public parameters; (ρ, ρ') are the random strings used for the encryptions; and $\mathsf{DH}(A, B, C, \omega)$ is defined as the following relation that checks that (A, B, C) is a Diffie-Hellman tuple with the witness ω:

$$\mathsf{DH}(A, B, C, \omega) := ((A = g^\omega) \wedge (C = B^\omega)) \ \vee \ ((B = g^\omega) \wedge (C = A^\omega))$$

Figure 3: NP language $L_{\mathsf{pk},\mathsf{pk}',g,\{g_i\}_{i\in[n]}}$.

  Note that the NIZK π ties together the ciphertexts (c, c') with the term $d = H(t)^{\alpha_i}$. This intuitively ties (c, c') with the time step t, such that it cannot be mix-and-matched with other time steps.

- $\mathsf{KeyGen}(\mathsf{msk}, f)$: To generate a server token for a function f over n parties' inputs compute token $\mathsf{TK}_f := i\mathcal{O}(P)$ for a Program P defined as in Figure 4:
- $\mathsf{Dec}(\mathsf{TK}_f, \mathsf{ct}_1, \ldots, \mathsf{ct}_n)$: Interpret TK_f as an obfuscated program. Output $\mathsf{TK}_f(\mathsf{ct}_1, \mathsf{ct}_2, \ldots, \mathsf{ct}_n)$.

In Appendix B.3 we provide a proof of the following theorem.

Theorem 2.10 *Let $\mathcal{G}$ be a group for which the Diffie-Hellman assumption holds, and let H be a random oracle. If the $i\mathcal{O}$ is secure, the NIZK is statistically simulation sound, and the encryption scheme is semantically secure and perfectly correct, then the above construction is selectively, IND-secure, as defined in Section 2.3.1.*

Removing the random oracle. It is trivial to remove the random oracle if we choose $h_1, h_2, \ldots, h_T$ at random in the setup algorithm, and give them to each user as part of their secret keys (i.e., equivalent to embedding them in the public parameters). This makes the user key $O(n + T)\mathrm{poly}(\kappa)$ in size, where n denotes the number of parties, and T denotes an upper bound on the number of time steps.

Program $P(\mathsf{ct}_1, \mathsf{ct}_2, \ldots, \mathsf{ct}_n)$**:**

Internal hard-coded state: $\mathsf{param} = (\mathsf{crs}, \mathsf{pk}, \mathsf{pk}', g, \{g_i\}_{i\in[n]}), \mathsf{sk}, f$

1. For $i \in [n]$, unpack $(t_i, j_i, c_i, c_i', d_i, \pi_i) \leftarrow \mathsf{ct}_i$. Check that $t_1 = t_2 = \ldots = t_n$, and that $j_i = i$. Let $\mathsf{stmt}_i := (t_i, j_i, c_i, c_i', d_i)$.
2. For $i \in [n]$, check that $\mathsf{NIZK.Verify}(\mathsf{crs}, \pi_i, \mathsf{stmt}_i) = 1$.
3. If any of these above checks fail, output $\perp$.

   Else: for $i \in [n]$, let $x_i \leftarrow \mathcal{E}.\mathsf{Dec}(\mathsf{sk}, c_i)$. Output $f(x_1, x_2, \ldots, x_n)$.

Figure 4: MC-FE: Program P.

Instantiation and efficiency. We can instantiate our scheme using the SSS-NIZK construction described in Appendix A.2, and the $i\mathcal{O}$ construction described by Garg et. al [17]. In this way, our ciphertext is succinct, and is only $\mathrm{poly}(\kappa)$ in size. Letting n denote the number of parties, the encryption time is $O(n)\mathrm{poly}(\kappa)$, and the decryption time is $O(n + |f|) \cdot \mathrm{poly}(\kappa)$. The dependence on n arises due to the need for the simulator to simulate $O(n)$ SSS-NIZKs. Each user's secret key is of size is $O(n)\mathrm{poly}(\kappa)$ for the version with the random oracle, and is $O(n + T)\mathrm{poly}(\kappa)$ for the version of the scheme without the random oracle. Note that due to our use of the Diffie-Hellman assumption, we have removed the dependence on T for encryption/decryption time in a non-trivial manner.

2.3.4 Construction based on Differing-inputs Obfuscation

Using a stronger assumption, i.e., differing input obfuscation, we can further compress the encryption and decryption time, as well as public parameter size. The $\mathsf{di}\mathcal{O}$-based construction is the same as our construction described in Section 2.3.3, except that we now replace the $i\mathcal{O}$ with $\mathsf{di}\mathcal{O}$, and SSS-NIZK with adaptive simulation-sound NIZK,

Theorem 2.11 *If $\mathsf{di}\mathcal{O}$ is differing-inputs obfuscation, NIZK is adaptive simulation sound NIZK, and the encryption scheme is semantically secure and perfectly correct, and that the hash function H is a random oracle*[2]*, then the above construction is selectively IND-secure as defined in Section 2.3.1.*

In Appendix B.3 we provide a proof of the above theorem.

Efficiency. Using $\mathsf{di}\mathcal{O}$, the public parameter size, and the ciphertext size are $\mathrm{poly}(\kappa)$. Encryption time is $\mathrm{poly}(\kappa)$, and decryption time is $O(|f|)\mathrm{poly}(\kappa)$. This assumes that the underlying $\mathsf{di}\mathcal{O}$ produces an obfuscated program of size $O(|f|)\mathrm{poly}(\kappa)$ to obfuscate a function of size $O(|f| + \kappa)$.

3 Simulation-Based Security

In this section, we present the simulation-based (SIM-security) definition and investigate its feasibility and impossibility. Note that known impossibility results for unary FE can be easily extended to the binary FE setting. But in Section 3.1 we show a stronger impossibility result for public-key binary FE; we define a class of learnable functions, and then prove that SIM-secure public key binary-FE schemes exist only for learnable functions.

[2] As mentioned earlier, the random oracle can be removed by simply including the terms $\{h_t\}$ in param.

Then in Section 3.1.2, we show that the above result is tight, demonstrating a construction of SIM-secure binary-FE for any learnable function class from a construction of Virtual Black Box obfuscation for polynomial-time functions. We further in Section 3.1.3 we show a construction of SIM-secure public-key binary FE for any learnable function class from any public IND-secure binary-FE, which can be constructed from indistinguishable obfuscation as shown in Section 2.1.1. The resulting construction only supports post-challenge key queries (i.e. queries made after the challenge plaintext has been specified.). We note that this is somewhat stronger than selective security, as we still allow the challenge message to depend on the public parameters.

Finally, in Section 3.2, we show a symmetric-key analogue of the same construction; this construction supports some bounded number of pre-challenge key queries, a bounded number of challenge plaintexts, and an unbounded number of post-challenge queries. Both of these constructions follow along the lines of the compiler from Caro et al. [14], extended in a natural way to the multi-input setting.

3.1 The Public-Key Setting

In Section 2.1.1, we considered the IND-security for public-key binary-FE. In this section, we study the SIM-security in the public key setting. By extending the SIM-security for unary FE, we can define the SIM-security as follows. As mentioned in the Introduction, the impossibility result of SIM-secure unary FE can be extended to binary FE in a straightforward way. Here we intend to show stronger impossibility result in the binary FE setting, and we consider binary FE with specific plaintext form. That is, we include a tag $\mathtt{tag} \in \{\text{“1”},\text{“2”}\}$ in plaintext, which indicates the position of the input to the function.

3.1.1 Definition and a Stronger Impossibility Result

Let $\mathcal{F} = \{\mathcal{F}_\kappa\}_{\kappa>0}$ be a collection of function families, where every $f \in \mathcal{F}_\kappa$ is a polynomial time function $f : \mathcal{M}_\kappa \times \mathcal{M}_\kappa \to \mathcal{M}'_\kappa$. A public-key binary FE scheme supporting $\mathcal{F}$ is a collection of algorithms: $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval})$. The first three algorithms are probabilistic, and Eval is deterministic. They have the following semantics:

Setup: $(\mathsf{msk}, \mathsf{param}) \leftarrow \mathsf{Setup}(1^\kappa)$

KeyGen: for any $f \in \mathcal{F}_\kappa$, $\mathsf{TK}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$

Enc: $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{param}, \mathsf{msg}, \mathtt{tag})$ where $\mathtt{tag} \in \{\text{“1”},\text{“2”}\}$

Eval: $\mathsf{ans} \leftarrow \mathsf{Eval}(\mathsf{param}, \mathsf{TK}_f, \mathsf{ct}_1, \mathsf{ct}_2)$

Correctness. The correctness property states that with overwhelming probability over the randomness used in Setup, KeyGen, and Enc, for all $f \in \mathcal{F}$ and all messages $x, y \in \mathcal{M}$,

$$\mathsf{Eval}(\mathsf{param}, \mathsf{KeyGen}(\mathsf{msk}, f), \mathsf{Enc}(\mathsf{param}, x, \text{“1”}), \mathsf{Enc}(\mathsf{param}, y, \text{“2”})) = f(x, y).$$

Definition 3.1 (Simulation security of binary FE) *Let* $\mathsf{FE} = \{\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}\}$ *be a binary functional encryption scheme for a family of functions* $\mathcal{F}$*. Then for polynomials* $q_1(\cdot), \ell(\cdot), q_2(\cdot)$*, a PPT adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, and a stateful simulator* $\mathcal{S}$*, consider the following experiments:*

Real$_{\mathcal{A},q_1(\kappa),\ell(\kappa),q_2(\kappa)}(1^\kappa)$:

$(\mathsf{param}, \mathsf{msk}) \leftarrow \mathsf{FE.Setup}(1^\kappa)$

$((x_1, y_1), \ldots, (x_\ell, y_\ell), st) \leftarrow \mathcal{A}_1^{\mathsf{FE.KeyGen}(\mathsf{msk},\cdot)}(\mathsf{param})$

for $i \in [\ell]$, $\mathsf{ct}_{x_i} \leftarrow \mathsf{FE.Enc}(\mathsf{param}, x_i, \text{“1”}), \mathsf{ct}_{y_i} \leftarrow \mathsf{FE.Enc}(\mathsf{param}, y_i, \text{“2”})$

$\alpha \leftarrow \mathcal{A}_2^{\mathsf{FE.KeyGen}(\mathsf{msk},\cdot)}((\mathsf{ct}_{x_1}, \mathsf{ct}_{y_1}), \ldots, (\mathsf{ct}_{x_\ell}, \mathsf{ct}_{y_\ell}), st)$

Output $((x_1, y_1), \ldots, (x_\ell, y_\ell), \alpha)$

$\boldsymbol{Ideal}_{\mathcal{A},\mathcal{S},q_1(\kappa),\ell(\kappa),q_2(\kappa)}(1^\kappa)$:
$\mathsf{param} \leftarrow \mathcal{S}(1^\kappa)$
$((x_1,y_1),\ldots,(x_\ell,y_\ell),st) \leftarrow \mathcal{A}_1^{\mathcal{S}(\cdot)}(\mathsf{param})$
$(\mathsf{ct}_{x_1},\mathsf{ct}_{y_1}),\ldots,(\mathsf{ct}_{x_\ell},\mathsf{ct}_{y_\ell}) \leftarrow \mathcal{S}^{O'_{x_1,y_1,\ldots,x_\ell,y_\ell}(\cdot)}(1^{|x_1|},1^{|y_1|},\ldots,1^{|x_\ell|},1^{|y_\ell|})$
$\alpha \leftarrow \mathcal{A}_2^{O''(\cdot)}((\mathsf{ct}_{x_1},\mathsf{ct}_{y_1}),\ldots,(\mathsf{ct}_{x_\ell},\mathsf{ct}_{y_\ell}),st)$
Output $((x_1,y_1),\ldots,(x_\ell,y_\ell),\alpha)$

The oracle $O'_{x_1,y_1,\ldots,x_\ell,y_\ell}(\cdot)$ has three modes: the first mode takes inputs a function f, two indices (i,j) and returns $f(x_i,y_j)$. The second mode takes inputs a function f, an index i, a string z and outputs $f(x_i,z)$. The third mode is similar to the second except it outputs $f(z,y_i)$. The oracle $O'' = \mathcal{S}^{O'_{x_1,y_1,\ldots,x_\ell,y_\ell}}$. In the experiments, $\mathcal{A}_1$ and $\mathcal{A}_2$ make at most $q_1(\kappa)$, $q_2(\kappa)$ key queries respectively.

We say a stateful $\mathcal{S}$ is an admissible simulator if for every f that $\mathcal{S}$ queries to its oracle, the adversary $\mathcal{A}$ queries the same f to its (simulated) KeyGen *oracle.*

We say the scheme is (q_1,ℓ,q_2)-simulation secure if there exists a stateful admissible simulator $\mathcal{S}$ such that for all PPT adversaries $\mathcal{A}$, the experiments $\{\boldsymbol{Real}_{\mathcal{A},q_1(\kappa),\ell(\kappa),q_2(\kappa)}(1^\kappa)\}_{\kappa\in\mathbb{N}}$ and $\{\boldsymbol{Ideal}_{\mathcal{A},\mathcal{S},q_1(\kappa),\ell(\kappa),q_2(\kappa)}(1^\kappa)\}_{\kappa\in\mathbb{N}}$ are indistinguishable.

Definition 3.2 (Learnable binary functions) *Let $\mathcal{F} = \{\mathcal{F}_k\}_{\kappa\in\mathbb{N}}$ be a class of binary functions and $\mathcal{F}_\kappa$ contains some binary functions $\left\{f : \{0,1\}^{\mu(\kappa)} \times \{0,1\}^{\mu(\kappa)} \to \{0,1\}^{\mu'(\kappa)}\right\}$, for some polynomials μ,μ'. We say the class $\mathcal{F}$ is learnable if there exist learners L_1,L_2 such that for every κ, every partial input $x \in \{0,1\}^{\mu(\kappa)}$, every function $f \in \mathcal{F}_\kappa$, and every polynomial sized distinguisher D, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that* $\left|\Pr[f' \leftarrow L_1^{f(x,\cdot)}(1^\kappa,f) : D^{f'(\cdot)}(1^\kappa)=1] - \Pr[D^{f(x,\cdot)}(1^\kappa)=1]\right| < \mathsf{negl}(\kappa)$*, and similarly* $\left|\Pr[f' \leftarrow L_2^{f(\cdot,x)}(1^\kappa,f) : D^{f'(\cdot)}(1^\kappa)=1] - \Pr[D^{f(\cdot,x)}(1^\kappa)=1]\right| < \mathsf{negl}(\kappa)$.

We prove the following theorem, which says that SIM-secure public key binary-FE schemes exist only for learnable functions.

Theorem 3.3 *Let* FE *be a public-key binary functional encryption scheme for a class of binary functions $\mathcal{F} = \{\mathcal{F}_\kappa\}$. Assume* FE *is either $(1,1,0)$ or $(0,1,1)$-simulation secure, then then the class $\mathcal{F}$ is learnable.*

3.1.2 Constructing SIM-secure Public Key FE from Virtual Black Box Obfuscation

Next we show a construction of a binary FE scheme for learnable functions, assuming the existence of a CCA encryption scheme, a strongly unforgeable signature scheme, and a VBB obfuscator [3].

Let $v\mathcal{O}$ be a VBB obfuscator, $\mathsf{E} = \{\mathsf{KeyGen},\mathsf{Enc},\mathsf{Dec}\}$ be a CCA-secure encryption scheme, $\mathsf{E}' = \{\mathsf{KeyGen},\mathsf{Enc},\mathsf{Dec}\}$ be any semantically secure encryption scheme, and $\Sigma = \{\mathsf{KeyGen},\mathsf{Sign},\mathsf{Ver}\}$ be a strongly unforgeable signature scheme. Let $\mathcal{F}_\kappa = \{f : \mathcal{M}_\kappa \times \mathcal{M}_\kappa \to \mathcal{M}'_\kappa\}$ be a binary function that is learnable by some $p(\kappa)$-time algorithm for some polynomial $p(\cdot)$, where $\mathcal{M}_\kappa = \{0,1\}^{\mu(\kappa)}$ and $\mathcal{M}'_\kappa = \{0,1\}^{\mu'(\kappa)}$. Consider the following algorithms $\mathsf{FE}.\{\mathsf{Setup},\mathsf{KeyGen},\mathsf{Enc},\mathsf{Dec}\}$ parameterized by $q_1(\kappa),\ell(\kappa),q_2(\kappa)$. Let L be the maximum length of the class $\mathcal{F}_\kappa$, and K be the length of the ciphertexts in E. Then we consider the following algorithms:

- $\mathsf{FE.Setup}(1^\kappa)$: sample keys $(\mathsf{ek},\mathsf{dk}) \leftarrow \mathsf{E.KeyGen}(1^\kappa)$, $(\mathsf{ek}',\mathsf{dk}') \leftarrow \mathsf{E}'.\mathsf{KeyGen}(1^\kappa)$, and $(\mathsf{vk},\mathsf{sk}) \leftarrow \Sigma.\mathsf{KeyGen}(1^\kappa)$. Then sample $\hat{D}_{\mathsf{dk},\mathsf{dk}',\mathsf{vk}} \leftarrow v\mathcal{O}(D_{\mathsf{dk},\mathsf{dk}',\mathsf{vk}})$, where $D_{\mathsf{dk},,\mathsf{dk}'\mathsf{vk}}$ is a circuit that does the following: on input ciphertexts $\mathsf{ct}_x,\mathsf{ct}_y,\mathsf{ct}_{\bar{f}}$ and a signature σ,
  - If $\Sigma.\mathsf{Ver}(\mathsf{vk},\mathsf{ct}_{\bar{f}},\sigma) = 0$, output $\perp$.

- Compute $f||0^t = \mathsf{E}'.\mathsf{Dec}(\mathsf{dk}', \mathsf{ct}_{\bar{f}})$ for some $t = \kappa + 2\ell \cdot (K + p) + \ell^2 \cdot \mu'$. Compute $(x, \mathtt{tag}_x, Z_x, f_x, b_x) = \mathsf{E}.\mathsf{Dec}(\mathsf{dk}, \mathsf{ct}_x)$ and $(y, \mathtt{tag}_y, Z_y, f_y, b_y) = \mathsf{E}.\mathsf{Dec}(\mathsf{dk}, \mathsf{ct}_y)$, where $\mathtt{tag}_x, \mathtt{tag}_y \in$ {"1", "2"}, $Z_x, Z_y \in \{0,1\}^{\mu' \cdot q_1 \cdot \ell}$, $f_x, f_y \in \{0,1\}^p$, $b_x, b_y \in \{0,1\}^\kappa$. If any of the ciphertexts is not of the correct form, then output $\perp$.
- If $(\mathtt{tag}_x, \mathtt{tag}_y) \neq$ ("1","2"), output $\perp$.
- Otherwise, output $f(x, y)$.

Finally, set $\mathsf{param} := (\mathsf{ek}, \mathsf{ek}', \mathsf{vk}, \hat{D}_{\mathsf{dk},\mathsf{dk}',\mathsf{vk}})$, $\mathsf{msk} := (\mathsf{dk}, \mathsf{dk}', \mathsf{sk})$.
(We note that $f_x, f_y, Z_x, Z_y, b_x, b_y$ will only be used in the simulated param.)

- $\mathsf{FE}.\mathsf{Enc}(\mathsf{param}, x, \mathtt{tag})$: on inputs a message x and a tag $\mathtt{tag} \in$ {"1","2"}, output $\mathsf{ct} \leftarrow \mathsf{E}.\mathsf{Enc}(\mathsf{ek}, (x, \mathtt{tag}, 0^{\mu' \cdot q_1}, 0^p, 0^\kappa))$.
- $\mathsf{FE}.\mathsf{KeyGen}(\mathsf{msk}, f)$: first compute an encryption of the padded function $f||0^t$ for some $t = \kappa + 2\ell \cdot (K + p) + \ell^2 \cdot \mu'$, i.e. $\mathsf{ct}_{\bar{f}} \leftarrow \mathsf{E}'.\mathsf{Enc}(\mathsf{ek}', f||0^t)$. Then generate a signature $\sigma \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}, \mathsf{ct}_{\bar{f}})$. Output $\mathsf{sk}_f := (\mathsf{ct}_{\bar{f}}, \sigma)$.

  Note: the extra 0's will only be used in the simulation.
- $\mathsf{FE}.\mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct}_x, \mathsf{ct}_y)$: run $\mathsf{Eval}(\hat{D}_{\mathsf{dk},\mathsf{vk}}, \mathsf{ct}_x, \mathsf{ct}_y, \mathsf{ct}_{\bar{f}}, \sigma)$, where sk_f can be parsed as $(\mathsf{ct}_{\bar{f}}, \sigma)$.

Then we establish the following theorem:

Theorem 3.4 *Assume $v\mathcal{O}$ is a VBB obfuscator for general functions, E is a CCA secure encryption scheme, E' is semantically secure, Σ is a strongly unforgeable signature scheme. Then for any polynomial q_1, ℓ, q_2, the above public-key binary FE scheme is (q_1, ℓ, q_2)-simulation secure.*

3.1.3 Constructing SIM-secure Public Key FE from IND-secure Public Key FE

In this section we show how to lift IND-secure public key binary FE to SIM-secure public key binary FE. The compiler is generic, and is a natural extension of the compiler described in Caro et al. [14], applied to the binary FE setting.

Let $\mathsf{FE}_{\mathsf{ind}}$ be a binary, indistinguishability secure, public key FE scheme, let commit be a statistically binding commitment scheme, and let E be a semantically secure symmetric key encryption scheme. Let $\mathcal{F}_\kappa = \{f : \mathcal{M}_\kappa \times \mathcal{M}_\kappa \to \mathcal{M}'_\kappa\}$ be a binary function that is learnable by some $p(\kappa)$-time algorithm for some polynomial $p(\cdot)$, where $\mathcal{M}_\kappa = \{0,1\}^{\mu(\kappa)}$ and $\mathcal{M}'_\kappa = \{0,1\}^{\mu'(\kappa)}$. Let L be the maximum length of the class $\mathcal{F}_\kappa$. Consider the following algorithms $\mathsf{FE}.\{\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}\}$ parameterized by $q_1(\kappa), \ell(\kappa), q_2(\kappa)$.

- $\mathsf{FE}.\mathsf{Setup}(1^\kappa)$: Choose a random $r \leftarrow \{0,1\}^\kappa$ and compute $\mathsf{com} = \mathsf{commit}(0^\kappa; r)$.
  Run $(\mathsf{param}_{\mathsf{ind}}, \mathsf{msk}_{\mathsf{ind}}) \leftarrow \mathsf{FE}_{\mathsf{ind}}.\mathsf{Setup}(1^\kappa)$.
  Set the public parameters to be $\mathsf{param} = (\mathsf{param}_{\mathsf{ind}}, \mathsf{com})$ and $\mathsf{msk} = (\mathsf{msk}_{\mathsf{ind}}, r)$.
- $\mathsf{FE}.\mathsf{Enc}(\mathsf{param}, x, \mathtt{tag})$: on inputs a message x and a tag $\mathtt{tag} \in$ {"1","2"}, sample $\mathsf{ct} \leftarrow \mathsf{FE}_{\mathsf{ind}}.\mathsf{Enc}(\mathsf{param}_{\mathsf{ind}}, (x, \mathtt{tag}, 0^\kappa, 0^\kappa, 0))$.
- $\mathsf{FE}.\mathsf{KeyGen}(\mathsf{msk}, f)$: sample $\tau \leftarrow \mathsf{E}.\mathsf{KeyGen}(1^\kappa)$. Let $c_1, c_2 \leftarrow \mathsf{E}.\mathsf{Enc}(\tau, 0^{p+\mu'})$.

  Output $\mathsf{FE}_{\mathsf{ind}}.\mathsf{KeyGen}(\mathsf{msk}, \hat{f}_{c_1,c_2})$, where, for any $\bar{c}_1, \bar{c}_2$ in the ciphertext space of E, $\hat{f}_{\bar{c}_1,\bar{c}_2}(x', y')$ is defined as follows.

  - Parse $x' = (x, \mathtt{tag}_x, \tau_x, r_x, b_x)$ and $y' = (y, \mathtt{tag}_y, \tau_y, r_y, b_y)$. If $(\mathtt{tag}_x, \mathtt{tag}_y) \neq$ ("1", "2") output $\perp$.

- If $(b_x, b_y) = (0,0)$, then output $f(x,y)$.
- If $(b_x, b_y) = (0,1)$, then, if $\mathsf{com} = \mathsf{commit}(\tau_y; r_y)$, compute $(f_y, Z_y) = \mathsf{E.Dec}(\tau_y, \bar{c}_2)$, and output $f_y(x)$, where the string $f_y \in \{0,1\}^p$ is interpreted as a function. Otherwise output $\perp$.
- If $(b_x, b_y) = (1,0)$, then, if $\mathsf{com} = \mathsf{commit}(\tau_x, r_x)$, compute $(f_x, Z_x) = \mathsf{E.Dec}(\tau_x, \bar{c}_1)$ and output $f_x(y)$. Otherwise output $\perp$.
- If $(b_x, b_y) = (1,1)$, then if $\mathsf{com} = \mathsf{commit}(\tau_x; r_x)$ and $\tau_x = \tau_y$, compute $(f_x, Z_x) = \mathsf{E.Dec}(\tau_x, \bar{c}_1)$, and output Z_x. Otherwise output $\perp$.

- $\mathsf{FE.Dec}(\mathsf{sk}_f, \mathsf{ct}_x, \mathsf{ct}_y)$: run $\mathsf{FE}_{\mathsf{ind}}(\mathsf{sk}_f, \mathsf{ct}_x, \mathsf{ct}_y)$.

Then we establish the following theorem:

Theorem 3.5 *Assume* $\mathsf{FE}_{\mathsf{ind}}$ *is an IND-secure public-key binary functional encryption scheme,* commit *is a statistically binding commitment scheme, and* E *is a semantically secure symmetric key encryption scheme. Then the above scheme is* $(0, 1, \mathrm{poly})$ *simulation-secure.*

Remark 3.6 As presented above, we handle an unlimited number of post-challenge key queries. We note that this does not violate the lower bound of Agrawal et al. [1], which only applies to pre-challenge queries. We can extend this to handle multiple challenge plaintexts in a natural way: in each simulated key token, we simply need to embed f_x and Z_x for every x in the challenge list. We then also add an additional slot in the plaintext that remains 0 in the real world, but allows the simulator to indicate the index of the challenge in the list. For ℓ challenge plaintexts, then, the function $\hat{f}_{c_1,\ldots,c_\ell}$ would recover the index of the challenge, decrypt the corresponding c_i, and use the function description recovered from c_i exactly as described above. Now the key size would grow with the number of challenges, which matches the lower bounds proven by Boneh et al. [10] and by Bellare and O'Neill [5].

3.2 The Symmetric-Key Setting

In this section we show how to lift IND-secure symmetric key binary-FE to SIM-secure symmetric key binary-FE. The compiler is generic, and is a natural extension of the compiler described in Caro et al. [14], applied to the binary FE setting. The syntax and security definition is very similar to that described in Section 3.1.1, except that now, instead of using param, the encryptor will use msk to encrypt plaintexts.

Let $\mathcal{F} = \{\mathcal{F}_\kappa\}_{\kappa>0}$ be a collection of function families, where every $f \in \mathcal{F}_\kappa$ is a polynomial time function $f : \mathcal{M}_\kappa \times \mathcal{M}_\kappa \to \mathcal{M}'_\kappa$, where $\mathcal{M}_\kappa = \{0,1\}^{\mu(\kappa)}$ and $\mathcal{M}'_\kappa = \{0,1\}^{\mu'(\kappa)}$. Let $\mathsf{FE}_{\mathsf{ind}}$ be an IND-secure, symmetric key binary-FE scheme supporting $\mathcal{F}$, and let E be a semantically secure symmetric key encryption scheme. Consider the following algorithms $\mathsf{FE}.\{\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}\}$.

- $\mathsf{FE.Setup}(1^\kappa)$: Choose a random $\tau \leftarrow \mathsf{E.KeyGen}(1^\kappa)$. Run $(\mathsf{param}_{\mathsf{ind}}, \mathsf{msk}_{\mathsf{ind}}) \leftarrow \mathsf{FE}_{\mathsf{ind}}.\mathsf{Setup}(1^\kappa)$. Set the public parameters to be $\mathsf{param} = \mathsf{param}_{\mathsf{ind}}$ and $\mathsf{msk} = (\mathsf{msk}_{\mathsf{ind}}, \tau)$.
- $\mathsf{FE.Enc}(\mathsf{msk}, x, \mathtt{tag})$: on inputs a message x and a tag $\mathtt{tag} \in \{$"1","2"$\}$, sample $\mathsf{ct} \leftarrow \mathsf{FE}_{\mathsf{ind}}.\mathsf{Enc}(\mathsf{msk}_{\mathsf{ind}}, (x, \mathtt{tag}, 0^\kappa, 0^{\mu'}, 0^\kappa, 0))$.
- $\mathsf{FE.KeyGen}(\mathsf{msk}, f)$: Sample $c \leftarrow \mathsf{E.Enc}(\tau, 0^{\mu'})$. Choose a random $\mathsf{idx}_f \leftarrow \{0,1\}^\kappa$. Output $\mathsf{FE}_{\mathsf{ind}}.\mathsf{KeyGen}(\mathsf{msk}, \hat{f}_{c,\mathsf{idx}_f})$, where, for any $\bar{c}$ in the ciphertext space of E, and $\mathsf{idx}_f \in \{0,1\}^\kappa$, $\hat{f}_{\bar{c},\mathsf{idx}_f}(x', y')$ is defined as follows.
  - Parse $x' = (x, \mathtt{tag}_x, (\mathsf{idx}_x, Z_x), \tau_x, b_x)$ and $y' = (y, \mathtt{tag}_y, (\mathsf{idx}_y, Z_y), \tau_y, b_y)$. If $(\mathtt{tag}_x, \mathtt{tag}_y) \neq$ ("1", "2") output $\perp$.

- If $(b_x, b_y) = (0, 0)$, then output $f(x, y)$.
- If $b_y = 1$,
  - if $\mathsf{idx}_f = \mathsf{idx}_y$, output Z_y.
  - Otherwise, output $\mathsf{E.Dec}(\tau_y, \bar{c})$.
- If $b_x = 1$,
  - if $\mathsf{idx}_f = \mathsf{idx}_x$, output Z_x.
  - Otherwise, output $\mathsf{E.Dec}(\tau_x, \bar{c})$.

- $\mathsf{FE.Dec}(\mathsf{sk}_f, \mathsf{ct}_x, \mathsf{ct}_y)$: run $\mathsf{FE}_{\mathsf{ind}}(\mathsf{sk}_f, \mathsf{ct}_x, \mathsf{ct}_y)$.

Then we establish the following theorem:

Theorem 3.7 *Assume* $\mathsf{FE}_{\mathsf{ind}}$ *is an IND-secure binary functional encryption scheme for functionality* $\mathcal{F}$*, and that* E *is a semantically secure symmetric key encryption scheme. Then the above scheme is a* $(1, 1, 1)$*-simulation-secure symmetric-key binary FE scheme for functionality* $\mathcal{F}$*.*

Remark 3.8 As presented above, we handle only a single pre-challenge key query, a single post-challenge key query and a single challenge plaintext pair. It is quite straightforward to extend the ideas above to achieve $q_1(\kappa), \ell(\kappa), q_2(\kappa)$ security, as in the work of Caro et al.[14]. To handle $q_1(\kappa)$ pre-challenge queries, $f_1, \ldots, f_{q_1(\kappa)}$, we simply embed $(\mathsf{idx}_{f_i}, f_i(x, y))$ in each simulated ciphertext for each $i \in [\mathrm{poly}(\kappa)]$. To handle $\ell(\kappa)$ challenge plaintext pairs, $(x_1, y_1), \ldots, (x_{\ell(\kappa)}, y_{\ell(\kappa)})$, to encrypt, say x_j, we include $(\mathsf{idx}_{f_i}, f_i(x_j, x_k))$ in the ciphertext for every $i \in [\mathrm{poly}(\kappa)], k \in [\ell(\kappa)]$. Finally, to support $q_2(\kappa)$ post-challenge queries, $(f_1, \ldots, f_{q_2(\kappa)})$, in the key for f_i, we embed an encryption of $f_i(x_j, x_k)$ for every $j, k \in [q_2(\kappa)]$. We note that the key size would grow quadratically in $\ell(\kappa)$ (i.e. in the number of challenge plaintexts), and the ciphertext size would grow as the product of $q_1(\kappa) \cdot \ell(\kappa)$. We can support an unbounded number of post-challenge queries in this manner.

Acknowledgements

We gratefully thank Kai-Min Chung and Hubert Chan for helpful discussions.

References

[1] Shweta Agrawal, Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption: New perspectives and lower bounds. In *CRYPTO (2)*, pages 500–518, 2013.

[2] Prabhanjan Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. Differing-inputs obfuscation and applications. Cryptology ePrint Archive, Report 2013/689, 2013. `http://eprint.iacr.org/`.

[3] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S.P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *Journal of the ACM*, 59(2), 2012.

[4] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. Cryptology ePrint Archive, Report 2013/631, 2013. http://eprint.iacr.org/.

[5] Mihir Bellare and Adam O'Neill. Semantically-secure functional encryption: Possibility results, impossibility results and the quest for a general definition. *IACR Cryptology ePrint Archive*, 2012:515, 2012.

[6] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *2007 IEEE Symposium on Security and Privacy*, pages 321–334. IEEE Computer Society Press, May 2007.

[7] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *STOC*, pages 103–112, 1988.

[8] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'Neill. Order-preserving symmetric encryption. In Antoine Joux, editor, *Advances in Cryptology – EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 224–241. Springer, April 2009.

[9] Alexandra Boldyreva, Nathan Chenette, and Adam O'Neill. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 578–595. Springer, August 2011.

[10] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *TCC 2011: 8th Theory of Cryptography Conference*, volume 6597 of *Lecture Notes in Computer Science*, pages 253–273. Springer, 2011.

[11] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *TCC 2007: 4th Theory of Cryptography Conference*, volume 4392 of *Lecture Notes in Computer Science*, pages 535–554. Springer, 2007.

[12] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. Cryptology ePrint Archive, Report 2013/650, 2013. `http://eprint.iacr.org/`.

[13] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. Cryptology ePrint Archive, Report 2013/563, 2013. http://eprint.iacr.org/.

[14] Angelo De Caro, Vincenzo Iovino, Abhishek Jain, Adam O'Neill, Omer Paneth, and Giuseppe Persiano. On the achievability of simulation-based security for functional encryption. In *CRYPTO (2)*, pages 519–535, 2013.

[15] Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Carlos Cid. Multi-client non-interactive verifiable computation. In *TCC*, pages 499–518, 2013.

[16] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple noninteractive zero knowledge proofs under general assumptions. *SIAM J. Comput.*, 29(1):1–28, 1999.

[17] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013. http://eprint.iacr.org/2013/451.

[18] Sanjam Garg, Craig Gentry, Shai Halevi, Amit Sahai, and Brent Waters. Attribute-based encryption for circuits from multilinear maps. In *CRYPTO*, pages 479–499, 2013.

[19] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *STOC*, pages 467–476, 2013.

[20] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. In *CRYPTO*, pages 536–553, 2013.

[21] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *STOC*, pages 555–564, 2013.

[22] Shafi Goldwasser and Guy N. Rothblum. On best-possible obfuscation. In *TCC 2007: 4th Theory of Cryptography Conference*, volume 4392 of *Lecture Notes in Computer Science*, pages 194–213. Springer, 2007.

[23] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 162–179. Springer, 2012.

[24] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In *STOC*, pages 545–554, 2013.

[25] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM CCS 06: 13th Conference on Computer and Communications Security*, pages 89–98. ACM Press, 2006.

[26] Jens Groth. Minimizing non-interactive zero-knowledge proofs using fully homomorphic encryption. In *Manuscript*, 2011.

[27] J. Katz, A. Sahai, and B. Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. *Journal of Cryptology*, 26(2):191–224, 2013.

[28] Omkant Pandey and Yannis Rouselakis. Property preserving symmetric encryption. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 375–391. Springer, April 2012.

[29] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *TCC 2012: 9th Theory of Cryptography Conference*, volume 7194 of *Lecture Notes in Computer Science*, pages 422–439. Springer, 2012.

[30] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th Annual Symposium on Foundations of Computer Science*, pages 543–553. IEEE Computer Society Press, October 1999.

[31] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In *ACM CCS 10: 17th Conference on Computer and Communications Security*, pages 463–472. ACM Press, 2010.

[32] Emily Shen, Elaine Shi, and Brent Waters. Predicate privacy in encryption systems. In *TCC 2009: 6th Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*, pages 457–473. Springer, 2009.

[33] Elaine Shi, T.-H. Hubert Chan, Eleanor G. Rieffel, Richard Chow, and Dawn Song. Privacy-preserving aggregation of time-series data. In *ISOC Network and Distributed System Security Symposium – NDSS 2011*. The Internet Society, February 2011.

A Preliminaries

A.1 Statistical Simulation-Sound NIZKs for Multiple Simulated Statements

Let R be a polynomial-time computable binary relation. For $(\mathsf{stmt}, w) \in R$, we call stmt the statement, and w the witness . Let L be the language consisting of all statements in R.

A Non-Interactive Zero-knowledge Proof system (NIZK) [7, 16] is a collection of three algorithms $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$. Let m be a parameter for the maximum number of false statements to be simulated m in the system. Consider the following three algorithms:

- $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\kappa, m)$: Takes in the security parameter κ, and generates a common reference string crs.
- $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, \mathsf{stmt}, w)$: Takes in crs, a statement stmt, and a witness w such that $(\mathsf{stmt}, w) \in L$, outputs a proof π.
- $b \leftarrow \mathsf{Verify}(\mathsf{crs}, \mathsf{stmt}, \pi)$: Takes in the crs, a statement stmt, and a proof π, and outputs 0 or 1, denoting rejection or acceptance. stmt, and a witness w,

Perfect completeness. A NIZK system is said to be perfectly complete, if an honest prover with a valid witness can always convince an honest verifier. More formally, for any $(\mathsf{stmt}, w) \in R$, we have

$$\Pr\left[\mathsf{crs} \leftarrow \mathsf{Setup}(1^\kappa, m),\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, \mathsf{stmt}, w) :\ \ \mathsf{Verify}(\mathsf{crs}, \mathsf{stmt}, \pi) = 1\right] = 1$$

Statistical soundness. A NIZK system is said to be statistically sound, if there does not exist a valid proof for any no false statement. More formally,

$$\Pr\left[\mathsf{crs} \leftarrow \mathsf{Setup}(1^\kappa, m),\ \exists(\mathsf{stmt}, \pi):\ (\mathsf{stmt} \notin L)\ \wedge\ (\mathsf{Verify}(\mathsf{crs}, \mathsf{stmt}, \pi) = 1)\right] = \mathsf{negl}(\kappa)$$

Computational zero-knowlege. Informally, a NIZK system is computationally zero-knowledge, if the proof does not reveal any information about the witness to any polynomial-time adversary. More formally, a NIZK system is said to computationally zero-knowledge, if there exists a simulator $S = (\mathsf{SimSetup}, \mathsf{SimProve})$, such that for all non-uniform polynomial-time adversary $\mathcal{A}$, for $m = \mathrm{poly}(\kappa)$, for any $\{\mathsf{stmt}_i, w_i\}_{i\in[m]}$ such that $(\mathsf{stmt}_i, w_i) \in R$ for all $i \in [m]$, it holds that

$$\left| \Pr\left[\begin{array}{l}\mathsf{crs} \leftarrow \mathsf{Setup}(1^\kappa, m),\\ \forall i \in [m] : \pi_i \leftarrow \mathsf{Prove}(\mathsf{crs}, \mathsf{stmt}_i, w_i) :\\ \mathcal{A}(\mathsf{crs}, \{\mathsf{stmt}_i, \pi_i\}_{i\in[m]}) = 1\end{array}\right] - \Pr\left[\begin{array}{l}(\widetilde{\mathsf{crs}}, \mathsf{trap}) \leftarrow \mathsf{SimSetup}(1^\kappa, \{\mathsf{stmt}_i\}_{i\in[m]}),\\ \forall i \in [m] : \pi_i \leftarrow \mathsf{SimProve}(\mathsf{crs}, \mathsf{stmt}_i, \mathsf{trap}) :\\ \mathcal{A}(\widetilde{\mathsf{crs}}, \{\mathsf{stmt}_i, \pi_i\}_{i\in[m]}) = 1\end{array}\right] \right| = \mathsf{negl}(\kappa)$$

Statistical simulation soundness [17]. Informally, a NIZK system is statistically simulation sound, if under a simulated $\widetilde{\mathsf{crs}}$, no proof for a false statement exists, except for the simulated proofs for statements fed into the $\mathsf{SimSetup}$ algorithm to generate $\widetilde{\mathsf{crs}}$. More formally, a NIZK system is said to be statistically simulation sound, if for $m = \mathrm{poly}(\kappa)$,

$$\Pr\left[\begin{array}{l}(\widetilde{\mathsf{crs}}, \mathsf{trap}) \leftarrow \mathsf{SimSetup}(1^\kappa, \{\mathsf{stmt}_i\}_{i\in[m]}), \forall i \in [m] : \pi_i \leftarrow \mathsf{SimProve}(\mathsf{crs}, \mathsf{stmt}_i, \mathsf{trap}) :\\ \exists(\mathsf{stmt}', \pi') \text{ s.t. } (\mathsf{stmt}' \notin \{\mathsf{stmt}_i\}_{i\in[m]})\ \wedge\ (\mathsf{Verify}(\widetilde{\mathsf{crs}}, \mathsf{stmt}', \pi') = 1)\end{array}\right] = \mathsf{negl}(\kappa).$$

A.2 SSS-NIZK Construction for Multiple Simulated Statements

Garg et al. [17] showed how to construct an SSS-NIZK where the simulator can simulate a single false statement (specified in the simulated setup). We can easily extend their construction to allow to allow simulation of a multiple number of false statements. We state the construction in the following:

Let $\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ be a statistically sound NIZK, and Com be a perfectly binding commitment scheme. Then consider the following NIZK system $\Pi' = (\mathsf{Setup}', \mathsf{Prove}', \mathsf{Verify}')$:

- $\mathsf{Setup}'(1^\kappa, m)$: generate $\sigma \leftarrow \mathsf{Setup}(1^\kappa)$, and $c_1 \leftarrow \mathsf{Com}(0^\ell; r_1), \ldots, c_m \leftarrow \mathsf{Com}(0^\ell; r_m)$. Output $\mathsf{crs} = (\sigma, c_1, \ldots, c_m)$.
- $\mathsf{Prove}'(\mathsf{crs}, x, w)$: generate a proof π using $\mathsf{Prove}(\sigma, x', w)$ where x' is the following statement:

$$\exists (w, i, r) \text{ s.t. } ((x, w) \in R) \bigvee (c_i = \mathsf{Com}(x; r)) .$$

- $\mathsf{Verify}'(\mathsf{crs}, x, \pi)$: output $\mathsf{Verify}(\sigma, x', \pi)$, where x' is the above statement.

Since the above construction extends the construction by Garg et al. in a strait forward way, we omit the proof and refer curious readers to the paper [17].

In the above construction, the common reference string blows up by a factor of m, so does the prover and verifier's running times. However, the length of proofs does not need to blow up with m. We observe that we can use a trick by Groth [26], to create a statistically sound NIZK whose proof length is linear in the length of the witness. Since the witness of the statement (i.e. (w, i, r)) does not grow with m, neither does the length of the proof for the Π'.

A.3 (Computationally) Simulation Sound NIZK

We state the (unbounded) simulation soundness as defined by Sahai [30].

A non-interactive zero-knowledge proof system is said to be (computationally) simulation sound for unbounded statements if for all non-uniform PPT adversary $\mathcal{A}$, $\mathcal{A}$ wins the following game with a negligible probability.

The game begins with an execution of the simulator of the NIZK proof system who generates a CRS together with a trapdoor. Then $\mathcal{A}$ is given oracle access to a simulator who has the trapdoor and produces simulated proofs (which is accepted by the verifier) of any statements (either true or false) $\mathcal{A}$ queries. The queries of statements can be adaptive. The game ends with $\mathcal{A}$ outputting (x, π). He wins if x is a false statement, π is accepted by the verifier, and (x, π) is not in the query list.

A.4 Indistinguishability Obfuscation

A uniform PPT machine $i\mathcal{O}$ is called an indistinguishable obfucastor [22, 17], for a circuit family $\{\mathcal{C}_\kappa\}$, if the following conditions hold:

- **Correctness.** For all $\kappa \in \mathbb{N}$, for all $C \in \mathcal{C}_\kappa$, for all inputs x, we have

$$\Pr\left[C' \leftarrow i\mathcal{O}(\kappa, C) : \ C'(x) = C(x)\right] = 1$$

- For any uniform or non-uniform PPT distinguisher D, for all security parameter $\kappa \in \mathbb{N}$, for all pairs of circuits $C_0, C_1 \in \mathcal{C}_\kappa$ such that $C_0(x) = C_1(x)$ for all inputs x, then

$$|\Pr\left[D(i\mathcal{O}(\kappa, C_0)) = 1\right] - \Pr\left[D(i\mathcal{O}(\kappa, C_1)) = 1\right]| \leq \mathsf{negl}(\kappa)$$

For simplicity, when the security parameter κ is clear, we write $i\mathcal{O}(C)$ in short.

Remark A.1 We observe that for the construction of [17], the size of the obfuscated circuit has only linear blowup, i.e. $|i\mathcal{O}(C)| = |C| \cdot \mathrm{poly}(\kappa)$. In the construction, in order to obfuscate any poly-sized C, they consider another NC1 program P that checks the trace of the computation of the FHE evaluation, and then decrypts. Essentially P is an "and" of $|C|$ local verifications (each of which has size $\mathrm{poly}(\kappa)$), and then a decryption, which has size $\mathrm{poly}(\kappa)$. We observe that each local verification can be written as a constant width, $\mathrm{poly}(\kappa)$ length branching program by the Barrington's theorem; also it takes an additive linear blow up to "AND" multiple branching programs. Thus, P can be transformed to a branching program of size $O(|C|) \cdot \mathrm{poly}(\kappa)$. Then the construction of [17] blows up the branching program by $\mathrm{poly}(\kappa)$. Thus, the obfuscated circuit has size $O(|C|) \cdot \mathrm{poly}(\kappa)$.

A.5 Differing-inputs Obfuscation for Circuits

Barak et al. [3] defined the notion of differing-inputs obfuscation. We present the notion of differing-inputs circuit family as the formulation in the works of Ananth et al. and Boyle et. al [2, 12]

Definition A.2 ([3, 2, 12]) *A circuit family $\mathcal{C}$ associated with a sampler* $\mathsf{Sampler}$ *is said to be a differing-inputs circuit family if for every PPT adversary $\mathcal{A}$ there exists a negligible function* negl *such that*

$$\Pr[C_0(x) \neq C_1(x) : (C_0, C_1, \mathsf{aux}) \leftarrow \mathsf{Sampler}(1^\kappa), x \leftarrow \mathcal{A}(1^\kappa, C_0, C_1, \mathsf{aux})] \leq \mathsf{negl}(\kappa).$$

We now define the notion of differing-inputs obfuscation for a differing-inputs circuit family.

Definition A.3 (Differing-inputs Obfuscators for circuits) *A uniform PPT machine* $\mathsf{di}\mathcal{O}$ *is called a Differing-inputs Obfuscator for a differing-inputs circuit family $\mathcal{C} = \{C_\kappa\}$ if the following conditions are satisfied:*

- *(Correctness): For all security parameter κ, all $C \in \mathcal{C}$, all inputs x, we have*

$$\Pr[C'(x) = C(x) : C' \leftarrow \mathsf{di}\mathcal{O}(\kappa, C)] = 1.$$

- *(Polynomial slowdown): There exists a universal polynomial p such that for any circuit C, we have $|C'| \leq p(|C|)$ for all $C' = \mathsf{di}\mathcal{O}(\kappa, C)$ under all random coins.*

- *(Differing-inputs): For any (not necessarily uniform) PPT distinguisher D, there exists a negligible function* negl *such that the following holds: for all security parameters κ, for $(C_0, C_1, \mathsf{aux}) \leftarrow \mathsf{Sampler}(1^\kappa)$, we have that*

$$|\Pr[D(\mathsf{di}\mathcal{O}(\kappa, C_0, \mathsf{aux})) = 1] - \Pr[D(\mathsf{di}\mathcal{O}(\kappa, C_1, \mathsf{aux})) = 1]| \leq \mathsf{negl}(\kappa).$$

B IND-Security: Proofs

B.1 IND-secure Public Key Binary FE from $i\mathcal{O}$

In this section we prove Theorem 2.2:

Theorem B.1 (Theorem 2.2) *If $i\mathcal{O}$ is indistinguishability obfuscation,* NIZK *is statistically simulation sound NIZK, and the encryption scheme is semantically secure and perfectly correct, then the above construction is selectively, IND-secure, as defined in Section 2.1.1.*

We define the following hybrid games, and then prove that each successive game is indistinguishable from its predecessor.

Hybrid X: This game is simply the IND-Secure defined above with $b = 0$. In other words, the challenger will encrypt x (in both copies of the ciphertexts c and c') and return them to the adversary.

Hybrid 1: In this game, the crs generation is simulated, and the proof π used during encryption is simulated. We note that here we are relying on the fact that IND-Secure is only selectively secure, which allows the challenger to construct statements stmt in Step 4 before constructing the simulated $\widetilde{\mathsf{crs}}$. More specifically, the game becomes the following (changes from Hybrid X are in red):

1. $(x, y) \leftarrow \mathcal{A}(1^\kappa)$
2. $\mathsf{Setup}(1^\kappa)$
   - $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa), (\mathsf{pk}', \mathsf{sk}') \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$
   - $\mathsf{msk} = (\mathsf{sk}, \mathsf{sk}')$
3. $c \leftarrow \mathcal{E}.\mathsf{Enc}(\mathsf{pk}, x)$ and $c' \leftarrow \mathcal{E}.\mathsf{Enc}(\mathsf{pk}', x)$.
   Let $\mathsf{stmt} := (c, c')$ be a statement of the language $L_{\mathsf{pk},\mathsf{pk}'}$.
   $(\widetilde{\mathsf{crs}}, \mathsf{trap}) \leftarrow \mathsf{NIZK.SimSetup}(1^\kappa, \mathsf{stmt})$.
   $\mathsf{param} = (\widetilde{\mathsf{crs}}, \mathsf{pk}, \mathsf{pk}')$.
   $\pi = \mathsf{NIZK.SimProve}(\widetilde{\mathsf{crs}}, \mathsf{stmt}, \mathsf{trap})$.
4. $b' \leftarrow \mathcal{A}^{\widetilde{\mathsf{KeyGen}}(\cdot)}(\mathsf{param}, (c, c', \pi))$

The simulated $\widetilde{\mathsf{crs}}$ is also used in constructing responses to $\widetilde{\mathsf{KeyGen}}$ queries in Step 5. $\widetilde{\mathsf{KeyGen}}$ is the same as KeyGen other than using the simulated $\widetilde{\mathsf{crs}}$ instead.

Claim B.2 *If the NIZK is computationally zero knowledge, then Hybrid X and Hybrid 1 are indistinguishable.*

Proof: Let $\mathcal{A}$ be an adversary that distinguishes the two hybrids with advantage δ. We construct an adversary $\mathcal{B}$ that attacks the zero knowledge property of the NIZK system with the same advantage. After receiving (x, y) from $\mathcal{A}$, $\mathcal{B}$ runs Step 2 of Hybrid 1 honestly. He uses the resulting values to construct statement stmt, as done in Step 4 (of both hybrid games). He then submits stmt to a NIZK challenger, and receives (crs, π) in response. He uses these values to complete Step 4 of Hybrid 1. Specifically, he uses crs to complete the construction of param and to answer any queries to KeyGen that might be made in Step 5. He uses π to complete the construction of the challenge ciphertexts. If the NIZK challenger faithfully generated $\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\kappa)$ and $\mathsf{NIZK.Prove}(\mathsf{crs}, \mathsf{stmt})$, then the view of $\mathcal{A}$ is drawn from the same distribution as in Hybrid X. If the NIZK challenger generated simulated $(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathsf{NIZK.SimSetup}(1^\kappa, \mathsf{stmt})$ and $\mathsf{NIZK.SimProve}(\mathsf{crs}, \mathsf{stmt}, \mathsf{trap})$, then $\mathcal{A}$'s view is drawn from the same distribution as in Hybrid 1. ■

Hybrid 2: In Hybrid 2, the simulator will encrypt x in the first ciphertext copy c, and encrypt y for the second ciphertext copy c'. Intuitively, because the $i\mathcal{O}$ does not include sk', Hybrid 2 is computationally indistinguishable form Hybrid 1 due to the semantic security of the encryption scheme.

More formally, the new Hybrid 2 game is described as follows.

1. $(x, y) \leftarrow \mathcal{A}(1^\kappa)$
2. $\mathsf{Setup}(1^\kappa)$
   - $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa), (\mathsf{pk}', \mathsf{sk}') \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$
   - $\mathsf{msk} = (\mathsf{sk}, \mathsf{sk}')$

Internal (hardcoded) state: $\mathsf{param} = (\mathsf{crs}, \mathsf{pk}, \mathsf{pk}')$, sk', f

On input: $\mathsf{ct}_0, \mathsf{ct}_1$

- Unpack $(c_0, c_0', \pi_0) \leftarrow \mathsf{ct}_0$ and $(c_1, c_1', \pi_1) \leftarrow \mathsf{ct}_1$. Let $\mathsf{stmt}_0 := (c_0, c_0')$, and $\mathsf{stmt}_1 := (c_1, c_1')$ be statements for the NP-language $L_{\mathsf{pk},\mathsf{pk}'}$.
- If $\mathsf{NIZK.Verify}(\mathsf{crs}, \pi_0, \mathsf{stmt}_0) = \mathsf{NIZK.Verify}(\mathsf{crs}, \pi_1, \mathsf{stmt}_1) = 1$ then, compute $x = \mathcal{E}.\mathsf{Dec}(\mathsf{sk}', c_0')$ and $y = \mathcal{E}.\mathsf{Dec}(\mathsf{sk}', c_1')$, and output $f(x, y)$.
- Else, output $\perp$.

Figure 5: Public-key IND-secure binary FE: Program P'

3. $c \leftarrow \mathcal{E}.\mathsf{Enc}(\mathsf{pk}, x)$ and $c' \leftarrow \mathcal{E}.\mathsf{Enc}(\mathsf{pk}', y)$.
   Let $\mathsf{stmt} := (c, c')$ be a *false* statement.
   $(\widetilde{\mathsf{crs}}, \mathsf{trap}) \leftarrow \mathsf{NIZK.SimSetup}(1^\kappa, \mathsf{stmt})$.
   $\mathsf{param} = (\widetilde{\mathsf{crs}}, \mathsf{pk}, \mathsf{pk}')$.
   $\pi = \mathsf{NIZK.SimProve}(\widetilde{\mathsf{crs}}, \mathsf{stmt}, \mathsf{trap})$.
4. $b' \leftarrow \mathcal{A}^{\widetilde{\mathsf{KeyGen}}(\cdot)}(\mathsf{param}, (c, c', \pi))$

Claim B.3 *Assume that the encryptions scheme $\mathcal{E}$ is semantically secure, then Hybrid 2 is computationally indistinguishable from Hybrid 1.*

Proof: Observe that the $i\mathcal{O}$ does not embed sk'. Therefore, this claim follows from a trivial reduction. ■

Hybrid 3. In Hybrid 3, the simulator will now use sk' in the $i\mathcal{O}$ for any KeyGen query. The modified program P' is described in Figure 5, with modifications highlighted in red.

Claim B.4 *If the $i\mathcal{O}$ is secure, the NIZK is statistically simulation sound, and the encryption scheme $\mathcal{E}$ is perfectly correct, then Hybrid 2 is computationally indistinguishable from Hybrid 3.*

Proof: By the security of $i\mathcal{O}$, it suffices to show that Program P and Program P' are functionally equivalent, i.e., agree on the outputs for all possible inputs. The inputs to both programs P and P' are two ciphertexts ct_0 and ct_1. We do a case-by-case analysis.

- Case 1: There exists a *bad* ciphertext. If at least one of ct_0 or ct_1 causes the NIZK verification to fail, then both P and P' output $\perp$. Henceforth for Cases 2 and 3, we assume that both ct_0 and ct_1 carry a valid NIZK proof.
- Case 2: At least one of the inputs is a simulated ciphertext. Suppose the NIZKs on both ct_0 and ct_1 verify. Further, one of them (without loss of generality, assuming ct_0) is a ciphertext given to the adversary by the simulator, while the other (i.e, ct_1) is not. Due to the statistical simulation soundness of the NIZK and the perfect correctness of the encryption scheme, $\mathsf{ct}_1 := (c_1, c_1')$ must decrypt to the same value m no matter whether sk or sk' is used to decrypt. Due to the perfect correctness of the encryption scheme, ct_0 will decrypt to x or y depending on whether sk or sk' is used. Due to the condition that $f(x, \cdot) = f(y, \cdot)$ and $f(\cdot, x) = f(\cdot, y)$, we know that the outcomes of P and P' must agree.

- **Case 3:** Neither input is a simulated ciphertext. For both input ciphertexts: due to the statistical simulation soundness of the NIZK and the perfect correctness of encryption, using either key sk or sk' to decrypt must result in the same plaintext. Therefore, the outcomes of P and P' must agree.

■

Hybrid Y'. Hybrid Y' is almost the same as the real-world with $b = 1$, where the challenger encrypts the challenge plaintexts in y (in both ciphertext copies). The only exception is that the challenger uses sk' in the $i\mathcal{O}$'s.

Claim B.5 *Suppose that the $i\mathcal{O}$ is secure, and that the encryption scheme is semantically secure and perfectly correct, and that the NIZK is statistical simulation sound and computationally zero-knowledge, then Hybrid Y' is computationally indistinguishable from Hybrid 3.*

Proof: By a symmetric argument as the proofs for Hybrid X through Hybrid 3. ■

Hybrid Y. Hybrid Y is the same as the real-world with $b = 1$, where the challenger encrypts the challenge plaintexts in y (in both ciphertext copies). The challenger uses sk in the $i\mathcal{O}$'s.

Claim B.6 *Suppose that the NIZK is statistically sound, the encryption scheme is perfectly correct, and that the $i\mathcal{O}$ is secure, then Hybrid Y is computationally indistuinguishable from Hybrid Y'.*

Proof: By the statistical soundness of the NIZK, any ciphertext ct which has a valid NIZK must encrypt consistent plaintexts on both copies c and c'. Due to the perfect correctness of encryption, the two ciphertext copies must decrypt to the same plaintexts regardless of whether sk is used to decrypt c, or sk' is used to decrypt c'. Therefore, the two $i\mathcal{O}$s, using sk or sk' are functionally equivalent. By the security of $i\mathcal{O}$, Hybrid Y' is computationally indistinguishable from Hybrid Y. ■

B.2 IND-secure Symmetric Key Binary FE from $i\mathcal{O}$

In this section we prove Theorem 2.3.

Theorem B.7 (Theorem 2.3) *If the $i\mathcal{O}$ is secure, the NIZK is statistically simulation sound, the commitment is perfectly binding and computationally hiding, and the encryption scheme is semantically secure and perfectly correct, then the above construction is selectively IND-secure, as defined in Section 2.2.1*

We define the following hybrid games, and then prove that each successive game is indistinguishable from its predecessor.

Hybrid X: This game is simply the $\mathsf{Ind\text{-}Secure}$ defined above with $b = 0$. The simulator honestly encrypts x_i in both copies of the ciphertext c_i and c'_i.

Hybrid 1: In this game, the crs generation is simulated, and the proof π used during encryption is simulated. We note that here we are relying on the fact that $\mathsf{Ind\text{-}Secure}$ is only selectively secure, which allows the challenger to construct statement s in Step 4 before constructing the simulated crs. More specifically, the game becomes the following (changes from Hybrid X are in red):

1. $\{(x_1, \ldots, x_m), (y_1, \ldots, y_m)\} \leftarrow \mathcal{A}(1^\kappa)$
2. $\mathsf{Setup}(1^\kappa)$
   - $\alpha, r \leftarrow \{0,1\}^\kappa$; $\mathsf{com} \leftarrow \mathsf{commit}(\alpha; r)$

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$, $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$
- $\mathsf{msk} = (\mathsf{sk}, \mathsf{sk}', \alpha, r)$

3. For $i \in [m]$: $c_i \leftarrow \mathcal{E}.\mathsf{Enc}(\mathsf{pk}, x_i)$ and $c'_i \leftarrow \mathcal{E}.\mathsf{Enc}(\mathsf{pk}', x_i)$.
   Let stmt_i be the statement for the i-th ciphertext ct_i:
   $\exists (r, \rho, \rho'), x,$ s.t.: $(c_i = \mathcal{E}.\mathsf{Enc}(\mathsf{pk}; x; \rho)) \wedge \left(c'_i = \mathcal{E}.\mathsf{Enc}(\mathsf{pk}'; x; \rho')\right) \wedge (\mathsf{com} = \mathsf{commit}(\alpha; r))$
   $(\widetilde{\mathsf{crs}}, \mathsf{trap}) \leftarrow \mathsf{NIZK.SimSetup}(1^\kappa, \{\mathsf{stmt}_i\}_{i\in[m]})$.
   $\mathsf{param} = (\widetilde{\mathsf{crs}}, \mathsf{com})$.
   $\pi_i = \mathsf{NIZK.SimProve}(\widetilde{\mathsf{crs}}, \mathsf{stmt}_i, \mathsf{trap})$.
4. $b' \leftarrow \mathcal{A}^{\widetilde{\mathsf{KeyGen}}(\cdot)}(\mathsf{param}, \{\mathsf{ct}_i\}_{i\in[m]})$ where $\mathsf{ct}_i := (c_i, c'_i, \alpha, \pi_i)$

The simulated $\widetilde{\mathsf{crs}}$ is also used in constructing responses to $\widetilde{\mathsf{KeyGen}}$ queries in Step 5.

Claim B.8 *If the NIZK system is zero knowledge, then Hybrid 1 and Hybrid X are computationally indistinguishable.*

Proof: Let $\mathcal{A}$ be an adversary that distinguishes the two hybrids with advantage δ. We construct an adversary $\mathcal{B}$ that attacks the zero knowledge property of the NIZK system with the same advantage. After receiving $\{(x_1, \ldots, x_m), (y_1, \ldots, y_m)\}$ from $\mathcal{A}$, $\mathcal{B}$ runs Step 2 of Hybrid 1 honestly. He uses the resulting values to construct statements $\{\mathsf{stmt}_i\}_{i\in[m]}$, as done in Step 4 (of both hybrid games). He then submits $\{\mathsf{stmt}_i\}_{i\in[m]}$ to his NIZK challenger, and receives $(\mathsf{crs}, \{\pi_i\}_{i\in[m]})$ in response. He uses these values to complete Step 4 of Hybrid 1. Specifically, he uses crs to complete the construction of param and to simulate any queries to KeyGen that might be made in Step 5. He uses π to complete the construction of the challenge ciphertexts. If his own challenge was generated using $\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\kappa)$ and $\mathsf{NIZK.Prove}(\mathsf{crs}, \mathsf{stmt}_i)$ for $i \in [m]$, then the view of $\mathcal{A}$ is drawn from the same distribution as it is drawn from in Hybrid X. If his challenge was generated using $(\widetilde{\mathsf{crs}}, \mathsf{trap}) \leftarrow \mathsf{NIZK.SimSetup}(1^\kappa, \{\mathsf{stmt}_i\}_{i\in[m]})$ and $\mathsf{NIZK.SimProve}(\widetilde{\mathsf{crs}}, \mathsf{stmt}_i, \mathsf{trap})$ for $i \in [m]$, then $\mathcal{A}$'s view is drawn from the same distribution as it is drawn from in Hybrid 1. ■

Hybrid 2: In this game, Step 4 of Hybrid 1 is modified in the following way. A random $\widetilde{\alpha} \leftarrow \{0,1\}^\kappa$ is chosen and for $i \in [m]$, and used in the challenge ciphertexts $\{\mathsf{ct}_i\}$. The NIZK is now simulating false statements.

1. $\{(x_1, \ldots, x_m), (y_1, \ldots, y_m)\} \leftarrow \mathcal{A}(1^\kappa)$
2. $\mathsf{Setup}(1^\kappa)$
   - $\alpha, r \leftarrow \{0,1\}^\kappa$; $\mathsf{com} \leftarrow \mathsf{commit}(\alpha; r)$
   - $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$, $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$
   - $\mathsf{msk} = (\mathsf{sk}, \mathsf{sk}', \alpha, r)$
3. For $i \in [m]$: $c_i \leftarrow \mathcal{E}.\mathsf{Enc}(\mathsf{pk}, x_i)$ and $c'_i \leftarrow \mathcal{E}.\mathsf{Enc}(\mathsf{pk}', x_i)$.
   Choose $\widetilde{\alpha} \leftarrow \{0,1\}^\kappa$.
   Let stmt_i be the *false* statement for the i-th ciphertext ct_i:
   $\exists (r, \rho, \rho'), x,$ s.t.: $(c_i = \mathcal{E}.\mathsf{Enc}(\mathsf{pk}, x; \rho)) \wedge \left(c'_i = \mathcal{E}.\mathsf{Enc}(\mathsf{pk}', x; \rho')\right) \wedge (\mathsf{com} = \mathsf{commit}(\widetilde{\alpha}; r))$
   $(\widetilde{\mathsf{crs}}, \mathsf{trap}) \leftarrow \mathsf{NIZK.SimSetup}(1^\kappa, \{\mathsf{stmt}_i\}_{i\in[m]})$.
   $\mathsf{param} = (\widetilde{\mathsf{crs}}, \mathsf{com})$.
   $\pi_i = \mathsf{NIZK.SimProve}(\widetilde{\mathsf{crs}}, \mathsf{stmt}_i, \mathsf{trap})$.
4. $b' \leftarrow \mathcal{A}^{\widetilde{\mathsf{KeyGen}}(\cdot)}(\mathsf{param}, \{\mathsf{ct}_i\}_{i\in[m]})$ where $\mathsf{ct}_i := (c_i, c'_i, \widetilde{\alpha}, \pi_i)$

We note that the statement s is now *false* with probability $1 - 2^{\kappa}$, since the challenge ciphertexts $\{\mathsf{ct}_i\}$ now include $\widetilde{\alpha}$ in place of the α that was chosen during the construction of com (and $\Pr[\widetilde{\alpha} \neq \alpha] = 1 - 2^{\kappa}$).

Claim B.9 *If the commitment scheme* commit *is computationally hiding, then Hybrid 1 is computationally indistinguishable from Hybrid 2.*

Proof: Let $\mathcal{A}$ be an adversary that distinguishes the two hybrids with advantage δ. We construct an adversary $\mathcal{B}$ that attacks commit with the same advantage. $\mathcal{B}$ begins by randomly choosing $\alpha, \widetilde{\alpha} \leftarrow \{0,1\}^{\kappa}$. He submits these two strings to his challenger, and receives a commitment, c, to one of the two. When simulating Setup, he uses the received challenge c in place of com; the remainder of the simulation is done honestly, exactly as written in Hybrid 2. If $\mathcal{B}$'s challenge contains a commitment to $\widetilde{\alpha}$, then the view of $\mathcal{A}$ is drawn from the same distribution as it is drawn from in Hybrid 1. On the other hand, if $\mathcal{B}$ receives a commitment to α, then the view of $\mathcal{A}$ is drawn from the same distribution as it is drawn from in Hybrid 2. ■

Hybrid 3: In this game, Step 4 of Hybrid 2 is modified in the following way. For $i \in [m]$: $c_i \leftarrow \mathcal{E}.\mathsf{Enc}(\mathsf{sk}, x_i)$ and $c'_i \leftarrow \mathsf{Enc}(\mathsf{sk}', y_i)$. Everything else remains the same as in Hybrid 2. Note that now the two copies of ciphertexts c_i and c'_i do not encrypt consistent messages. However, since we are in a world with a simulated NIZK-CRS, the NIZKs can be simulated.

For Hybrids 2 and 3 to be computationally indistinguishable, it is important to observe that at this point, the $i\mathcal{O}$ only embeds secret key sk. A formal proof is presented below.

Claim B.10 *If the encryption scheme $\mathcal{E}$ is semantically secure, Hybrid 2 is computationally indistinguishable from Hybrid 3.*

Proof: Let $\mathcal{A}$ be an adversary that distinguishes the two hybrids with non-negligible probability. We construct an adversary $\mathcal{B}$ that attacks the CPA-security of $\mathcal{E}$ with non-negligible advantage. After receiving $\{(x_1, \ldots, x_m), (y_1, \ldots, y_m)\}$ from $\mathcal{A}$, $\mathcal{B}$ chooses $\alpha, r \leftarrow \{0,1\}^{\kappa}$ and computes $\mathsf{com} = \mathsf{commit}(\alpha; r)$. $\mathcal{B}$ then obtains pk from the CPA challenger. To simulate Step 4, $\mathcal{B}$ forwards $\{(x_1, \ldots, x_m), (y_1, \ldots, y_m)\}$ to the CPA challenger. He receives challenge $(c'_1, c'_2, \ldots, c'_m)$ in response. For $i \in [m]$, he computes $c_i \leftarrow \mathcal{E}.\mathsf{Enc}(\mathsf{pk}, x_i)$, and he sets $\mathsf{ct}_i = (c_i, c'_i)$. He continues the remainder of the simulation in the obvious way. Depending on the challenger's random coin, the adversary $\mathcal{A}$'s view is either the same distribution as Hybrid 2 or Hybrid 3. Therefore, if $\mathcal{A}$ could distinguish Hybrid 2 and Hybrid 3 with non-negligible probability, the simulator $\mathcal{B}$ could break semantic security of the underlying encryption scheme. ■

Hybrid 4: Hybrid 4 is the same as Hybrid 3, except that the responses to all KeyGen queries in Step 5 are modified to use Program P', as defined in Figure 6, in place of Program P. The difference between Program P' and Program P is that Program P uses the first secret key sk to decrypt, while Program P' uses the second secret key sk' to decrypt.

Claim B.11 *If the NIZK scheme is statistically simulation sound (SSS-NIZK), the commitment scheme is perfectly binding, the encryption scheme $\mathcal{E}$ is perfectly correct, and the $i\mathcal{O}$ scheme is secure, then, Hybrid 4 and Hybrid 3 are computationally indistinguishable.*

Proof: By the security of $i\mathcal{O}$, it suffices to show that Program P and Program P' are functionally equivalent (i.e., agree on all inputs). To prove that this is the case, we begin by defining three different types of possible input, and claim that all inputs fall into one of these three types. We assume (without loss of generality) that all inputs are of the form $\mathsf{ct} = (c, c', \widehat{\alpha}, \pi)$.

1. A *bad* ciphertext ct, where $\mathsf{NIZK.Verify}(\widetilde{\mathsf{crs}}, \mathsf{stmt}, \pi) = 0$.

Internal (hardcoded) state: $\mathsf{param} = (\mathsf{crs}, \mathsf{pk}, \mathsf{pk}', \mathsf{com}), \mathsf{sk}'$

On input: $\mathsf{ct}_0, \mathsf{ct}_1$

- Parse ct_0 as $(c_0, c_0', \alpha_0, \pi_0)$ and ct_1 as $(c_1, c_1', \alpha_1, \pi_1)$. Let $\mathsf{stmt}_0 := (c_0, c_0', \alpha_0)$, and $\mathsf{stmt}_1 := (c_1, c_1', \alpha_1)$.
- Verify that $\alpha_0 = \alpha_1$ and $\mathsf{NIZK.Verify}(\mathsf{crs}, \mathsf{stmt}_0, \pi_0) = \mathsf{NIZK.Verify}(\mathsf{crs}, \mathsf{stmt}_1, \pi_1) = 1$. If verification fails, output $\perp$.
- Compute $x_0 = \mathcal{E}.\mathsf{Dec}(\mathsf{sk}', c_0')$ and $x_1 = \mathcal{E}.\mathsf{Dec}(\mathsf{sk}', c_1')$ output $f(x_0, x_1)$.

Figure 6: Program P'

2. A *simulated* ciphertexts ct, i.e., those returned by the simulator to the adversary during the simulation. In this case, $\mathsf{NIZK.Verify}(\widetilde{\mathsf{crs}}, \mathsf{ct}, \pi) = 1$, however, $\widetilde{\alpha} \neq \alpha$ where α is committed to in the public parameters. Furthermore, if $x = \mathcal{E}.\mathsf{Dec}(\mathsf{sk}, c)$ and $y = \mathcal{E}.\mathsf{Dec}(\mathsf{sk}', c')$, then $x = x_i$ and $y = y_i$ for some $i \in [m]$, where x_i and y_i are the challenge plaintexts selected by the adversary.
3. An *honest* ciphertext ct, where $\mathsf{NIZK.Verify}(\widetilde{\mathsf{crs}}, \mathsf{ct}, \pi) = 1$. However, ct is not equal to any ciphertext returned by the simulator to the adversary. Due to the statistical simulation soundness of the NIZK and the perfect binding property of the commitment, it must hold that $\widetilde{\alpha} = \alpha$. Further, due to the statistical simulation soundness of the NIZK and the perfect correctness of the encryption scheme, if $x := \mathcal{E}.\mathsf{Dec}(\mathsf{sk}, c)$ and $x' := \mathcal{E}.\mathsf{Dec}(\mathsf{sk}', c')$, then $x = x'$.

It is not hard to see that all ciphertexts must fall into one of these three types (except with negligible probability over the choice of $\widetilde{\mathsf{crs}}$). We can now prove that Program P and Program P' are indistinguishable through a case by case analysis:

- **At least one of ct_0 and ct_1 is bad.** Both Program P and Program P' return $\perp$.
- **Honest + Honest.** If both ct_0 and ct_1 are honest ciphertext, then obviously Program P and Program P' give the same output.
- **Simulated+ Simulated.** If both ct_0 and ct_1 are simulated ciphertext, then due to the non-trivial adversary condition specified in the security definition: for every query F made to the $\mathsf{KeyGen}(\cdot)$ oracle, and for all $i, j \in [m]$, it holds that $f(x_i, x_j) = f(y_i, y_j)$. We can conclude that Program P and Program P' give the same output. Note also that all simulated ciphertexts use the same $\widetilde{\alpha}$ value, so the α value consistency check in Programs P and P' will always pass in this case.
- **Simulated + Honest.** If one of c and c' is simulated and the other is honest, the α value consistency check (i.e., the $\alpha_0 = \alpha_1$ check) will fail. Hence, both Program P and Program P' will output $\perp$.

We conclude that the programs are functionally equivalent. It is now straightforward to reduce the distinguishability of the two hybrid games to the security of the $i\mathcal{O}$ scheme. This is done in the same manner as was done in Claim B.8, and we omit the rest of the proof. ■

Remark B.12 Note that it is actually computationally infeasible for the adversary to generate any honest ciphertexts in a simulation where only simulated ciphertexts are returned to the adversary. In fact, if the adversary could generate an honest ciphertext, it would be able to guess the α value in the commitment, which breaks the computational hiding property of the commitment scheme. However, our proof must still consider honest ciphertexts, since *$i\mathcal{O}$ requires the two functions to agree on all possible inputs, including the ones that are computationally infeasible for the adversary to compute*.

Hybrid Y'. Hybrid Y' is almost the same as the real-world game with $b = 1$, where the simulator encrypts y_i in both copies of the challenge ciphertexts c_i and c'_i — the only exception is that the simulator uses sk' in the $i\mathcal{O}$ as the decryption key.

Claim B.13 *Assume that the $i\mathcal{O}$ is secure, the NIZK is computationally hiding and statistical simulation sound, the encryption scheme $\mathcal{E}$ is semantically secure and perfectly correct, and that the commitment scheme is perfectly binding and computationally hiding, then Hybrid Y' is computationally indistinguishable from Hybrid 4.*

Proof: The proof is completely symmetric to the proofs of Hybrid 1 through Hybrid 4. ■

Hybrid Y. Hybrid Y is the same as the real-world game with $b = 1$, where the simulator encrypts y_i in both copies of the challenge ciphertexts c_i and c'_i. The simulator also uses sk in the $i\mathcal{O}$ as the decryption key.

Claim B.14 *Assume that the $i\mathcal{O}$ is secure, the NIZK is statistically sound, the encryption scheme is perfectly correct, Hybrid Y' is computationally indistinguishable from Hybrid Y.*

Proof: It is not hard to see that if the NIZK is statistically sound, and that the encryption scheme is perfectly correct, then decrypting either copy will always give the same plaintext. Therefore, using either key sk or sk' to decrypt in the $i\mathcal{O}$ is functionally equivalent. Now, due to the security of the $i\mathcal{O}$, Hybrid Y' is computationally indistinguishable from Hybrid Y. ■

B.3 IND-secure Multi-Client FE from $i\mathcal{O}$

In this section we prove Theorem 2.10 as described in Section 2.3.3.

Theorem B.15 (Theorem 2.10) *Let $\mathcal{G}$ be a group for which the Diffie-Hellman assumption holds, and let H be a random oracle. If the $i\mathcal{O}$ is secure, the NIZK is statistically simulation sound, and the encryption scheme is semantically secure and perfectly correct, then the above construction is selectively, IND-secure, as defined in Section 2.3.1.*

Without loss of generality, assume that the uncorrupted set $G = \{1, 2, \ldots, |G|\}$.

Hybrid X. Same as the real-world with $b = 0$, where the challenger encrypts $\vec{x}^*_G$ in both copies c_i and c'_i of the challenge ciphertexts.

Hybrid 1. For the uncorrupted set G, pick random $\vec{\beta}_G := (\beta_1, \beta_2, \ldots, \beta_{|G|})$, such that $\langle \vec{\beta}_G, \vec{\alpha}_G \rangle = 0$. For any token on function f queried, compute and return an $i\mathcal{O}$ for the modified program P_1 as in Figure 7.

Claim B.16 *Assuming that the $i\mathcal{O}$ is secure, the NIZK scheme is statistically sound, and the encryption scheme $\mathcal{E}$ is perfectly correct. Then, Hybrid 1 is computationally indistinguishable from Hybrid X.*

Proof: By the security of $i\mathcal{O}$, it suffices to show that the modified program P_1 is input/output equivalent to the original program P. To show this, it suffices to show that the added "orthogonal check" will never fail (if program execution reached the orthogonal check). If the program execution reached the orthogonal check without returning $\perp$, then $\forall i \in [n]$: $\mathsf{NIZK.Verify}(\mathsf{crs}, \pi_i, \mathsf{stmt}_i) = 1$. Since the NIZK scheme is statistically simulation sound, write $\mathsf{ct}_i := (t, i, c_i, c'_i, d_i, \pi_i)$, it holds that

$$\begin{aligned}&\text{For } i \in [n]: \ \exists\, x, (\rho, \rho'), \omega \ \ s.t. \\ &\ \mathsf{DH}(h_t, g_i, d_i, \omega) \wedge (c_i = \mathcal{E}.\mathsf{Enc}(\mathsf{pk}, x; \rho)) \wedge \left(c'_i = \mathcal{E}.\mathsf{Enc}(\mathsf{pk}', x; \rho')\right)\end{aligned}$$

Program $P_1(\mathsf{ct}_1, \mathsf{ct}_2, \ldots, \mathsf{ct}_n)$**:**

Internal hard-coded state: $\mathsf{param}, \vec{\beta}_G, \mathsf{sk}, f$.

1. For $i \in [n]$, unpack $(t_i, j_i, c_i, c_i', d_i, \pi_i) \leftarrow \mathsf{ct}_i$. Check that $t_1 = t_2 = \ldots = t_n$, and that $j_i = i$. Let $\mathsf{stmt}_i := (t_i, j_i, c_i, c_i', d_i)$.
2. For $i \in [n]$, check that $\mathsf{NIZK.Verify}(\mathsf{crs}, \pi_i, \mathsf{stmt}_i) = 1$.
3. Orthogonal check: check that the $\{d_i\}_{i \in G}$ satisfy: $\prod_{i \in G} d_i^{\beta_i} = 1$.
4. If any of these above checks fail, output $\perp$.

   Else, for $i \in [n]$, let $x_i \leftarrow \mathcal{E}.\mathsf{Dec}(\mathsf{sk}, c_i)$. Output $f(x_1, x_2, \ldots, x_n)$.

Figure 7: MC-FE: Program P_1.

We can conclude that in step 3 of the modified program P_1, the d_i's must satisfy: $d_i = h_t^{\alpha_i}$, Because $\vec{\beta}_G$ is chosen such that $\langle \vec{\beta}_G, \vec{\alpha}_G \rangle = 0$, it must hold that $\prod_{i \in G} d_i^{\beta_i} = 1$. ■

Hybrid 2. In this game, the crs generation is simulated, and the proof π used during encryption is simulated. We note that here we are relying on the fact that our MC-FE scheme is only selectively secure, which allows the challenger to construct the statement s to simulate before constructing the simulated crs. More specifically, the game now becomes the following (with changes from the previous Hybrid 1 highlighted in red).

Let $\widetilde{K}(\cdot)$ denote the modified key generation oracle as specified by Hybrid 1, which on query f, generates a token for the modified program P_1 which includes the orthogonoal check. Note that $\widetilde{K}$ uses the simulated param which includes the simulated $\widetilde{\mathsf{crs}}$.

1. $G, \overline{G}, (\vec{x}_G^*, \vec{y}_G^*) \leftarrow \mathcal{A}$.
2. $b \xleftarrow{\$} \{0,1\}$.

   Generate random $\alpha_1, \ldots, \alpha_n \xleftarrow{\$} \mathbb{Z}_p$ and let $g_i := g^{\alpha_i}$. Run $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$, $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$,

   Let T denote the total number of time steps the adversary will query the oracle E_G.

   Guess $t^* \xleftarrow{\$} [T]$. If the guess turns out to be wrong later, abort. This only loses a polynomial factor in the security reduction.

   The simulator now precomputes the ciphertexts for the set G for the guessed challenge time step t^*, and runs NIZK.SimSetup accordingly so the proofs for the ciphertexts ct_G in t^* can be simulated:

   For $i \in G$, let $c_i := \mathcal{E}.\mathsf{Enc}(\mathsf{pk}, x_i^*; \rho_i)$ and $c_i' := \mathcal{E}.\mathsf{Enc}(\mathsf{pk}', x_i^*; \rho_i')$, for random ρ_i and ρ_i'. Let $d_i := h_{t^*}^{\alpha_i}$, where $h_{t^*} := H(t^*)$. For $i \in G$, let s_i be the statement that

$$\exists\, x, (\rho, \rho'), \omega \;\; s.t. \;\; \mathsf{DH}(h_{t^*}, g_i, d_i, \omega) \wedge (c_i = \mathcal{E}.\mathsf{Enc}(\mathsf{pk}, x; \rho)) \wedge \left(c_i' = \mathcal{E}.\mathsf{Enc}(\mathsf{pk}', x; \rho')\right)$$

   $(\widetilde{\mathsf{crs}}, \mathsf{trap}) \leftarrow \mathsf{NIZK.SimSetup}(1^\kappa, \{s_i\}_{i \in G})$.

   Let $\mathsf{param} := (\widetilde{\mathsf{crs}}, \mathsf{pk}, \mathsf{pk}', g, \{g_i\}_{i \in [n]})$. For $i \in [n]$, let $\mathsf{usk}_i := [\alpha_i, \mathsf{param}]$.

3. "challenge" $\leftarrow \mathcal{A}^{\widetilde{K}(\cdot),E_G(\cdot)}(\vec{\mathsf{usk}}_{\overline{G}})$.

4. If the current time step is not the guessed t^*, abort.
   For $i \in G$: $\pi_i \leftarrow \mathsf{NIZK.SimProve}(\widetilde{\mathsf{crs}}, s_i, \mathsf{trap})$. $\vec{\mathsf{ct}}_i^* \leftarrow (t^*, i, c_i, c_i', d_i, \pi_i)$.

5. $b' \leftarrow \mathcal{A}^{\widetilde{K}(\cdot),E_G(\cdot)}(\vec{\mathsf{ct}}_G^*)$.

Claim B.17 *Assuming that the NIZK is computational zero-knowledge, then Hybrid 2 is computationally indistinguishable from Hybrid 1.*

Proof: By a trivial reduction. ■

Hybrid 3.

1. $G, \overline{G}, (\vec{x}_G^*, \vec{y}_G^*) \leftarrow \mathcal{A}$.

2. $b \xleftarrow{\$} \{0,1\}$.

   $\alpha_1, \ldots, \alpha_n \xleftarrow{\$} \mathbb{Z}_p$. Run $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$, $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$,

   Let T denote the total number of time steps the adversary will query the oracle E_G.

   Guess $t^* \xleftarrow{\$} [T]$. If the guess turns out to be wrong later, abort. This only loses a polynomial factor in the security reduction.

   For $i \in G$, let $c_i := \mathcal{E}.\mathsf{Enc}(\mathsf{pk}, x_i^*; \rho_i)$, and $c_i' := \mathcal{E}.\mathsf{Enc}(\mathsf{pk}', x_i^*; \rho_i')$ for random ρ_i and ρ_i'. Pick random $\mu \xleftarrow{\$} \mathbb{Z}_p$. Let $d_i := h_{t^*}^{\mu\alpha_i}$ where $h_{t^*} := H(t^*)$. Let s_i be the *false* statement that

$$\exists\, x, (\rho, \rho'), \omega \;\; s.t. \;\; \mathsf{DH}(h_{t^*}, g_i, d_i, \omega) \wedge (c_i = \mathcal{E}.\mathsf{Enc}(\mathsf{pk}, x; \rho)) \wedge \left(c_i' = \mathcal{E}.\mathsf{Enc}(\mathsf{pk}', x; \rho')\right)$$

   Compute $(\widetilde{\mathsf{crs}}, \mathsf{trap}) \leftarrow \mathsf{NIZK.SimSetup}(1^\kappa, \{s_i\}_{i \in G})$.

   Let $\mathsf{param} := (\widetilde{\mathsf{crs}}, \mathsf{pk}, \mathsf{pk}', g, \{g_i\}_{i \in [n]})$. For $i \in [n]$, let $\mathsf{usk}_i := [\alpha_i, \mathsf{param}]$.

3. "challenge" $\leftarrow \mathcal{A}^{\widetilde{K}(\cdot),E_G(\cdot)}(\vec{\mathsf{usk}}_{\overline{G}})$.

4. If the current time step is not the guessed t^*, abort.
   For $i \in G$: $\pi_i \leftarrow \mathsf{NIZK.SimProve}(\widetilde{\mathsf{crs}}, s_i, \mathsf{trap})$. $\vec{\mathsf{ct}}_i^* \leftarrow (t^*, i, c_i, c_i', d_i, \pi_i)$.

5. $b' \leftarrow \mathcal{A}^{\widetilde{K}(\cdot),E_G(\cdot)}(\vec{\mathsf{ct}}_G^*)$.

Claim B.18 *Assuming that Decisional Diffie-Hellman (DDH) is hard in the group $\mathcal{G}$, H is a random oracle, and that the NIZK is computationally zero-knowledge, then Hybrid 3 is computationally indistinguishable from Hybrid 2.*

Proof: We will construct a simulation as below, where a simulator obtains a DDH instance (g^A, g^B, R) from a DDH challenger, where either $R = g^{AB}$ or R is a random group element. The simulator also simulates answers to the random oracle H.

We will craft the simulation such that the following holds: In the former case where the DDH instance is a true DH tuple, the simulation will be computationally indistinguishable from Hybrid 2 if the NIZK is

computationally zero-knowledge. Otherwise, the simulation will be indistinguishable from Hybrid 3 if the NIZK is computationally zero-knowledge.

The simulator will guess t^* upfront as mentioned earlier, and simply aborts if the guess is wrong. Upon random oracle query on $H(t^*)$, the simulator will return $h_{t^*} := g^B$. For all other time steps $t \neq t^*$, on random oracle query $H(t)$, the simulator picks $h_t := g^{\eta_t}$ where η_t is picked at random and known by the simulator. When constructing param, for the honest set $i \in G$, the simulator will choose random $\nu_i \in \mathbb{Z}_p$, and compute $g_i := (g^A)^{\nu_i}$. Implicitly, the simulator sets $\alpha_i := A\nu_i$ without knowing the value of the α_i's. Note that even though the simulator does not know α_i for $i \in G$, the simulator can compute the NIZKs for all non-challenge time steps $t \neq t^*$, since the simulator knows the alternative witnesses η_t to the DH relation where $h_t := g^{\eta_t}$. Due to the computational zero-knowledge property of the NIZK, the adversary cannot tell that an alternative witness is used in constructing the NIZK. In addition, the simulator can still generate $\vec{\beta}_G$ used in the modified program P_1, by choosing random $\vec{\beta}_G$ such that $\langle \vec{\beta}_G, \vec{\nu}_G \rangle = 0$.

For the challenge time step t^*, the simulator also chooses $d_i := R^{\nu_i}$, such that if (g^A, g^B, R) is a real DH tuple, then d_i has identical distribution as in Hybrid 2 — and since the NIZK is computationally zero-knowledge, in this case the game will be computationally indistinguishable from Hybrid 2. Otherwise, if R is chosen at random, then d_i has identical distribution as in Hybrid 3 — and since the NIZK is computationally zero-knowledge, in this case the game will be computationally indistinguishable from Hybrid 3.

■

Hybrid 4. For the challenge time step t^*, the simulator will replace the second copy of the ciphertext to encryptions of y^*_G, as elaborated below:

1. $G, \overline{G}, (\vec{x}^*_G, \vec{y}^*_G) \leftarrow \mathcal{A}$.
2. $b \xleftarrow{\$} \{0,1\}$.

   $\alpha_1, \ldots, \alpha_n \xleftarrow{\$} \mathbb{Z}_p$. Run $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$, $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$,

   Let T denote the total number of time steps the adversary will query the oracle E_G.

   Guess $t^* \xleftarrow{\$} [T]$. If the guess turns out to be wrong later, abort. This only loses a polynomial factor in the security reduction.

   For $i \in G$, let $c_i := \mathcal{E}.\mathsf{Enc}(\mathsf{pk}; x^*_i; \rho_i)$ $c'_i := \mathcal{E}.\mathsf{Enc}(\mathsf{pk}'; y^*_i; \rho'_i)$ for random ρ_i and ρ'_i. Pick random $\mu \xleftarrow{\$} \mathbb{Z}_p$. Let $d_i := h_{t^*}^{\mu\alpha_i}$ where $h_{t^*} := H(t^*)$. Let s_i be the *false* statement that

$$\exists\, m, (\rho, \rho'), \omega \;\; s.t. \;\; \mathsf{DH}(h_{t^*}, g_i, d_i, \omega) \wedge (c_i = \mathcal{E}.\mathsf{Enc}(\mathsf{pk}; m; \rho)) \wedge \left(c'_i = \mathcal{E}.\mathsf{Enc}(\mathsf{pk}'; m; \rho')\right)$$

   Compute $(\widetilde{\mathsf{crs}}, \mathsf{trap}) \leftarrow \mathsf{NIZK.SimSetup}(1^\kappa, \{s_i\}_{i \in G})$.

   Let $\mathsf{param} := (g, \widetilde{\mathsf{crs}}, \{\mathsf{pk}, \mathsf{pk}', g_i\}_{i \in [n]})$. For $i \in [n]$, let $\mathsf{usk}_i := \alpha_i$.
3. "challenge" $\leftarrow \mathcal{A}^{\widetilde{K}(\cdot), E_G(\cdot)}(\mathsf{param}, \vec{\mathsf{usk}}_{\overline{G}})$.
4. If the current time step is not the guessed t^*, abort.

   For $i \in G$: $\pi_i \leftarrow \mathsf{NIZK.SimProve}(\widetilde{\mathsf{crs}}, s_i, \mathsf{trap})$. $\vec{\mathsf{ct}}^*_i \leftarrow (t^*, i, c_i, c'_i, d_i, \pi_i)$.
5. $b' \leftarrow \mathcal{A}^{\widetilde{K}(\cdot), E_G(\cdot)}(\vec{\mathsf{ct}}^*_G)$.

Program $P_2(\mathsf{ct}_1, \mathsf{ct}_2, \ldots, \mathsf{ct}_n)$**:**

Internal hard-coded state: $\mathsf{param}, \vec{\beta}_G, \mathsf{sk}', f$.

1. For $i \in [n]$, unpack $(t_i, j_i, c_i, c_i', d_i, \pi_i) \leftarrow \mathsf{ct}_i$. Check that $t_1 = t_2 = \ldots = t_n$, and that $j_i = i$. Let $\mathsf{stmt}_i := (t_i, j_i, c_i, c_i', d_i)$.
2. For $i \in [n]$, check that $\mathsf{NIZK.Verify}(\widetilde{\mathsf{crs}}, \pi_i, \mathsf{stmt}_i) = 1$.
3. Orthogonal check: check that the $\{d_i\}_{i \in G}$ satisfy: $\prod_{i \in G} d_i^{\beta_i} = 1$.
4. If any of these above checks fail, output $\perp$.

   Else, for $i \in [n]$, let $x_i \leftarrow \mathcal{E}.\mathsf{Dec}(\mathsf{sk}'; c_i')$. Output $f(x_1, x_2, \ldots, x_n)$.

Figure 8: MC-FE: Program P_2.

Claim B.19 *Assuming that the encryption scheme $\mathcal{E}$ is semantically secure, Hybrid 4 is computationally indistinguishable from Hybrid 3.*

Proof: By a trivial reduction. Observe that the KeyGen algorithm produces an $i\mathcal{O}$ that embeds only the first (unprimed) copy of secret key, i.e., sk. ■

Hybrid 5. The simulator will now use sk' instead of the sk in the $i\mathcal{O}$, as shown in Figure 8:

Claim B.20 *Assuming that the $i\mathcal{O}$ is secure, and that the NIZK is statistically simulation sound, and that the encryption scheme $\mathcal{E}$ is perfectly correct, then Hybrid 5 is computationally indistinguishable from Hybrid 4.*

Proof: By the security of the $i\mathcal{O}$, it suffices to prove that except with some negligible probability of some bad event happening, the program P_2 in Hybrid 5 is input-output equivalent (on *all* inputs) to the program P_1 used in Hybrids 1 to 4. *We stress that this is different from two programs that agree on most inputs, but differ on a negligible fraction of inputs.* In our case, except with negligible probability over parameters chosen by the simulator, the two programs P_2 and P_1 agree on *all* inputs — that is why we can apply $i\mathcal{O}$ to switch the programs.

If the inputs to the programs P_2 or P_1 do not contain any ciphertexts from the challenge time step t^*, then by the statistical simulation soundness of the NIZK, if all NIZKs on the ciphertexts verify, then all ciphertexts must be truthfully formed. This means that the two copies of the encryptions c_i and c_i' must encrypt the same plaintext. Further, due to the perfect correctness of the encryption scheme $\mathcal{E}$, it is not hard to see that the programs P_2 and P_1 are input-output equivalent under this case.

Now, consider the case when the inputs to the programs P_2 or P_1 contain the challenge time step t^*. Due to the checks performed in Steps 1 and 2 of both programs P_2 and P_1, for either program not to return $\perp$: 1) all input ciphertexts must correspond to the same time step t^*; further, must be tagged with indices $1, 2, 3, \ldots, n$ in this order; and 2) the NIZKs must all verify on all ciphertexts.

Assume that this is indeed the case and the Steps 1 and 2 pass the checks. Now, consider the ciphertexts $\vec{\mathsf{ct}}_G$ corresponding to the good parties. There are the following cases:

- **All simulated.** Suppose all $\vec{\mathsf{ct}}_G$ input to the programs are generated by the the simulator when answering the challenge phase.

  By perfect security of the encryption scheme, in the programs, the decrypted ciphertext in Step 5 of either program, the decrypted values must be the same as the challenge plaintexts selected by the

adversary, denoted $\vec{x}^*_G$ and $\vec{y}^*_G$. Further, by definition of the security game, $f(\vec{x}^*_G, \cdot) = f(\vec{y}^*_G, \cdot)$. Now, due to the statistical simulation soundness of the NIZK, if the NIZKs verify, then the ciphertexts $\vec{\mathsf{ct}}_{\overline{G}}$ corresponding to the compromised set must encrypt consistent plaintexts in the two copies c_i and c'_i. Due to the perfect correctness of the encryption scheme $\mathcal{E}$, it follows that decrypting either copy c_i or c'_i will result in the same plaintexts for the $\overline{G}$ parties.

Now, due to the fact that $f(\vec{x}^*_G, \cdot) = f(\vec{y}^*_G, \cdot)$, it is not hard to see that P_1 and P_2 are input-output equivalent under this case.

- **All honest.** Suppose all $\vec{\mathsf{ct}}_G$ ciphertexts input to the programs are honestly generated ciphertexts by following the Enc algorithm of the MC-FE scheme.

  Due to the statistical simulation soundness of the NIZK scheme, all ciphertexts ct_i must encrypt two consistent plaintext for the c_i copy and the c'_i copy. Due to the perfect correctness of the encryption scheme $\mathcal{E}$, the decrypted values must be those that are encrypted. Therefore, regardless of which decryption key sk or sk' is used, the decrypted values will be the same. We can conclude that P_1 and P_2 are input-output equivalent in this case.

  Remark. One interesting is the following: when interacting with the simulation, it is actually not computationally feasible for the adversary to generate any honest ciphertexts for the good parties in G on its own (except with negligible probability) — otherwise, the adversary must be able to compute an honestly formed d_i, and it would then be able to break Computational Diffie-Hellman. The reason why we still need to cover this case is that the security of $i\mathcal{O}$ requires the two programs to be input-output equivalent on all possible inputs, regardless of whether it might be computationally feasible for the adversary to generate those inputs.

- **Mixing honest and simulated.** The most interesting case is when a subset of the input ciphertexts in ct_G are honestly generated by following the Enc algorithm of the MC-FE scheme, while the remaining subset is simulated, i.e., generated by the simulator when answering the adversary in the challenge phase. This is when the added orthogonal check will be useful.

  We argue that under this case, except with negligible probability over choices made by the simulator, the orthogonal check will fail on *all possible inputs that mix honest and simulated ciphertexts* (assuming that they pass the checks in Step 1 and Step 2), and hence both programs will return $\perp$. The reason is as follows: for a honestly generated ciphertext ct_i, due to the statistical simulation soundness of the NIZK, the d_i terms in the ciphertext ct_i must satisfy $d_i := h_{t^*}^{\alpha_i}$. However, for a simulated ciphertext ct_i, $d_i := h_{t^*}^{\mu\alpha_i}$, for the random $\mu \in \mathbb{Z}_p$ chosen by the simulator. By construction, $\langle \vec{\beta}_G, \vec{\alpha}_G \rangle = 0$. Now that a subset of G is scaled by a random $\mu \in \mathbb{Z}_p$, it is not hard to see that for a fixed $\Gamma \cup \overline{\Gamma} = G$, where Γ represents the honestly generated set of ciphertexts, and $\overline{\Gamma}$ represents the simulated set: the orthogonal check will fail except with $\frac{1}{p}$ probability, where the probability is taken over the choice of $\vec{\beta}_G$, $\vec{\alpha}_G$, and μ.

  By a union bound, the probability that there exists some non-empty proper subset $\Gamma \subset G$ such that the orthogonal check fails is $\frac{2^{|G|}}{p}$. For this to be a negligible failure probability, we just have to set $p > 2^n \cdot 2^\kappa$.

Hybrid Y'. Almost the same as a real-world simulation, where the challenger encrypts $\vec{y}^*_G$ in both copies of the challenge ciphertexts. The only exception is that the challenger uses sk' instead of sk for constructing the $i\mathcal{O}$ in the token queries.

Claim B.21 *Assuming that the $i\mathcal{O}$ scheme is secure, the NIZK is statistically simulation sound and computationally zero-knowledge, the encryption scheme $\mathcal{E}$ is semantically secure and perfectly correct, and that*

H is a random oracle, then Hybrid Y' is computationally indistinguishable from Hybrid 5.

Proof: By a symmetric argument as in Hybrid X through Hybrid 4, we can conclude that Hybrid Y' is computationally indistinguishable from Hybrid 5. ■

Hybrid Y. Same as a real-world simulation, where the challenger encrypts $\vec{y}^*_G$ in both copies of the challenge ciphertexts. Further, the challenger uses sk for constructing the $i\mathcal{O}$ in the token queries.

Claim B.22 *Assuming that the $i\mathcal{O}$ is secure, that the NIZK is statistically sound, and that the encryption scheme is perfectly correct, Hybrid Y' and Hybrid Y are computationally indistinguishable.*

Proof: By the security of $i\mathcal{O}$, it suffices to show that using sk or sk' in the $i\mathcal{O}$ program result in exactly the same input-output behavior. It is not hard to see that if the NIZK is statistically sound and the encryption scheme is perfectly correct, this must be the case. ■

■

Removing random oracle. As mentioned earlier, the random oracle can be removed by embedding the terms h_t's in the public parameters. The proof would basically remain the same, except that now for the indistinguishability between Hybrids 2 and 3, the simulator embeds the DDH instance obtained from a DDH challenger directly in the public parameters, instead of in answering a random oracle query.

B.4 IND-secure Multi-Client FE from $\mathsf{di}\mathcal{O}$

In this section we prove Theorem 2.11 in Section 2.3.4. To proof the security, we will show Hybrid X and Hybrid Y (defined in the following) are indistinguishable by a series of hybrids. Given any $G, \bar{G}$, we define the following experiments.

Hybrid X: This game is simply the defined above with $b = 0$. In other words, the challenger will encrypt $\vec{x}^*_G$ (in both copies of the ciphertexts c and c') and return them to the adversary.

Hybrid 1: In this hybrid, when the challenger answers an adversary's query, the crs generation is simulated, allowing the simulator to obtain a simulated $\widetilde{\mathsf{crs}}$ and a trapdoor trap for simulating fake proofs. Further, the NIZK proofs for the challenge time-step will now be simulated. For the non-challenge time steps, the proofs are generated honestly.

Hybrid 2: In this hybrid, instead of using the true $\{h_{t^*}^{\alpha_i}\}_i$ in the challenge time step, the simulator will pick a random μ, and use $h_{t^*}^{\mu\alpha_i}$ for $i \in G$. This has the effect of scaling the $\vec{\alpha}_G$ vector by a constant without changing its angle. The NIZK proofs for the honest set G for the challenge time steps are simulated.

Hybrid 3: In the challenge time step, we will switch the challenge ciphertexts to $\mathcal{E}.\mathsf{Enc}(\mathsf{pk}, x_i; \rho), \mathcal{E}.\mathsf{Enc}(\mathsf{pk}, y_i; \rho')$ for all $i \in G$, where t is the time step of the challenge. The NIZK proofs for the honest set G for the challenge ciphertexts are simulated.

Hybrid 4: In this hybrid, we will switch the obfuscated program to $\mathsf{di}\mathcal{O}(P')$, where P' is similar to P except it is hardwired sk', and it decrypts the second copies of the ciphertexts and computes accordingly.

Hybrid Y': In this hybrid, the challenger encrypts $\vec{y}_G$ in both copies of the challenge ciphertexts. $\mathsf{di}\mathcal{O}(P')$ is given to the adversary which uses sk' to decrypt.

Hybrid Y: This game is simply the defined above with $b = 1$. In other words, the challenger will encrypt $\vec{y}^*_G$ (in both copies of the ciphertexts c and c') and return them to the adversary. $\mathsf{di}\mathcal{O}(P)$ is given to the adversary which uses sk to decrypt.

Then we show the indistinguishability of the hybrids by the following claims:

Claim B.23 *If the NIZK is computationally zero knowledge, then Hybrid X and Hybrid 1 are computationally indistinguishable.*

Proof: By a trivial reduction. ■

Claim B.24 *If the DDH assumption holds in the group $\mathcal{G}$, H is a random oracle*[3]*, and that the NIZK is computationally zero-knowledge, then Hybrid 1 and Hybrid 2 are computationally indistinguishable.*

Proof: The proof follows in a similar way as the proof of Claim B.18. ■

Claim B.25 *If the encryption scheme E is semantically secure, Hybrid 2 and Hybrid 3 are computationally indistinguishable.*

Proof: By a trivial reduction. ■

Claim B.26 *If the security of the obfuscation $\mathsf{di}\mathcal{O}$ and the simulation soundness of the NIZK hold, then Hybrid 3 and Hybrid 4 are computationally indistinguishable.*

Define $\mathsf{Sampler}$ to be the following algorithm: First, run the Setup algorithm of the MC-FE scheme, with the following exception: given t^* and the challenge plaintexts, $\mathsf{Sampler}$ runs the simulated (instead of real) setup algorithm of the simulation-sound NIZK[4]. $\mathsf{Sampler}$ then includes the msk, all the $\{\mathsf{usk}_i\}$, as well as the simulated NIZKs for the challenge time step in aux. $\mathsf{Sampler}$ also outputs the programs P and P'.

Assuming that the NIZK is simulation sound, it is not hard to see that no polynomial-time adversary, when given aux, can find an input where P and P' differ – since to cause P and P' to output different answers, the adversary must have forged a NIZK for a false statement different from the fake NIZKs of the challenge time step.

Now suppose $\mathcal{B}$ is given aux and one of $\mathsf{di}\mathcal{O}(P)$ or $\mathsf{di}\mathcal{O}(P')$. $\mathcal{B}$ can interact with the adversary $\mathcal{A}$ who is trying to distinguish Hybrid 3 and Hybrid 4. It is not hard to see that $\mathcal{B}$ can simulate answers to all the queries asked by the adversary $\mathcal{A}$. Further, if $\mathcal{A}$ can distinguish which world it is in with non-negligible probability, then $\mathcal{B}$ has a non-negligible advantage in distinguishing $\mathsf{di}\mathcal{O}(P)$ and $\mathsf{di}\mathcal{O}(P')$. This contradicts the assumption that $\mathsf{di}\mathcal{O}$ is secure.

Claim B.27 *Assuming that NIZK is computationally zero-knowledge and simulation sound, the DDH assumption holds in the group $\mathcal{G}$, H is a random oracle, the encryption scheme is semantically secure, and that $\mathsf{di}\mathcal{O}$ is secure, then Hybrid 4 and Hybrid Y' are computationally indistinguishable.*

Proof: The proof is completely symmetric to the proof why Hybrid X through Hybrid 4 are computationally indistinguishable. ■

[3] As mentioned earlier, the random oracle assumption can be removed by choosing the terms $\{h_t\}_t$ during setup and including them in each user's secret key.

[4] Note that this requires selective security

Claim B.28 *Assuming that* $\mathsf{di}\mathcal{O}$ *is secure, and that NIZK is computationally sound, and that the encryption scheme is perfectly correct, then Hybrid Y is computationally indistiguishable from Hybrid Y'.*

Proof: Consider a $\mathsf{Sampler}$ algorithm that runs the real setup algorithm of the MC-FE scheme, and outputs msk, and all $\{\mathsf{usk}_i\}_i$ as aux. Further, $\mathsf{Sampler}$ outputs two programs P and P' whose only difference is whether sk or sk' is used in decryption.

It is not hard to see that no polynimial-time adversary can find an input that cause P and P' to give different outputs, assuming that the NIZK is computationally sound and that the encryption scheme is perfectly correct. Since to cause the two programs to give different outputs, it is necessary for an adversary to forge a proof (under a real CRS) for a false statement.

Now, by the security of $\mathsf{di}\mathcal{O}$, Hybrid Y and Hybrid Y' are computationally indistinguishable. ■

C SIM-Security: Proofs

Theorem C.1 (Theorem 3.3) *Let* FE *be a binary functional encryption scheme for a class of binary functions* $\mathcal{F} = \{\mathcal{F}_\kappa\}$*. Assume* FE *is either* $(1,1,0)$ *or* $(0,1,1)$*-simulation secure, then the class* $\mathcal{F}$ *is learnable.*

Proof: We consider the case of $(1,1,0)$-simulation secure, and remark that the other case is similar. Our goal is to construct a learner L_1 for $f(x,\cdot)$, for every κ, every function $f \in \mathcal{F}_\kappa$, every input x, and similarly for another learner L_2. By symmetry, we only describe the construction and analysis of L_1. The same arguments carry for L_2.

By the security of the binary FE scheme, we know that there exists a simulator $\mathcal{S}$ such that for any adversary, the real experiment is indistinguishable from the ideal one. Now we define a (partial) selective adversary $\mathcal{A}_1$ parameterized by κ, f, x where $f \in \mathcal{F}_\kappa$ as follows: In the first place, $\mathcal{A}_1$ first asks for challenge message $(x, 0)$ and the key query $f(\cdot,\cdot)$. Then he will receive a token sk_f. Then $\mathcal{A}_1$ outputs the token as its state.

Then we define a learner L_1 on input 1^κ and a function f runs the following procedure that simulates the ideal experiment.

- $\mathsf{param} \leftarrow \mathcal{S}(1^\kappa)$.
- $st := \tilde{\mathsf{sk}}_f \leftarrow \mathcal{S}^{O'_{x,0}(\cdot)}(1^\kappa)$. (Recall that in the selective security, the simulator gets oracle access to $O'_{x,0}$ here.)
- Run $\mathcal{S}^{O'_{x,0}(\cdot)}(1^{|x|}, 1^{|0|})$. At the end, the $\mathcal{S}$ outputs simulated ciphertexts $\tilde{\mathsf{ct}}_x$ and $\tilde{\mathsf{ct}}_0$.
- Finally, the learner outputs $(\tilde{\mathsf{param}}, \tilde{\mathsf{sk}}_f, \tilde{\mathsf{ct}}_x)$ as a representation of the function $f(x,\cdot)$. More formally, define $f'_{(\tilde{\mathsf{param}}, \tilde{\mathsf{sk}}_f, \tilde{\mathsf{ct}}_x)}(y)$ as the following procedure:

  1. Sample $\mathsf{ct}_y \leftarrow \mathsf{FE.Enc}(\mathsf{param}, y)$.
  2. Output $\mathsf{FE.Eval}(\tilde{\mathsf{param}}, \tilde{\mathsf{sk}}_f, \tilde{\mathsf{ct}}_x, \mathsf{ct}_y)$.

We note that the oracle $O'_{x,0}(\cdot)$ can be perfectly simulated via oracle access to $f(x,\cdot)$. (Obviously, L_1 can simulate $f(\cdot, 0)$.)

Then we show that L_1 is a good learner by the following claim:

Claim C.2 *For every input* x*, and every polynomial sized distinguisher* D*, according to the following experiment, we have*

$$|\Pr[D^{f'(\cdot)}(1^\kappa) = 1] - \Pr[D^{f(x,\cdot)}(1^\kappa) = 1]| < \mathsf{negl}(\kappa),$$

for some negligible function $\mathsf{negl}(\cdot)$.

We show the claim by contradiction. Suppose there exist an input x, a polynomial sized distinguisher D and a non-negligible ϵ, such that $\Pr[D^{f'(\cdot)}(1^\kappa) = 1] - \Pr[D^{f(x,\cdot)}(1^\kappa) = 1]| > \epsilon$, then we are going to show that there exists an adversary $\mathcal{A}'$ that breaks the binary functional encryption scheme FE (with respect to the simulator $\mathcal{S}$ from the premise that FE is a secure scheme).

We define define $\mathcal{A}' = (\mathcal{A}'_1, \mathcal{A}'_2)$ as the following: $\mathcal{A}'_1$ follows exactly the same as $\mathcal{A}_1$. $\mathcal{A}'_2$ on inputs $\tilde{\mathsf{param}}, \tilde{\mathsf{ct}}_x, \tilde{\mathsf{ct}}_0, \tilde{\mathsf{sk}}_f$ simulates an oracle $g(\cdot)$ that on input y samples $\mathsf{ct}_y \leftarrow \mathsf{FE.Enc}(\tilde{\mathsf{param}}, y)$ and outputs $\mathsf{FE.Eval}(\tilde{\mathsf{param}}, \tilde{\mathsf{sk}}_f, \tilde{\mathsf{ct}}_x, \mathsf{ct}_y)$. Then $\mathcal{A}'_2$ outputs $D^{g(\cdot)}(1^\kappa)$. It is not hard to see that if $\tilde{\mathsf{param}}, \tilde{\mathsf{sk}}_f, \tilde{\mathsf{ct}}_x, \mathsf{ct}_y$, come from the real experiment, then $g(\dot{})$ behaves statistically close to $f(x, \cdot)$; if they are from the simulator, then $g(\cdot)$ behaves according to $f'(\cdot)$. Thus, the real experiment $\mathbf{Real}_{\mathcal{A}',1}(1^\kappa)$ outputs $(x, 0, 1)$ with probability $\Pr[D^{f(x,\cdot)}(1^\kappa) = 1]$, and the ideal experiment $\mathbf{Ideal}_{\mathcal{A}',\mathcal{S},1}(1^\kappa)$ outputs $(x, 0, 1)$ with probability $\Pr[D^{f'(\cdot)}(1^\kappa) = 1]$. Thus, they are distinguishable, and this completes the proof.

■

Theorem C.3 (Theorem 3.4) *Assume $v\mathcal{O}$ is a VBB obfuscator for general functions, E is a CCA secure encryption scheme, E' is semantically secure, Σ is a strongly unforgeable signature scheme. Then for any polynomial q_1, ℓ, q_2, the above public-key binary FE scheme is (q_1, ℓ, q_2)-simulation secure.*

Proof: To prove the theorem, we need to construct a simulator $\mathcal{S}$ such that for all adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, the real experiment is indistinguishable from the ideal one. Consider the following simulation algorithms:

- $\mathcal{S}(1^\kappa)$ generates a public parameter param and a secret key msk as the following: sample keys $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{E.KeyGen}(1^\kappa)$, $(\mathsf{ek}', \mathsf{dk}') \leftarrow \mathsf{E}'.\mathsf{KeyGen}(1^\kappa)$, and $(\mathsf{vk}, \mathsf{sk}) \leftarrow \Sigma.\mathsf{KeyGen}(1^\kappa)$. Then sample a random string $\tau \leftarrow \{0,1\}^\kappa$, and $\hat{D}'_{\mathsf{dk},\mathsf{dk}',\mathsf{vk},\tau} \leftarrow v\mathcal{O}(D'_{\mathsf{dk},\kappa',\mathsf{vk},\tau})$, where $D'_{\mathsf{dk},\mathsf{vk},\tau}$ is a circuit that does the following: on input ciphertexts $\mathsf{ct}_x, \mathsf{ct}_y, \mathsf{ct}_{\bar{f}}$ and a signature σ,
  - If $\Sigma.\mathsf{Ver}(\mathsf{vk}, \mathsf{ct}_{\bar{f}}, \sigma) = 0$, output $\perp$.
  - Compute $f||i||(L_x, L_y, Q) = \mathsf{E}'.\mathsf{Dec}(\mathsf{dk}', \mathsf{ct}_{\bar{f}})$, where L_x, L_y are two lists of size ℓ, and each entry is of the form (ct', f') for some ciphertext ct' (of the scheme E) and some partial function f' (of length at most $p(\kappa)$); Q is an $\ell \times \ell$ matrix where each entry is an μ'-bit string.
  - Compute $(x, \mathtt{tag}_x, Z_x, f_x, b_x) = \mathsf{E.Dec}(\mathsf{dk}, \mathsf{ct}_x)$ and $(y, t_y, Z_y, f_y, b_y) = \mathsf{E.Dec}(\mathsf{dk}, \mathsf{ct}_y)$, where $\mathtt{tag}_x, \mathtt{tag}_y \in \{\text{“1”}, \text{“2”}\}$, $Z_x, Z_y \in \{0,1\}^{\mu' \cdot q_1 \cdot \ell}$, $f_x, f_y \in \{0,1\}^p$, $b_x, b_y \in \{0,1\}^\kappa$.
  - If $(\mathtt{tag}_x, \mathtt{tag}_y) \neq$ (“1”,“2”), output $\perp$. Otherwise we consider the following cases:
  - If $(b_x, b_y) = (\tau, \tau)$, then check if (L_x, L_y, Q) is the all zero string.
    - if no, then $Q[j][k]$ where j, k are the indices of the positions $\mathsf{ct}_x, \mathsf{ct}_y$ in L_x, L_y.
    - otherwise, interpret Z_x and Z_y as two $q_1 \times \ell$ matrices, where each entry is an μ'-bit string. Check whether $z_x[i][y] = z_y[i][x]$. If not output $\perp$, otherwise $z_x[i][y]$.
  - If $(b_x, b_y) = (\tau, \neq \tau)$, then check if (L_x, L_y, Q) is the all zero string.
    - if no, then output $f'(y)$, where the j-th entry of L_x is (ct_x, f').
    - otherwise, interpret f_x as a function and output $f_x(y)$.
  - If $(b_x, b_y) = (\neq \tau, \tau)$, then check then check if (L_x, L_y, Q) is the all zero string.
    - if no, then output $f'(x)$, where the j-th entry of L_y is (ct_y, f').
    - otherwise, interpret f_y as a function and output $f_y(x)$.
  - If none of the above is true, output $f(x, y)$.

  Finally, set $\mathsf{param} := (\mathsf{ek}, \mathsf{ek}', \mathsf{vk}, \hat{D}_{\mathsf{dk},\mathsf{dk}',\mathsf{vk},\tau})$, $\mathsf{msk} := (\mathsf{dk}, \mathsf{sk})$.

- Upon a pre-challenge key query f from the adversary, $\mathcal{S}$ generates the secret key in the following way. Let f be the i-th function that the adversary queries. (Since the simulator is stateful, he can record this information). The simulator computes $\mathsf{ct}_{\bar{f}} \leftarrow \mathsf{E}'.\mathsf{Enc}(\mathsf{ek}', f||i||0^{2\ell\cdot(K+p)+\ell^2\cdot\mu'})$. Then generate a signature $\sigma \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}, \mathsf{ct}_{\bar{f}})$. Output $\mathsf{sk}_f := (\mathsf{ct}_{\bar{f}}, \sigma)$.

- Now $\mathcal{S}$, given oracle access to $O'_{x_1,y_1,\ldots,x_\ell,y_\ell}(\cdot)$, generates ciphertexts $(\mathsf{ct}_{x_1}, \mathsf{ct}_{y_1}), \ldots, (\mathsf{ct}_{x_\ell}, \mathsf{ct}_{y_\ell})$ in the following way:

  - By the learnability of $f_{x_i}^{(j)}(\cdot), f_{y_i}^{(j)}(\cdot)$ (for all $i \in [\ell], j \in [q_1]$), the simulator uses the learners and oracle access to $O'()$, to find descriptions of the functions: say $f_{x_i}^{(j)}, f_{y_i}^{(j)}$ for $i \in [\ell], j \in [q_1]$. We note that their lengths can be bounded by $p(\kappa)$. (Recall that we assume the learners' running time is bounded by $p(\kappa)$). The simulator is clearly admissible since he only needs to query the oracle with the functions that the adversary queried in the previous step.
  - Then for all $i \in [\ell]$, $\mathcal{S}$ sets z_{x_i} as a $q_1 \times \ell$ array where the entry $z_{x_i}[j][k] = f^{(j)}(x_i, y_k)$ for $j \in [q_1]$ and $k \in [\ell]$; similarly, he sets z_{y_i} as a $q_1 \times \ell$ array where the entry $z_{y_i}[j][k] = f^{(j)}(x_k, y_i)$. This can be done by querying the oracle with input $(f^{(j)}, i, k)$ using the second mode, or $(f^{(j)}, k, i)$ using the third mode.
  - Finally $\mathcal{S}$ outputs $\mathsf{ct}_{x_i} = \mathsf{E}.\mathsf{Enc}(\mathsf{pk}, (i, \text{“1”}, z_{x_i}, f_{x_i}, \tau))$, and $\mathsf{ct}_{y_i} = \mathsf{E}.\mathsf{Enc}(\mathsf{pk}, (i, \text{“2”}, z_{y_i}, f_{y_i}, \tau))$ for all $i \in [\ell]$.

- On a post-challenge key query f, $\mathcal{S}$ first learns $f_{x_k}(\cdot), f_{y_k}(\cdot)$ for $k \in [\ell]$ via oracle queries to O'. Denote these as strings $((f_{x_1}, f_{y_1}), \ldots, (f_{x_\ell}, f_{y_\ell}))$. $\mathcal{S}$ sets L_x as a list $\{(\mathsf{ct}_{x_1}, f_{x_1}), \ldots, (\mathsf{ct}_{x_\ell}, f_{x_\ell})\}$, and similarly $L_y = \{(\mathsf{ct}_{y_1}, f_{y_1}), \ldots, (\mathsf{ct}_{y_\ell}, f_{y_\ell})\}$. Recall the ciphertexts were generated in the previous step by the simulator. Then $\mathcal{S}$ sets Q to be an $\ell \times \ell$ matrix where $Q[i][j] = f(x_i, y_j)$. This can be computed via oracle access to O'.

  Finally, $\mathcal{S}$ computes $\mathsf{ct}_{\bar{f}} = \mathsf{E}'.\mathsf{Enc}(\mathsf{ek}', f||0^\kappa||(L_x, L_y, Q))$, and a signature $\sigma \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}, \mathsf{ct}_{\bar{f}})$. Then he outputs $\mathsf{sk}_f := (\mathsf{ct}_{\bar{f}}, \sigma)$.

Next, we want to show that the simulation produces an indistinguishable ideal view. We will show a stronger statement in the following claim, which implies that the simulator produces indistinguishable views for all adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. This would complete the proof of the theorem.

Claim C.4 *For any functions $f_1, \ldots, f_{q_1} \in \mathcal{F}_\kappa$ as pre-challenge queries, $g_1, g_2, \ldots, g_{q_2} \in \mathcal{F}_\kappa$ as post-challenge queries, and any messages $(x_1, y_1), \cdots, (x_\ell, y_\ell)$, the distributions are indistinguishable:*

$$\mathsf{Dist}_{\textbf{\textit{Real}}} = (\mathsf{param}, \mathsf{sk}_{f_1}, \ldots, \mathsf{sk}_{f_{q_1}}, (\mathsf{ct}_{x_1}, \mathsf{ct}_{y_1}), \ldots, (\mathsf{ct}_{x_\ell}, \mathsf{ct}_{y_\ell}), \mathsf{sk}'_{g_1}, \ldots, \mathsf{sk}'_{g_{q_2}}) \approx_c$$

$$\mathsf{Dist}_{\textbf{\textit{Ideal}}} = (\tilde{\mathsf{param}}, \tilde{\mathsf{sk}}_{f_1}, \cdots, \tilde{\mathsf{sk}}_{f_{q_1}}, (\tilde{\mathsf{ct}}_{x_1}, \tilde{\mathsf{ct}}_{y_1}), \ldots, (\tilde{\mathsf{ct}}_{x_\ell}, \tilde{\mathsf{ct}}_{y_\ell}), \tilde{\mathsf{sk}}'_{g_1}, \ldots, \tilde{\mathsf{sk}}'_{g_{q_2}}),$$

where tilde denotes the strings generated by the simulator.

To show the claim, we consider the following experiments. Let $\mathcal{B}$ be any distinguisher, and $\mathsf{Dist}_{\textbf{Real}_1}$ is identical to $\mathsf{Dist}_{\textbf{Real}}$ except it does not contain the obfuscated circuit. (Instead we will give the distinguisher oracle access to the circuit). Similar we define $\mathsf{Dist}_{\textbf{Ideal}_1}$:

$$\mathsf{Dist}_{\textbf{Real}_1} = (\mathsf{ek}, \mathsf{ek}', \mathsf{vk}, \mathsf{sk}_{f_1}, \ldots, \mathsf{sk}_{f_{q_1}}, (\mathsf{ct}_{x_1}, \mathsf{ct}_{y_1}), \ldots, (\mathsf{ct}_{x_\ell}, \mathsf{ct}_{y_\ell}), \mathsf{sk}'_{g_1}, \ldots, \mathsf{sk}'_{g_{q_2}}),$$

$$\mathsf{Dist}_{\textbf{Ideal}_1} = (\mathsf{ek}, \mathsf{ek}', \mathsf{vk}, \tilde{\mathsf{sk}}_{f_1}, \cdots, \tilde{\mathsf{sk}}_{f_{q_1}}, (\tilde{\mathsf{ct}}_{x_1}, \tilde{\mathsf{ct}}_{y_1}), \ldots, (\tilde{\mathsf{ct}}_{x_\ell}, \tilde{\mathsf{ct}}_{y_\ell}), \tilde{\mathsf{sk}}'_{g_1}, \ldots, \tilde{\mathsf{sk}}'_{g_{q_2}}),$$

Claim C.5 *Assume the security of the VBB obfuscator. Then for any functions in $\mathcal{F}_\kappa$ as key queries and messages as the challenge, suppose there exists a poly-sized distinguisher $\mathcal{B}$ and some non-negligible ϵ such that* $\Pr[\mathcal{B}(\mathsf{Dist}_{\boldsymbol{Real}}) = 1] - \Pr[\mathcal{B}(\mathsf{Dist}_{\boldsymbol{Ideal}}) = 1] > \epsilon$*, then there exists another poly-sized distinguisher $\mathcal{B}'$ such that* $\Pr[\mathcal{B}'^{D_{\mathsf{dk},\mathsf{dk}',\mathsf{vk},\tau}}(\mathsf{Dist}_{\boldsymbol{Real}_1}) = 1] - \Pr[\mathcal{B}'^{D_{\mathsf{dk},\mathsf{dk}',\mathsf{vk},\tau}}(\mathsf{Dist}_{\boldsymbol{Ideal}_1}) = 1] > \epsilon - \mathsf{negl}(\kappa)$.

We show that there exists another distinguisher $\mathcal{B}'$ who gets oracle access to the functionalities $D_{\mathsf{dk},\mathsf{dk}',\mathsf{vk},\tau}$ can still distinguish the modified real from the modified ideal. Intuitively, this follows from the VBB property.

More precisely, define $\mathcal{B}_{\mathsf{ek},\mathsf{ek}',\mathsf{vk},\vec{\mathsf{sk}},\vec{\mathsf{ct}}}$ be an adversary (in the VBB security sense), who is parameterized by $\mathsf{ek}, \mathsf{ek}', \mathsf{vk}, \vec{\mathsf{sk}}, \vec{\mathsf{ct}}$ for some fixed keys and ciphertexts. For any such fixed parameters, $\mathcal{B}_{\mathsf{ek},\mathsf{ek}',\mathsf{vk},\vec{\mathsf{sk}},\vec{\mathsf{ct}}}$ on input an obfuscated circuit $\hat{C}$ runs $\mathcal{B}(\mathsf{ek}, \mathsf{ek}', \mathsf{vk}, \vec{\mathsf{sk}}, \vec{\mathsf{ct}}, \hat{C})$. Then by the VBB property, there exists a simulator $\mathsf{Sim}_{\mathcal{B},\mathsf{ek},\mathsf{ek}',\mathsf{vk},\vec{\mathsf{sk}},\vec{\mathsf{ct}}}$ who gets oracle access to C such that the probabilities they output 1 are negligibly close.

Then we look at the oracle $C = D_{\mathsf{dk},\mathsf{dk}',\mathsf{vk}}$ in the real world. It is not hard to see:

$$\Pr[\mathcal{B}(\mathsf{Dist}_{\mathbf{Real}})] = \Pr\left[\mathcal{B}_{\mathsf{ek},\mathsf{ek}',\mathsf{vk},\vec{\mathsf{sk}},\vec{\mathsf{ct}}}(\hat{C}) = 1\right] \approx \Pr\left[\mathsf{Sim}^{C(\cdot)}_{\mathcal{B},\mathsf{ek},\mathsf{ek}',\mathsf{vk},\vec{\mathsf{sk}},\vec{\mathsf{ct}}}(1^\kappa) = 1\right]$$

Then we consider the experiment that $\mathsf{Sim}_{\mathcal{B},\mathsf{ek},\mathsf{ek}',\mathsf{vk},\vec{\mathsf{sk}},\vec{\mathsf{ct}}}$ gets oracle access to $C' = D'_{\mathsf{dk},\mathsf{dk}',\mathsf{vk},\tau}$ for a randomly chosen τ. It it not hard to see:

$$\Pr\left[\mathsf{Sim}^{C(\cdot)}_{\mathcal{B},\mathsf{ek},\mathsf{ek}',\mathsf{vk},\vec{\mathsf{sk}},\vec{\mathsf{ct}}}(1^\kappa) = 1\right] \approx \Pr_{\tau\leftarrow\{0,1\}^\kappa}\left[\mathsf{Sim}^{C'(\cdot)}_{\mathcal{B},\mathsf{ek},\mathsf{ek}',\mathsf{vk},\vec{\mathsf{sk}},\vec{\mathsf{ct}}}(1^\kappa) = 1\right],$$

since for any set of queries, the two oracle behaves identically with overwhelming probability over the random choice of τ. Recall that $D_{\mathsf{ek},\mathsf{ek}',\mathsf{vk},\tau}$ deviates only when any of the inputs has τ. Since τ is independently and randomly chosen and hidden in the oracle, any polynomial-size queries will not hit τ with overwhelming probability.

Then we look at the ideal experiment. It is not hard to see:

$$\Pr[\mathcal{B}(\mathsf{Dist}_{\mathbf{Ideal}})] = \Pr\left[\mathcal{B}_{\mathsf{ek},\mathsf{ek}',\mathsf{vk},\vec{\mathsf{sk}},\vec{\mathsf{ct}}}(\hat{C}') = 1\right] \approx \Pr_{\tau\leftarrow\{0,1\}^\kappa}\left[\mathsf{Sim}^{C'(\cdot)}_{\mathcal{B},\mathsf{ek},\mathsf{ek}',\mathsf{vk},\vec{\mathsf{sk}},\vec{\mathsf{ct}}}(1^\kappa) = 1\right].$$

Now we define $\mathcal{B}'$ as a distinguisher that on input $(\mathsf{ek}, \mathsf{ek}', \mathsf{vk}, \vec{\mathsf{sk}}, \vec{\mathsf{ct}})$ and oracle access to $C'(\cdot)$ outputs $\mathsf{Sim}^{C'(\cdot)}_{\mathcal{B},\mathsf{ek},\mathsf{ek}',\mathsf{vk},\vec{\mathsf{sk}},\vec{\mathsf{ct}}}(1^\kappa)$. From the above calculations, we have

$$\Pr[\mathcal{B}'^{C'(\cdot)}(\mathsf{Dist}_{\mathbf{Real}_1}) = 1] - \Pr[\mathcal{B}'^{C'(\cdot)}(\mathsf{Dist}_{\mathbf{Ideal}_1}) = 1] > \epsilon - \mathsf{negl}(\kappa),$$

which completes the proof of the claim.

Then we consider two modified experiments: $\mathbf{Real}_2$ and $\mathbf{Ideal}_2$, where in the two experiments, the key generation algorithm outputs $\mathsf{sk}_f = \mathsf{E}'.\mathsf{Enc}(0)$ along with a signature σ to the adversary. Also similarly, change sk_g to $\mathsf{E}'.\mathsf{Enc}(0)$ along with a signature σ. Then we consider a modified oracle C'', who is hardcoded with all the functions f's (or g's) associated with the sk_f's, so that when the adversary queries with the ciphertext, the oracle evaluates it in the same way as $\mathbf{Real}_1$ and $\mathbf{Ideal}_1$.

It is not hard to see that $\mathbf{Real}_1$ (with the oracle C') and $\mathbf{Real}_2$ (with the oracle C'') are identical if the adversary only queries the sk_f's that have been generated in the key generation algorithms. From the security of the signature scheme, no adversary can query any valid sk_g (with overwhelming probability) such that g was not queried in the key generation phase. (Otherwise, there is a forge of the signature). Similarly, we can argue that $\mathbf{Ideal}_1$ (with oracle C') and $\mathbf{Ideal}_2$ (with the oracle C'') are indistinguishable.

In the rest, we are going to show that $\mathbf{Real}_2$ is indistinguishable from $\mathbf{Ideal}_2$ in the following claim.

Claim C.6 *Assume the encryption scheme* E *is CCA secure. Then no adversary, with oracle access to* C''*, can distinguish* $\textbf{\textit{Real}}_2$ *from* $\textbf{\textit{Ideal}}_2$*.*

We prove this by contradiction. Assume there exist messages $(x_1, y_1, \ldots, x_\ell, y_\ell)$, functions in $\mathcal{F}$ and an adversary D that can distinguish the two distributions. Then we will construct a reduction that breaks the CCA security of E. Then the reduction on receiving pk from the CCA challenger chooses a random τ, and sets the challenge ciphertexts to be either $M_0 = \{(x_i, \text{“1”}, 0^{\mu' q+p+\kappa}), (y_i, \text{“2”}, 0^{\mu' q+p+\kappa})\}_{i\in[\ell]}$, $M_1 = \{(i, \text{“1”}, z_{x_i}, f_{x_i}, \tau), (i, \text{“2”}, z_{y_i}, f_{y_i}, \tau)\}_{i\in[\ell]}$, as the challenge ciphertexts in the $\textbf{Real}_2$ and $\textbf{Ideal}_2$. Then the reduction simulates the distinguisher D and the oracle in the following way. The reduction sets up the parameter param by sampling $(\mathsf{ek}', \mathsf{dk}') \leftarrow \mathsf{E}'.\mathsf{KeyGen}(1^\kappa)$, $(\mathsf{vk}, \mathsf{sk}) \leftarrow \Sigma.\mathsf{KeyGen}$, and embedding the pk from the CCA challenger. Then the reduction generates sk_f's (as encryptions of the zero strings, with a signature). Then he simulate an oracle for the distinguisher D: if D submits a ciphertext that is not equal to the challenge ciphertexts, then the reduction asks the CCA oracle to decrypt and then he can simulate the oracle computation faithfully. If D sends one of the challenge ciphertexts, then the reduction already knows the answer, so he can reply the query faithfully. Thus, it is not hard to see the distinguisher can distinguish encryptions of M_0 from M_1, as D distinguishes $\textbf{Real}_2$ from $\textbf{Ideal}_2$.

Putting things together, we can show that Claim C.4 follows from Claims C.5 and C.6. This completes the proof of the theorem.

■

Theorem C.7 (Theorem 3.5) *Assume* $\mathsf{FE}_{\mathsf{ind}}$ *is an IND-secure public-key binary functional encryption scheme,* commit *is a statistically binding commitment scheme, and* E *is a semantically secure symmetric key encryption scheme. Then the above scheme is* $(0, 1, \mathrm{poly})$ *simulation-secure.*

Proof: We begin by describing the ideal world simulator, $\mathcal{S}(1^\kappa)$. Recall that when the adversary queries $f_1, \ldots, f_\ell$, for each $i \in [\ell]$ the simulator is given $f_i(x, y)$, as well as oracle access to the partial functions $f_i(x, \cdot)$ and $f_i(\cdot, y)$. By our hypothesis, these partial functions are learnable; $\mathcal{S}$ can run the code of the learner L, forwarding oracle queries to his own oracle. Below we let f_x denote the output of the learner L when given oracle access to $f(x, \cdot)$, and similarly for f_y and $f(\cdot, y)$. The full description of our simulator is as follows.

- Setup: $\mathcal{S}$ runs $(\mathsf{param}_{\mathsf{ind}}, \mathsf{msk}_{\mathsf{ind}}) \leftarrow \mathsf{FE}_{\mathsf{ind}}.\mathsf{Setup}(1^\kappa)$. He chooses a random $r \leftarrow \{0,1\}^\kappa$, $\tau \leftarrow \mathsf{E}.\mathsf{KeyGen}(1^\kappa)$ and computes $\mathsf{com} = \mathsf{commit}(\tau; r)$. He outputs $\mathsf{param} = (\mathsf{param}_{\mathsf{ind}}, \mathsf{com})$ and $\mathsf{msk} = (\mathsf{msk}_{\mathsf{ind}}, r, \tau)$.
- Enc: For the challenge plaintext with tag “1”, $\mathcal{S}$ samples $\mathsf{FE}_{\mathsf{ind}}.\mathsf{Enc}(\mathsf{msk}_{\mathsf{ind}}, (0^{|x|}, \text{“1”}, r, \tau, 1))$, where r and τ are the values stored in msk. For the challenge with tag “2” he samples $\mathsf{FE}_{\mathsf{ind}}.\mathsf{Enc}(\mathsf{msk}_{\mathsf{ind}}, (0^{|y|}, \text{“2”}, r, \tau, 1))$
- KeyGen: For each key query f, $\mathcal{S}$ computes f_x using his oracle $f(x, \cdot)$. He samples $c_1 = \mathsf{E}.\mathsf{Enc}(\tau, (f_x, Z_x))$ and $c_2 = \mathsf{E}.\mathsf{Enc}(\tau, (f_y, Z_y))$ (recall, $Z_x = Z_y = f(x, y)$ and is given to him by his oracle). He constructs a circuit computing $\hat{f}_{c_1,c_2}$ as described previously, and outputs $\mathsf{FE}_{\mathsf{ind}}.\mathsf{KeyGen}(\mathsf{msk}_{\mathsf{ind}}, \hat{f}_{c_1,c_2})$. Note that because we assume selective-security, he can handle both adaptive and non-adaptive key queries in this way.

We wish to demonstrate that for any adversary challenge pair x, y, and for any set of function queries $f_1, \ldots, f_\ell$, the following two distributions are indistinguishable:

$$\{\mathsf{ct}_x, \mathsf{ct}_y, \mathsf{param}, \mathsf{sk}_{f_1}, \ldots, \mathsf{sk}_{f_\ell}\} \overset{c}{\approx} \{\widetilde{\mathsf{ct}}_x, \widetilde{\mathsf{ct}}_y, \widetilde{\mathsf{param}}, \widetilde{\mathsf{sk}}_{f_1}, \ldots, \widetilde{\mathsf{sk}}_{f_\ell}\}$$

where the second distribution is output by the simulator as previously described, and both distributions are taken over the randomness used to construct each value (the randomness is left implicit). We proceed to prove that this through a sequence of hybrid distributions.

Hybrid 0: This is the first ditribution described above. I.e. it is the output distribution of the real world.

Hybrid 1: In this distribution, we generate param and the ciphertexts as in the real world, but we generate the decryption keys as in the ideal world. Namely, we consider the distribution $\{\mathsf{ct}_x, \mathsf{ct}_y, \mathsf{param}, \widetilde{\mathsf{sk}}_{f_1}, \ldots, \widetilde{\mathsf{sk}}_{f_\ell}\}$. Recall that the difference between simulated decryption keys and real decryption keys lies in the values c_1, c_2 that are used in the description of the function $\hat{f}_{c_1,c_2}$. In the real world, $c_1, c_2 \leftarrow \mathsf{E.Enc}(\tau, 0^{p+\mu'})$, while in the ideal world, $c_1 = \mathsf{E.Enc}(\tau, (f_x, Z_x))$ and $c_2 = \mathsf{E.Enc}(\tau, (f_y, Z_y))$.

Claim C.8 *If there exists an adversary $\mathcal{A}$ that can distinguish Hybrid 0 and Hybrid 1 with advantage ϵ, then there exists an adversary $\mathcal{R}$ attacking the semantic security of E with the same advantage.*

Proof: Let x, y be the challenge pair for which $\mathcal{A}$ gains his advantage. $\mathcal{R}$ begins by running the Setup procedure as in the real world protocol, but he skips the step where he would generate $\tau \leftarrow \mathsf{E.KeyGen}(1^\kappa)$. He outputs param. He uses these parameters to encrypt x and y as in the real world: $\mathsf{ct}_x = \mathsf{FE.Enc}(\mathsf{param}, (x, \text{“1”}, 0^\kappa, 0^\kappa, 0))$ and $\mathsf{ct}_y = \mathsf{FE.Enc}(\mathsf{param}, (y, \text{“2”}, 0^\kappa, 0^\kappa, 0))$. To create $\mathsf{sk}_{\hat{f}_{\bar{c}_1,\bar{c}_2}}$, $\mathcal{R}$ creates challenge plaintext vectors $((x, 0^\kappa, 0^\kappa), (y, 0^\kappa, 0^\kappa))$ and $((0^{|x|}, f_x, Z_x), (0^{|y|}, f_y, Z_y))$ and passes them to his challenger in the CPA game. He receives back challenge ciphertexts $(\bar{c}_1, \bar{c}_2)$ and uses them to complete key generation. If the challenge bit in the CPA game is 0, the simulation is drawn from the same distribution as Hybrid 0, while if it is 1, the simulation is drawn from the same distribution of Hybrid 1. ■

Hybrid 2: In this distribution, we replace param with simulated parameters $\widetilde{\mathsf{param}}$. Namely, we consider the distribution $\{\mathsf{ct}_x, \mathsf{ct}_y, \widetilde{\mathsf{param}}, \widetilde{\mathsf{sk}}_{f_1}, \ldots, \widetilde{\mathsf{sk}}_{f_\ell}\}$. Recall that in the real parameters, there is a commitment $\mathsf{com} = \mathsf{commit}(0^\kappa; r)$, while in the simulated parameters, $\mathsf{com} = \mathsf{commit}(\tau; r)$ where $\tau \leftarrow \mathsf{E.KeyGen}(1^\kappa)$.

Claim C.9 *If there exists an adversary $\mathcal{A}$ the can distinguish Hybrid 1 from Hybrid 2 with advantage ϵ, then there exists an adversary $\mathcal{R}$ attacking the hiding property of commit with the same advantage.*

Proof: $\mathcal{R}$ runs $\tau \leftarrow \mathsf{E.KeyGen}(1^\kappa)$ and submits challenge $(0^\kappa, \tau)$ to his challenger. He receives a commitment com and uses this in his simulation of param. Note that if his challenge is a commitment to 0^κ, param are the public parameters from Hybrid 1, and otherwise they are the public parameters from Hybrid 2. We note that $\mathcal{R}$ does not now know the value r used in com, but this is not needed for the remainder of the simulation, since ct_x and ct_y encrypt 0^κ instead of $r \leftarrow \{0,1\}^\kappa$. ■

Hybrid 3: In this distribution, we replace $\mathsf{ct}_x, \mathsf{ct}_y$ with simulated ciphertexts $\widetilde{\mathsf{ct}}_x, \widetilde{\mathsf{ct}}_y$, resulting in the ideal world distribution.

Claim C.10 *If there exists an adversary $\mathcal{A}$ the can distinguish Hybrid 2 from Hybrid 3 on some inputs x, y with advantage ϵ, then there exists a non-trivial adversary $\mathcal{R}$ attacking selective security of $\mathsf{FE}_{\mathsf{ind}}$ with the same advantage.*

Proof: $\mathcal{R}$ samples $\tau \leftarrow \mathsf{E.KeyGen}(1^\kappa)$ and $r \leftarrow \{0,1\}^\kappa$, and computes $\mathsf{com} = \mathsf{commit}(r; \tau)$. He constructs the following two challenge plaintext pairs, and submits them to his challenger: $((x, \text{“1”}, 0^\kappa, 0^\kappa, 0), (y, \text{“2”}, 0^\kappa, 0^\kappa, 0))$ and $((0^{|x|}, \text{“1”}, r, \tau, 1), (0^{|y|}, \text{“2”}, r, \tau, 1))$. He receives $\mathsf{param}_{\mathsf{ind}}$ and $(\mathsf{ct}^*_x, \mathsf{ct}^*_y)$ in response, sets $\widetilde{\mathsf{param}} = (\mathsf{param}_{\mathsf{ind}}, \mathsf{com})$ and forwards this and the challenge ciphertexts to $\mathcal{A}$. He then uses r and τ to answer key queries as follows. On query f, he computes $c_1 = \mathsf{E.Enc}(\tau, (f_x, Z_x)), c_2 = \mathsf{E.Enc}(\tau, (f_y, Z_y))$, and requests a key for $\hat{f}_{c_1,c_2}$ from his own challenger. He forwards the response to $\mathcal{A}$.

If the challenge bit in the $\mathsf{FE}_{\mathsf{ind}}$ game is 0, the view of $\mathcal{A}$ is distributed just as in Hybrid 2. Otherwise, it is distributed as in Hybrid 3.

To show that $\mathcal{R}$ is non-trivial, we consider inputs $x_1 = (x, \text{“1”}, 0^\kappa, 0^\kappa, 0)$ and $x_2 = (0^{|x|}, \text{“1”}, r, \tau, 1)$, as well as the function $\hat{f}_{c_1,c_2}$. Recall that $c_1 = \mathsf{E.Enc}(\tau, (f_x, Z_x))$ and $c_2 = \mathsf{E.Enc}(\tau, (f_y, Z_y))$. We must show that for every input $\bar{y}$, $\hat{f}_{c_1,c_2}(x_1, \bar{y}) = \hat{f}_{c_1,c_2}(x_2, \bar{y})$.

- If the last bit of $\bar{y}$ is 0, then it follows from the function description that $\hat{f}_{c_1,c_2}(x_1, \bar{y}) = f(x, \bar{y})$, and $\hat{f}_{c_1,c_2}(x_2, \bar{y}) = f_x(\bar{y})$. Since $\mathcal{R}$ has correctly learned $f(x, \cdot)$, f_x correctly compute $f(x, \cdot)$ and these two outputs are equivalent.
- If the last bit of $\bar{y}$ is 1, parse $\bar{y}$ as $(0^{|y|}, \mathtt{tag}_{\bar{y}}, r_{\bar{y}}, \tau_{\bar{y}}, 1)$.
  - If $\mathtt{tag}_{\bar{y}} \neq$ “2”, then $\hat{f}_{c_1,c_2}(x_1, \bar{y}) = \hat{f}_{c_1,c_2}(x_2, \bar{y}) = \perp$.
  - If $\mathsf{commit}(r_{\bar{y}}; \tau_{\bar{y}}) \neq \mathsf{com}$, then $\hat{f}_{c_1,c_2}(x_1, \bar{y}) = \hat{f}_{c_1,c_2}(x_2, \bar{y}) = \perp$.
  - If $\mathsf{commit}(r_{\bar{y}}; \tau_{\bar{y}}) = \mathsf{com}$, then by the statistical binding property of the commitment scheme, it holds that $\mathsf{E.Dec}(\tau_{\bar{y}}; c_2) = (f_y, Z_y)$ (where these are the values previously learned by $\mathcal{R}$ after interacting with his oracles). It follows that $\hat{f}_{c_1,c_2}(x_1, \bar{y}) = f_y(x)$, and $\hat{f}_{c_1,c_2}(x_2, \bar{y}) = Z_x = f(x, y)$. Again, since f_y correctly computes $f(\cdot, y)$, these outputs are equivalent.

An identical analysis follows for inputs $(y, \text{“2”}, 0^\kappa, 0^\kappa, 0)$ and $(0^{|y|}, \text{“2”}, r, \tau, 1)$. We omit the details. ■

■

Theorem C.11 (Theorem 3.7) *Assume* $\mathsf{FE}_{\mathsf{ind}}$ *is an IND-secure binary functional encryption scheme for functionality* $\mathcal{F}$*, and that* E *is a semantically secure symmetric key encryption scheme. Then the above scheme is a* $(1, 1, 1)$*-simulation-secure symmetric-key binary FE scheme for functionality* $\mathcal{F}$*.*

Proof: We begin by describing the ideal world simulator, $\mathcal{S}(1^\kappa)$.

- Setup: $\mathcal{S}$ runs $(\mathsf{param}_{\mathsf{ind}}, \mathsf{msk}_{\mathsf{ind}}) \leftarrow \mathsf{FE}_{\mathsf{ind}}.\mathsf{Setup}(1^\kappa)$. He chooses a random $\tau \leftarrow \mathsf{E.KeyGen}(1^\kappa)$. He outputs $\mathsf{param} = (\mathsf{param}_{\mathsf{ind}})$ and $\mathsf{msk} = (\mathsf{msk}_{\mathsf{ind}}, \tau)$.
- KeyGen: $\mathcal{S}$ chooses $\mathsf{idx}_f \leftarrow \{0,1\}^\kappa$. If the key query is received before the challenge plaintext submission, $\mathcal{S}$ samples $c \leftarrow \mathsf{E.Enc}(\tau, (0^{\mu'}))$. If the key query comes after the challenge plaintext submission, he samples $c \leftarrow \mathsf{E.Enc}(\tau, (Z_x))$ (recall, $Z_x = Z_y = f(x, y)$ and is given to him by his oracle). He constructs a circuit computing $\hat{f}_{c,\mathsf{idx}_f}$ as described previously, and outputs $\mathsf{FE}_{\mathsf{ind}}.\mathsf{KeyGen}(\mathsf{msk}_{\mathsf{ind}}, \hat{f}_{c,\mathsf{idx}_f})$.
- Enc: If there has been a key query for some function f, $\mathcal{S}$ receives $f(x, y)$ from his oracle. He sets $Z_x = Z_y = f(x, y)$. Then, to simulate the challenge ciphertext with tag “1”, $\mathcal{S}$ samples $\mathsf{FE}_{\mathsf{ind}}.\mathsf{Enc}(\mathsf{msk}_{\mathsf{ind}}, (0^{|x|}, \text{“1”}, \mathsf{idx}_f, Z_x, \tau, 1))$, where τ is the value stored in msk, and idx_f is the random value that was assigned to f during key generation. He simulates the ciphertext with tag “2” with $\mathsf{FE}_{\mathsf{ind}}.\mathsf{Enc}(\mathsf{msk}_{\mathsf{ind}}, (0^{|y|}, \text{“2”}, \mathsf{idx}_f, Z_y, \tau, 1))$. If there hasn't been a key query yet, he instead samples $\mathsf{FE}_{\mathsf{ind}}.\mathsf{Enc}(\mathsf{msk}_{\mathsf{ind}}, (0^{|x|}, \text{“1”}, 0^\kappa, 0^{\mu'}, \tau, 1))$. He similarly simulates the challenge ciphertext with tag “2”, outputting either $\mathsf{FE}_{\mathsf{ind}}.\mathsf{Enc}(\mathsf{msk}_{\mathsf{ind}}, (0^{|y|}, \text{“2”}, 0^\kappa, Z_y, \tau, 1))$.

■

D Strong Differing-inputs Obfuscation and Adaptive Security

In this section, we define a stronger notion of differing-inputs obfuscation, and show how this notion can be used to achieve adaptive security.

Definition D.1 *A circuit family $\mathcal{C}$ associated with a sampler* $\mathsf{Sampler}$ *and a function oracle O is said to be a differing-inputs circuit family if for every PPT adversary $\mathcal{A}$ there exists a negligible function* negl *such that*

$$\Pr[C_0(x) \neq C_1(x) : (C_0, C_1, \mathsf{aux}) \leftarrow \mathsf{Sampler}(1^\kappa), x \leftarrow \mathcal{A}^{O(1^\kappa, \mathsf{aux}, \cdot)}(1^\kappa, C_0, C_1)] \leq \mathsf{negl}(\kappa).$$

We now define the notion of differing-inputs obfuscation for a differing-inputs circuit family.

Definition D.2 (Strong Differing-inputs Obfuscators for circuits) *A uniform PPT machine* $\mathsf{di}\mathcal{O}$ *is called a strong Differing-inputs Obfuscator for a differing-inputs circuit family $\mathcal{C} = \{C_\kappa\}$ if the following conditions are satisfied:*

- *(Correctness): For all security parameter κ, all $C \in \mathcal{C}$, all inputs x, we have*

$$\Pr[C'(x) = C(x) : C' \leftarrow \mathsf{di}\mathcal{O}(\kappa, C)] = 1.$$

- *(Polynomial slowdown): There exists a universal polynomial p such that for any circuit C, we have $|C'| \leq p(|C|)$ for all $C' = \mathsf{di}\mathcal{O}(\kappa, C)$ under all random coins.*

- *(Differing-inputs): For any (not necessarily uniform) PPT distinguisher D, there exists a negligible function* negl *such that the following holds: for all security parameters κ, for $(C_0, C_1, \mathsf{aux}) \leftarrow \mathsf{Sampler}(1^\kappa)$, we have that*

$$|\Pr[D^{O(1^\kappa, \mathsf{aux}, \cdot)}(\mathsf{di}\mathcal{O}(\kappa, C_0)) = 1] - \Pr[D^{O(1^\kappa, \mathsf{aux}, \cdot)}(\mathsf{di}\mathcal{O}(\kappa, C_1)) = 1]| \leq \mathsf{negl}(\kappa).$$

D.1 Adaptively Secure Multi-Client FE from Differing-inputs Obfuscation

We now present a multi-client functional encryption (MC-FE) scheme that is adaptively, IND-secure based on strong differing-inputs obfuscation. Let $\mathcal{E} := (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ denote a public-key encryption scheme, NIZK denote a non-interactive zero knowledge proof system, commit be a commitment scheme.

- $\mathsf{Setup}(1^\kappa, n)$: Compute $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$, and $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$. Run $\mathsf{crs} := \mathsf{NIZK.Setup}(1^\kappa)$.

  For $i \in [n]$, choose $\alpha_i, r_i \leftarrow \{0,1\}^\kappa$, and compute $\mathsf{com}_i = \mathsf{commit}(\alpha_i; r_i)$. Set $\mathsf{param} = [\mathsf{crs}, \mathsf{pk}, \mathsf{pk}', \mathsf{com}_1, \ldots, \mathsf{com}_n]$. The secret keys for each user are: $\mathsf{usk}_i := [r_i, \mathsf{param}]$. The master secret key is: $\mathsf{msk} := \big[\mathsf{param}, \mathsf{sk}, \mathsf{sk}', \{\alpha_i, r_i\}_{i \in [n]}\big]$

- $\mathsf{Enc}(\mathsf{usk}_i, x, t)$: For user i to encrypt a message x and a time step t, it computes the following. Choose random ρ and ρ' as the random bits needed for the public-key encryption scheme. Let $c := \mathcal{E}.\mathsf{Enc}(\mathsf{pk}, x; \rho)$ and $c' := \mathcal{E}.\mathsf{Enc}(\mathsf{pk}', x; \rho')$. Let statement $\mathsf{stmt} := (t, i, \alpha_i, c, c')$; let witness $w := (\rho, \rho', x, r_i)$. Let the NP language be defined as in Figure 9. Let $\pi := \mathsf{NIZK.Prove}(\mathsf{crs}, \mathsf{stmt}, w)$. The ciphertext is defined as $\mathsf{ct} := (t, i, \alpha_i, c, c', \pi)$.

- $\mathsf{KeyGen}(\mathsf{msk}, f)$: To generate a token for a function f over n parties' inputs compute token $\mathsf{TK}_f := \mathsf{di}\mathcal{O}(P)$ for a Program P defined as in Figure 10:

- $\mathsf{Dec}(\mathsf{TK}_f, \mathsf{ct}_1, \ldots, \mathsf{ct}_n)$: Interpret TK_f as an obfuscated program. Output $\mathsf{TK}_f(\mathsf{ct}_1, \mathsf{ct}_2, \ldots, \mathsf{ct}_n)$.

Our NP language $L_{\mathsf{pk},\mathsf{pk}',\{\mathsf{com}_i\}_{i\in[n]}}$ is parameterized by $(\mathsf{pk}, \mathsf{pk}', \{\mathsf{com}_i\}_{i\in[n]})$ output by the Setup algorithm as part of the public parameters. A statement of this language is of the format $\mathsf{stmt} := (t, i, \alpha_i, c, c')$, and a witness is of the format $w := (\rho, \rho', x, r_i)$. A statement $\mathsf{stmt} := (t, i, \alpha, c, c') \in L_{\mathsf{pk},\mathsf{pk}',\{\mathsf{com}_i\}_{i\in[n]}}$, iff

$$\exists\, x, (\rho, \rho'), r \quad \text{s.t. } \mathsf{com}_i = \mathsf{commit}(\alpha; r) \wedge (c = \mathcal{E}.\mathsf{Enc}(\mathsf{pk}, x; \rho)) \wedge \left(c' = \mathcal{E}.\mathsf{Enc}(\mathsf{pk}', x; \rho')\right)$$

Figure 9: **NP language** $L_{\mathsf{pk},\mathsf{pk}',\{\mathsf{com}_i\}_{i\in[n]}}$.

Program $P(\mathsf{ct}_1, \mathsf{ct}_2, \ldots, \mathsf{ct}_n)$**:**

Internal hard-coded state: $\mathsf{param} := [\mathsf{crs}, \mathsf{pk}, \mathsf{pk}', \mathsf{com}_1, \ldots, \mathsf{com}_n]$, sk, f

1. For $i \in [n]$, unpack $(t_i, j_i, \alpha_i, c_i, c_i', \pi_i) \leftarrow \mathsf{ct}_i$, and check that $t_1 = t_2 = \cdots = t_n$ and that $j_i = i$.
2. For $i \in [n]$, let $\mathsf{stmt}_i := (t_i, j_i, \alpha_i, c_i, c_i')$ and check that $\mathsf{NIZK.Verify}(\mathsf{crs}, \pi_i, \mathsf{stmt}_i) = 1$.
3. If any of these above checks fail, output $\perp$. Else, for $i \in [n]$, let $x_i \leftarrow \mathcal{E}.\mathsf{Dec}(\mathsf{sk}, c_i)$ and output $f(x_1, x_2, \ldots, x_n)$.

Figure 10: MC-FE: Program P.

Theorem D.3 (MC-FE from $\mathsf{di}\mathcal{O}$**.)** *If* $\mathsf{di}\mathcal{O}$ *is a strong differing-inputs obfuscation,* NIZK *is an adaptive simulation sound non-interactive zero-knowledge proof system, the commitment scheme is computationally hiding, and the encryption scheme is semantically secure and perfectly correct, then the above construction is adaptively IND-secure, as defined in Section 2.3.2.*

Instantiation and efficiency. We can instantiate our scheme using the simulation sound NIZK scheme as constructed by Sahai et al. [30]. We can use the construction described by Garg et al. [17] as a candidate of the strong $\mathsf{di}\mathcal{O}$. This way, the ciphertext is succinct, and is $\mathrm{poly}(\kappa)$ in size. The public parameter size, encryption time are both succinct, i.e. $\mathrm{poly}(\kappa)$, and the decryption time is $O(n + |f|) \cdot \mathrm{poly}(\kappa)$.

Then we are going to prove Theorem D.3.

Proof: To proof the security, we will show Hybrid X and Hybrid Y (defined in the following) are indistinguishable by a series of hybrids. Given any $G, \bar{G}$, we define the following experiments.

Hybrid X: This game is simply the defined above with $b = 0$. In other words, the challenger will encrypt $\vec{x}^*_G$ (in both copies of the ciphertexts c and c') and return them to the adversary.

Hybrid 1: In this hybrid, when the challenger answers an adversary's query, the crs generation is simulated, allowing the simulator to obtain a simulated $\widetilde{\mathsf{crs}}$ and a trapdoor trap for simulating fake proofs.

Hybrid 2: In this hybrid, we switch the commitments to $\mathsf{com}(0^\kappa)$ for all $i \in G$. The ciphertexts are still of the form $(t, i, \alpha_i, c, c', \pi)$, yet proofs π in the ciphertexts of the set G are generated by the simulator (since they prove false statements, i.e. the commitments are not for α_i) as Hybrid 1.

Hybrid 3: In the challenge time step, we will switch the challenge ciphertexts to $\mathcal{E}.\mathsf{Enc}(\mathsf{pk}, x_i; \rho), \mathcal{E}.\mathsf{Enc}(\mathsf{pk}', y_i; \rho')$ for all $i \in G$. The NIZK proofs for the challenge ciphertexts are simulated.

Hybrid $4^{(i)}$: In this hybrid, we let P_j denote the program constructed from the jth key query of the adversary, and for $j < i$, we switch the obfuscated program to $\mathsf{di}\mathcal{O}(P'_j)$, where P'_j is similar to P_j except it is hardwired with sk', and decrypts the second copy of each ciphertext pair and computes accordingly. Note that Hybrid $4^{(1)}$ is the same as Hybrid 3.

Hybrid Y': In this hybrid, the challenger encrypts $\vec{y}_G$ in both copies of the challenge ciphertexts. $\mathsf{di}\mathcal{O}(P')$ is given to the adversary which uses sk' to decrypt.

Hybrid Y: This game is simply the defined above with $b = 1$. In other words, the challenger will encrypt $\vec{y}^*_G$ (in both copies of the ciphertexts c and c') and return them to the adversary. $\mathsf{di}\mathcal{O}(P)$ is given to the adversary which uses sk to decrypt.

Then we show the indistinguishability of the hybrids by the following claims:

Claim D.4 *If the NIZK is computationally zero knowledge, then Hybrid X and Hybrid 1 are computationally indistinguishable.*

Claim D.5 *If the commitment is computationally hiding, then Hybrid 1 and Hybrid 2 are computationally indistinguishable.*

Claim D.6 *If the encryption scheme* E *is semantically secure, Hybrid 2 and Hybrid 3 are computationally indistinguishable.*

The above three claims can be proved by somewhat trivial reductions. The reductions are very similar to those presented previously, so we omit repeating them here.

Claim D.7 *Let ℓ denote the number of key queries made by the adversary. If the security of the obfuscation $\mathsf{di}\mathcal{O}$ and the simulation soundness of the NIZK hold, then for $0 < i < \ell$, Hybrid $4^{(i)}$ and Hybrid $4^{(i+1)}$ are computationally indistinguishable.*

Let param be a set of parameters sampled as in Hybrids $4^{(i)}$ and $4^{(i+1)}$. Namely, after running $(\widetilde{\mathsf{crs}}, \mathsf{trap}) \leftarrow \mathsf{NIZK.SimSetup}(1^\kappa)$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$, $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$, and for $i \in [n]$, $\mathsf{com}_i = \mathsf{commit}(0^\kappa; r_i)$, let $\mathsf{param} = (\widetilde{\mathsf{crs}}, \mathsf{pk}, \mathsf{pk}', \{\mathsf{com}_i\})$. Let $\mathcal{A}$ be some adversary that distinguishes the two hybrid worlds, and let f_i denote the ith key query of $\mathcal{A}$ given param and key the preceeding query responses $\mathsf{sk}_{f_1}, \ldots, \mathsf{sk}_{f_{i-1}}$. Then we define a sampler $\mathsf{Sampler}$ and an associated function oracle O as follows. $\mathsf{Sampler}$ samples parameters exactly as just described, and then outputs $(P_{f_i}, P'_{f_i}, \mathsf{aux})$, where $\mathsf{aux} = (\mathsf{pk}, \mathsf{pk}', \widetilde{\mathsf{crs}}, \{\mathsf{com}_i\}, \mathsf{sk}, \mathsf{sk}', \mathsf{trap})$, and the two programs are as described in Figure 10, differing only in that the first has sk hardcoded in its state while the second has sk' hardcoded in its state. The function oracle $O(1^\kappa, \mathsf{aux}, \cdot)$ responds to queries in the following manner. On input "params", it outputs $(\mathsf{pk}, \mathsf{pk}', \widetilde{\mathsf{crs}}, \{\mathsf{com}_i\}, \mathsf{sk}, \mathsf{sk}')$ (while keeping trap hidden). On input $\vec{x}_G, \vec{y}_G$ such that $f_i(\vec{x}_G, \cdots) = f_i(\vec{y}_G, \cdots)$, O samples random ρ_j and ρ'_j for $j \in G$, to be used as the random bits in the public-key encryption scheme. He computes $c_j := \mathcal{E}.\mathsf{Enc}(\mathsf{pk}, x_j; \rho_j)$ and $c'_j := \mathcal{E}.\mathsf{Enc}(\mathsf{pk}', y_j; \rho'_j)$. Let statement $\mathsf{stmt} := (t, i, \alpha_i, c_j, c'_j)$. He proves that stmt is in the NP language defined in Figure 9 by computing $\pi_j := \mathsf{NIZK.SimProve}(\widetilde{\mathsf{crs}}, \mathsf{stmt}, \mathsf{trap})$. Finally, he outputs $\{c_j, c'_j, \pi_j\}_{j \in G}$. On any other inputs, O outputs $\perp$.

We first show that the circuit family associated with the $\mathsf{Sampler}$ and O defined above is differing-input secure. This follows from the simulation soundness of the NIZK scheme. The circuits P_{f_i} and P'_{f_i} both output $\perp$ anytime the proof π does not verify for statement stmt. If the proof verifies, then either stmt is in the language, in which case the circuits clearly have the same output, or stmt is not in the language. In this latter case, if $\{c_j, c'_j, \pi_j\}_{j \in G}$ were output by O, then by the definition of O, we know

that $f_i(\vec{x}_G, \cdots) = f_i(\vec{y}_G, \cdots)$ and the two programs have the same output. If an adversary can find a false statement with a valid proof that was not output by O, we can use this adversary to break the simulation soundness of the NIZK.

Suppose there exists an adversary $\mathcal{A}$ that distinguishes Hybrid $4^{(i)}$ from Hybrid $4^{(i+1)}$. Then we can construct a distinguisher D that distinguishes $\mathsf{diO}(P_{f_i})$ from $\mathsf{diO}(P'_{f_i})$. D on input either $\mathsf{diO}(P_{f_i})$ or $\mathsf{diO}(P'_{f_i})$, queries O with "params" and receives $(\mathsf{pk}, \mathsf{pk}', \widetilde{\mathsf{crs}}, \{\mathsf{com}_i\}, \mathsf{sk}, \mathsf{sk}')$; he uses these values to simulate the view of $\mathcal{A}$ as follows. He simulates Setup with $\widetilde{\mathsf{crs}}$, pk, pk' and $\{\mathsf{com}_i\}$. For $j < i$, he simulates key query f_j by computing $\mathsf{diO}(P'_{f_j})$, and for $j > i$, he simulates that by by computing $\mathsf{diO}(P_{f_j})$. We note that he has sk and sk', which suffice for this; he does not need trap to construct the obfuscation. To reply to key query sk_{f_i}, he just forwards his input from the challenger, i.e. either $\mathsf{diO}(P_{f_i})$ or $\mathsf{diO}(P'_{f_i})$. When he receives (legal) challenge plaintexts $\vec{x}_G$ and $\vec{y}_G$ from $\mathcal{A}$, D queries these values to O and receives back $\{c_j, c'_j, \pi_j\}_{j \in G}$. The reader can verify that $\mathcal{A}$'s view is perfectly simulated according to either Hybrid $4^{(i)}$ or Hybrid $4^{(i+1)}$, depending on the challenge D receives from his challenger. Thus, D can distinguish $\mathsf{diO}(P_{f_i})$ from $\mathsf{diO}(P'_{f_i})$. This contradicts the fact that the family is strong differing-inputs as argued above. This completes the proof of the claim.

The proofs for the remaining half of the hybrid sequence (from Hybrid 4 to Hybrid Y), are exactly symmetric to the first half of the hybrid sequence (from Hybrid X to Hybrid 4) described above.

■